certiprof®

ARTIFICIAL INTELLIGENCE
EXPERT CERTIFICATE

CAIEC® Versión 062021

ARTIFICIAL INTELLIGENCE EXPERT CERTIFICATE CAIEC®

CAIEC® Versión 062021

¿Quién es Certiprof®?

Certiprof® es una entidad certificadora fundada en los Estados Unidos en 2015, ubicada actualmente en Sunrise, Florida.

Nuestra filosofía se basa en la creación de conocimiento en comunidad y para ello su red colaborativa está conformada por:

- **Nuestros Lifelong Learners (LLL)** se identifican como Aprendices Continuos, lo que demuestra su compromiso inquebrantable con el aprendizaje permanente, que es de vital importancia en el mundo digital en constante cambio y expansión de hoy. Independientemente de si ganan o no el examen.
- Las universidades, centros de formación, y facilitadores en todo el mundo forman parte de nuestra red de aliados **CPLS (Certified Partner For Learning Solutions).**
- **Los autores (co-creadores)** son expertos de la industria o practicantes que, con su conocimiento, desarrollan contenidos para la creación de nuevas certificaciones que respondan a las necesidades de la industria.
- **Personal Interno:** Nuestro equipo distribuido con operaciones en India, Brasil, Colombia y Estados Unidos está a cargo de superar obstáculos, encontrar soluciones y entregar resultados excepcionales.

Nuestras Afiliaciones

Memberships

itcc
IT CERTIFICATION COUNCIL

Agile Alliance®

ANSI MEMBER

Digital badges issued by

Credly
by Pearson

IT Certification Council - ITCC

Certiprof® es un miembro activo de ITCC.

Una de las ventajas de hacer parte del ITCC es como líderes del sector colaboran entre sí en un formato abierto para explorar maneras nuevas o diferentes formas de hacer negocios que inspiran y fomentan la innovación, estableciendo y compartiendo buenas prácticas que nos permiten extender ese conocimiento a nuestra comunidad.

Certiprof ha contribuido a la elaboración de documentos blancos en el Career Path Ways Taskforce, un grupo de trabajo que se implementó internamente para ofrecer a los estudiantes la oportunidad de saber qué camino tomar después de una certificación.

Algunos de los miembros del ITCC

- **IBM**
- **CISCO**
- **ADOBE**
- **AWS**
- **SAP**
- **GOOGLE**
- **ISACA**

Agile Alliance

Certiprof® es un miembro corporativo de Agile Alliance.

Al unirnos al programa corporativo Agile Alliance, continuamos empoderando a las personas ayudándolas a alcanzar su potencial a través de la educación. Cada día, brindamos más herramientas y recursos que permiten a nuestros socios formar profesionales que buscan mejorar su desarrollo profesional y sus habilidades.

https://www.agilealliance.org/organizations/certiprof/

Credly

Esta alianza permite que las personas y empresas certificadas con Certiprof® cuenten con una distinción a nivel mundial a través de un distintivo digital.

Credly es el emisor de insignias más importante del mundo y empresas líderes en tecnología como IBM, Microsoft, PMI,Nokia, la Universidad de Stanford, entre otras, emiten sus insignias con Credly.

Empresas que emiten insignias de validación de conocimiento con Credly:

- **IBM**
- **Microsoft**
- **PMI**
- **Universidad de Stanford**
- **Certiprof**

Insignias Digitales

Insignias Digitales:
¿Qué Son?

- Según el estudio del IT Certification Council (ITCC), años atrás, la gente sabía muy poco sobre las insignias digitales. Hoy, grandes empresas e instituciones educativas de todo el mundo expiden insignias.
- Las insignias digitales contienen metadatos detallados sobre quién las ha obtenido, las competencias requeridas y la organización que las ha expedido. Algunas insignias incluso están vinculadas a las actividades necesarias para obtenerlas.
- Para las empresas e instituciones educativas, las insignias y la información que proporcionan son tan importantes que muchas decisiones, como las de contratación o admisión, se basan en los datos que aportan.

¿Por qué son importantes?

- **Facilidad de Compartir y Verificar Logros:**
  - Las insignias digitales permiten a los profesionales mostrar y verificar sus logros de manera instantánea y global. Según un informe de Credly, **los perfiles de LinkedIn con insignias digitales reciben un 40% más de atención por parte de reclutadores y empleadores.**

- **Visibilidad en Plataformas Digitales:**

  En una encuesta realizada por Pearson y Credly, el **85%** de los usuarios que obtuvieron insignias digitales **las compartieron en LinkedIn,** y el **75%** reportó que esto mejoró su **credibilidad profesional en sus redes.** Además, el **76%** de los empleadores encuestados afirmó que las insignias digitales les ayudan a identificar rápidamente habilidades específicas.

¿Por qué son importantes?

- **Impacto en la Contratación:**

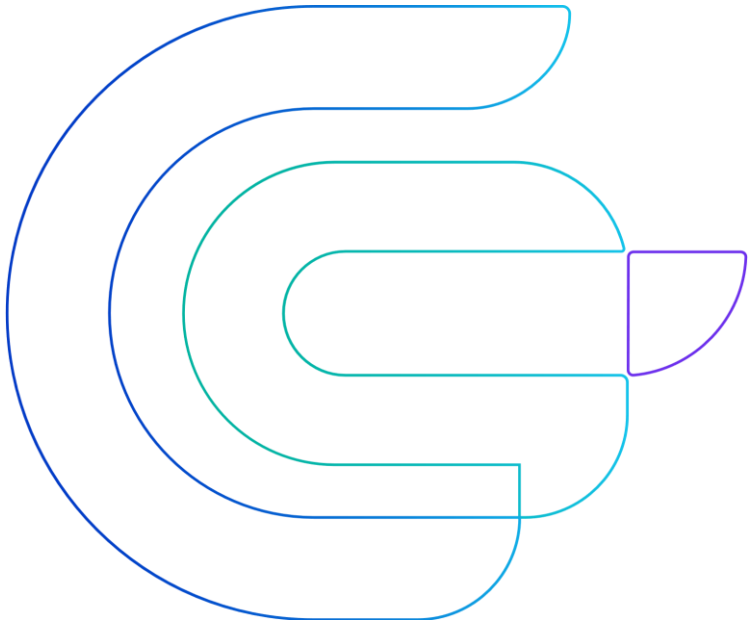
  Un estudio de la **Asociación Internacional de Gestión de Proyectos (PMI)** encontró que los candidatos que muestran insignias digitales de gestión de proyectos tienen **un 60%** más de probabilidades de ser contratados en comparación con aquellos que solo mencionan sus habilidades sin verificación digital.

¿Por qué son importantes?

- **Empoderamiento de la Marca Personal:**

La visibilidad y verificación instantánea proporcionada por las insignias digitales permiten a los profesionales no solo demostrar sus habilidades, sino también construir una marca personal fuerte. Según un estudio de LinkedIn, los profesionales que utilizan insignias digitales tienen un 24% más de probabilidades de avanzar en sus carreras.

La certificación y las insignias digitales no son solo una validación del conocimiento, sino también una herramienta poderosa para la mejora continua y la empleabilidad. En un mundo donde el aprendizaje permanente se ha convertido en la norma, estas credenciales son clave para el desarrollo profesional y la competitividad en el mercado laboral global.



- No todas las insignias son iguales, y en **Certiprof,** estamos comprometidos con ofrecerte más que un simple reconocimiento digital. Al obtener una insignia emitida por certiprof, estarás recibiendo una validación de tu conocimiento respaldada por una de las entidades líderes en certificación profesional a nivel mundial.

- **Da el siguiente paso y obtén la insignia que te abrirá puertas y te posicionará como un experto en tu campo.**

¿Por qué es importante obtener su certificado?

- **Prueba de experiencia:** Su certificado es un reconocimiento formal de las habilidades y conocimientos que ha adquirido. Sirve como prueba verificable de sus cualificaciones y demuestra su compromiso con la excelencia en su campo.

- **Credibilidad y reconocimiento:** En el competitivo mercado laboral actual, las empresas y los compañeros valoran las credenciales que le distinguen de los demás. Un certificado de una institución reconocida, como Certiprof, proporciona credibilidad instantánea e impulsa su reputación profesional.

- **Avance profesional:** Tener tu certificado puede abrirte las puertas a nuevas oportunidades. Ya se trate de un ascenso, un aumento de sueldo o un nuevo puesto de trabajo, las certificaciones son un factor diferenciador clave que los empleadores tienen en cuenta a la hora de evaluar a los candidatos.

¿Por qué es importante obtener su certificado?

- **Oportunidades de establecer contactos:** Poseer un certificado le conecta con una red de profesionales certificados. Muchas organizaciones cuentan con grupos de antiguos alumnos o de trabajo en red en los que puede compartir experiencias, intercambiar ideas y ampliar su círculo profesional.

- **Logro personal:** Obtener una certificación es un logro importante, y su certificado es un recordatorio tangible del trabajo duro, la dedicación y el progreso que ha realizado. Es algo de lo que puede sentirse orgulloso y mostrar a los demás.

Insignia

Earn this Badge

Artificial Intelligence Expert Certificate - CAIEC®

Issued by Certiprof

Artificial Intelligence Expert certification holders understand the fundamental keys of the deep learning approach, practical bases of architecture, and convergence of neural networks. They have demonstrated how to master the methodologies for setting up neural networks, the strengths, and limitations of existing tools and libraries.

Learn more

Certification Paid

Skills

AI | Data Analysts | Data Scientists | Data Stewards | Deep Learning | Expert | Machine Learning

https://www.credly.com/org/certiprof/badge/artificial-intelligence-expert-certificate-caiec.1

Aprendizaje Permanente

- Certiprof ha creado una insignia especial para reconocer a los aprendices constantes.
- Para el 2024, se han emitido más de 1,000,000 de estas insignias en más de 11 idiomas.

Propósito y Filosofía

- Esta insignia está destinada a personas que creen firmemente en que la educación puede cambiar vidas y transformar el mundo.
- La filosofía detrás de la insignia es promover el compromiso con el aprendizaje continuo a lo largo de la vida.

Acceso y Obtención de la Insignia

- La insignia de Lifelong Learning se entrega sin costo a aquellos que se identifican con este enfoque de aprendizaje.
- Cualquier persona que se considere un aprendiz constante puede reclamar su insignia visitando:

https://certiprof.com/pages/certiprof-lifelong-learning

COMPARTE Y VERIFICA TUS LOGROS DE APRENDIZAJE FÁCILMENTE

#CAIEC #certiprof

certiprof®

I. Fundamentos del Aprendizaje Profundo

I.1 Representación de Redes Neuronales

I.1.1 Modelos no lineales

La inspiración de las redes neuronales artificiales (o "redes neuronales" para abreviar) proviene en parte de las redes neuronales biológicas. Las células de la mayoría de los cerebros (incluido el nuestro) se conectan y trabajan juntas. Llamamos **neurona** a cada una de estas células de una red neuronal. Las neuronas del cerebro humano se comunican mediante el intercambio de señales eléctricas.

Los modelos de redes neuronales se inspiran en la estructura de las neuronas de nuestro cerebro y en la forma en que se transmiten los mensajes. Sin embargo, las similitudes entre las redes neuronales biológicas y las redes neuronales artificiales terminan aquí.

Una **red neuronal profunda** es un tipo específico de red neuronal que destaca por captar relaciones no lineales en los datos. Las redes neuronales profundas han superado muchos puntos de referencia en la clasificación de audio e imágenes. Anteriormente, se solían utilizar modelos lineales con transformaciones no lineales descubiertas meticulosamente a mano.

En esta lección, exploraremos las redes neuronales profundas. Estos son algunos de los aspectos que puedes esperar al final de esta lección:

I.1 Representación de Redes Neuronales

- Cómo representar visualmente las redes neuronales
- Cómo implementar la regresión lineal y logística como redes neuronales
- Las diferencias entre las funciones de activación no lineales

Para aprovechar al máximo esta lección, necesitarás conocer las bibliotecas NumPy, sklearn y pandas. También necesitarás sentirte cómodo programando en Python. Nos basaremos en la estadística, el cálculo y el álgebra lineal. Se debe entender el flujo de trabajo del aprendizaje automático tradicional, así como los modelos de regresión lineal y logística.

I.1 Representación de Redes Neuronales

I.1.2 Grafos

Solemos representar las redes neuronales como **grafos**. Un grafo es una estructura de datos formada por **nodos** (representados como círculos)y conectados por **aristas** (representadas como líneas).

I.1 Representación de Redes Neuronales

Solemos utilizar los grafos para representar las relaciones o vínculos entre los componentes de un sistema. Por ejemplo, el **Grafo Social de Facebook** describe la interconexión de todos los usuarios de Facebook (y este grafo cambia constantemente a medida que los usuarios añaden y eliminan amigos). Google Maps utiliza grafos para representar ubicaciones en el mundo físico como nodos y carreteras como aristas.

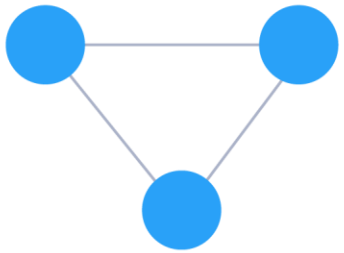
Grafo Social de Facebook

Google Maps

I.1 Representación de Redes Neuronales

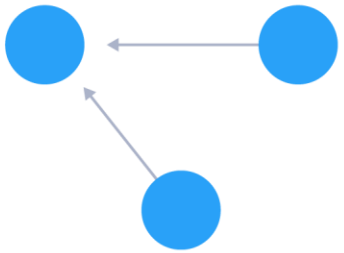
Los grafos son una estructura de datos muy flexible; incluso se puede representar una lista de valores como un grafo. A menudo clasificamos los grafos por sus propiedades, que actúan como restricciones. Puedes leer sobre las diferentes formas de clasificar los grafos **en Wikipedia.**

Grafo no dirigido



Los vértices **no tienen dirección**

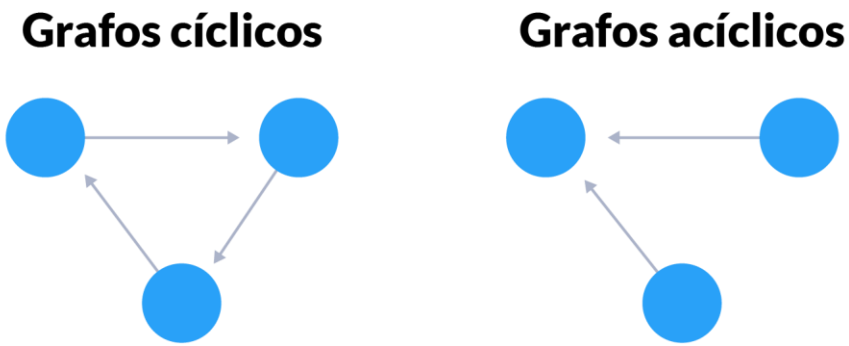
Grafo dirigido



Los vértices **tienen dirección**

I.1 Representación de Redes Neuronales

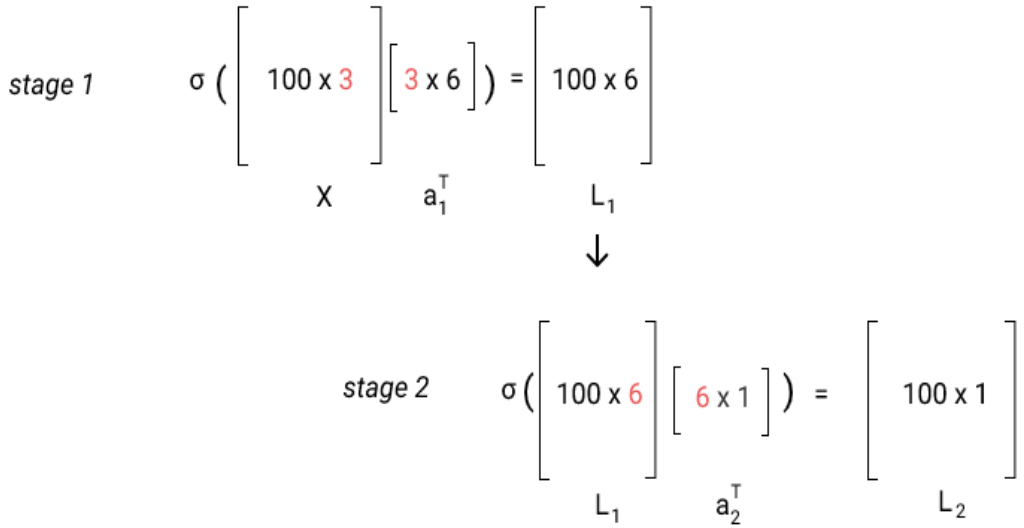
Una forma de clasificar los grafos es la presencia de la dirección de las aristas. Dentro de los grafos dirigidos, los grafos son cíclicos o acíclicos.



I.1 Representación de Redes Neuronales

I.1.3 Grafos Computacionales

Los grafos proporcionan un modelo mental para pensar en una clase específica de modelos: los que consisten en una serie de funciones ejecutadas en un orden específico. En el contexto de las redes neuronales, los grafos nos ayudan a expresar la ejecución de una serie de funciones en sucesión.



I.1 Representación de Redes Neuronales

Esta línea de producción tiene 2 etapas de funciones que se suceden en secuencia:

- En la primera etapa, se calcula L1:

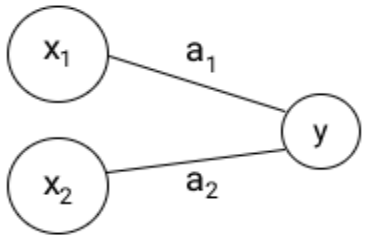
$$L_1 = Xa_1{}^T$$

- En la primera etapa, se calcula L2:

$$L_2 = L_1a_2{}^T$$

La segunda etapa no puede ocurrir sin la primera porque L1 es una entrada para la segunda etapa. El cómputo sucesivo de funciones está en el corazón de los modelos de redes neuronales. Se trata de un grafo computacional. **Un grafo computacional** utiliza nodos para describir variables y aristas para describir la combinación de variables. He aquí un ejemplo sencillo:

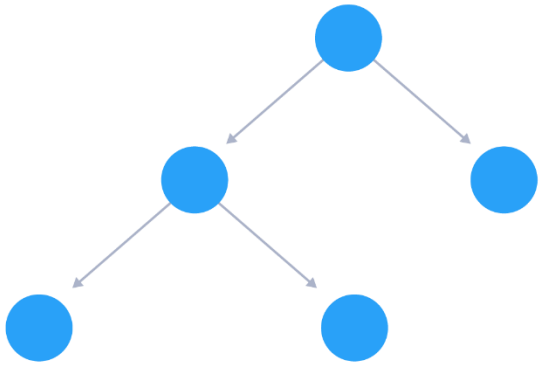
$$y = a_1x_1 + a_2x_2$$



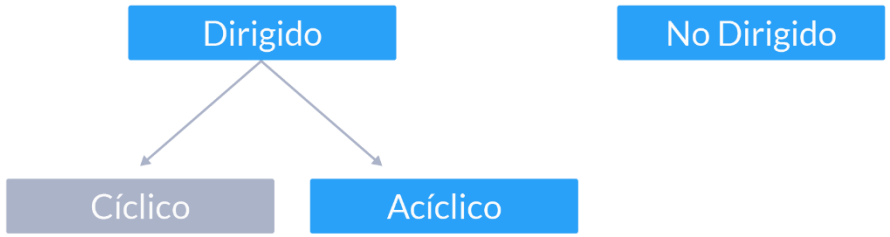
I.1 Representación de Redes Neuronales

El grafo computacional es una representación poderosa porque nos permite representar modelos con muchas capas de anidamiento. De hecho, un árbol de decisión es realmente un tipo específico de grafos computacionales. No hay una forma compacta de expresar un modelo de árbol de decisión utilizando sólo ecuaciones y notación algebraica estándar.

Árbol de Decisiones



Árbol representando algunos tipos de grafos



Para entender mejor esta representación, representaremos un modelo de regresión lineal utilizando la notación de red neuronal. Esto le ayudará a aprender esta representación única, y nos permitirá explorar parte de la terminología de las redes neuronales.

I.1 Representación de Redes Neuronales

I.1.4 Red neuronal frente a regresión lineal

$\hat{y}$ he aquí una representación de un modelo de regression lineal:

$$\hat{y} = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$$

- $\hat{y}$ a_0 representa el intercepto (también conocido como **el sesgo**)
- $\hat{y}$ a_1 hasta a_n representan los pesos del modelo entrenado
- $\hat{y}$ x_1 hasta x_n representan las características
- $\hat{y}$ representan el valor previsto

El primer paso es reescribir este modelo utilizando la notación del álgebra lineal, como un producto de dos vectores.

$$Xa^T = \hat{y}$$

I.1 Representación de Redes Neuronales

He aquí un ejemplo de este modelo:

$$\begin{bmatrix} 1 & 2.3 & 0.2 \\ 1 & 3.1 & 0.9 \\ 1 & 1.1 & 0.5 \end{bmatrix} \begin{bmatrix} 0.9 \\ 0.7 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 2.61 \\ 3.52 \\ 1.92 \end{bmatrix}$$

$$X \qquad a^T \qquad \hat{y}$$

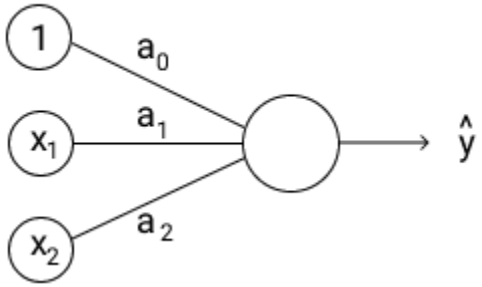
Representación de Redes Neuronales

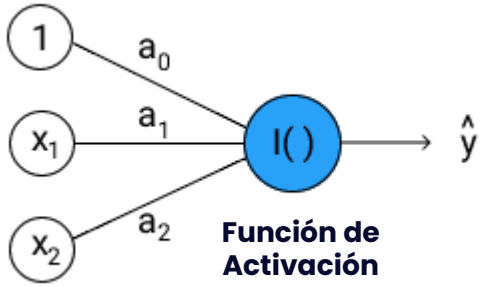
En la representación de la red neuronal de este modelo, vemos lo siguiente:

- **Una neurona de entrada** representa cada columna de características en un conjunto de datos
- Cada valor de peso se representa como una flecha desde la columna de características que multiplica hasta la neurona de salida

Las neuronas y las flechas representan la suma ponderada, que es la combinación de las columnas de características y los pesos.

I.1 Representación de Redes Neuronales





Inspirada en las redes neuronales biológicas, una función de activación determina si la neurona se dispara o no. En un modelo de red neuronal, la función de activación transforma la suma ponderada de los valores de entrada. Para esta red, la función de activación es la **función de identidad**. La función identidad devuelve el mismo valor que se pasó en lo siguiente: f(x)=x

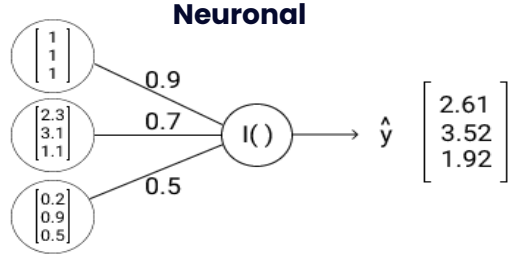
Aunque la función de activación no es interesante para una red que realiza una regresión lineal, es útil para la regresión logística y las redes más complejas. He aquí una comparación de ambas representaciones del mismo modelo de regresión lineal:

Algebra lineal

$$I\left(\begin{bmatrix}1 & 2.3 & 0.2\\ 1 & 3.1 & 0.9\\ 1 & 1.1 & 0.5\end{bmatrix}\begin{bmatrix}0.9\\ 0.7\\ 0.5\end{bmatrix}\right)=\begin{bmatrix}2.61\\ 3.52\\ 1.92\end{bmatrix}$$

X $\quad$ a^T $\quad$ $\hat{y}$

Red Neuronal



I.1 Representación de Redes Neuronales

I.1.5 Manipulación de datos de regresión

Empezaremos a trabajar con datos que generaremos nosotros mismos en lugar de un conjunto de datos externo. Generar los datos nosotros mismos nos da más control sobre las propiedades del conjunto de datos (por ejemplo, como el número de características, las observaciones y el ruido en las características). Los conjuntos de datos que creamos en los que las redes neuronales destacan contienen la misma no linealidad que los conjuntos de datos del mundo real, por lo que podemos aplicar lo que aprendemos aquí.

Scikit-learn contiene las siguientes funciones convenientes para generar datos:

- **sklearn.datasets.make_regression()**
- **sklearn.datasets.make_classification()**
- **sklearn.datasets.make_moons()**

El siguiente código genera un conjunto de datos de regresión con 3 características, 1000 observaciones y una semilla aleatoria de 1:

```
from sklearn.datasets import make_regression
data = make_regression(n_samples=1000, n_features=3,
random_state=1)
```


I.1 Representación de Redes Neuronales

La función make_regression() devuelve una tupla de dos objetos NumPy

```
>> print(type(data))
tuple
```

Las características están en la primera matriz NumPy, y las etiquetas están en la segunda matriz NumPy:

```
# Features
print(data[0])
>> array([[ 0.93514778,  1.81252782,  0.14010988],
       [-3.06414136,  0.11537031,  0.31742716],
       [-0.42914228,  1.20845633,  1.1157018 ],
       ...,
       [-0.42109689,  1.01057371,  0.20722995],
       [ 2.18697965,  0.44136444, -0.10015523],
       [ 0.440956  ,  0.32948997, -0.29257894]])

# First row
data[0][0]
>>array([ 0.93514778,  1.81252782,  0.14010988])
```

```
# Labels
data[1]
>> array([  2.55521349e+02,  -2.24416730e+02,   1.77695808e+02,
        -1.78288470e+02,  -6.31736749e+01,  -5.52369226e+01,
        -2.33255554e+01,  -8.81410996e+01,  -1.75571964e+02,
         1.06048917e+01,   7.07568627e+01,   2.86371625e+02,
         ...,
        -7.38320267e+01,  -2.38437890e+02,  -1.23449719e+02,
         3.36130733e+01,  -2.67823475e+02,   1.21279169e+00,
        -2.62440408e+02,   1.32486453e+02,  -1.93414037e+02,
        -2.75702376e+01,  -1.00678877e+01,   2.05169507e+02,
        -1.52978767e+02,   1.18361239e+01,  -2.97505169e+02,
        -2.40169605e+02,   7.33158364e+01,   2.18888903e+02,
         3.92751308e+01])

# First label
data[1][0]
>> 255.52134901495128
```


I.1 Representación de Redes Neuronales

Podemos entonces utilizar el constructor pandas.DataFrame() para crear DataFrames :

```
features = pd.DataFrame(data[0])
```

Vamos a generar algunos datos para la red que estamos construyendo.

Instrucciones

1. Gener un grupo de datos para regresion que incluya lo siguiente:
   - Exactamente 100 observaciones
   - Exactamente 3 caracteristicas
   - La muestra aleatoria 1
2. Convertir la matriz NumPy de características generadas en un DataFrame de pandas, y asignar a las **características (features)**
3. Convertir el array NumPy de etiquetas generadas en una serie de pandas y asignar a las **etiquetas (labels)**

I.1 Representación de Redes Neuronales

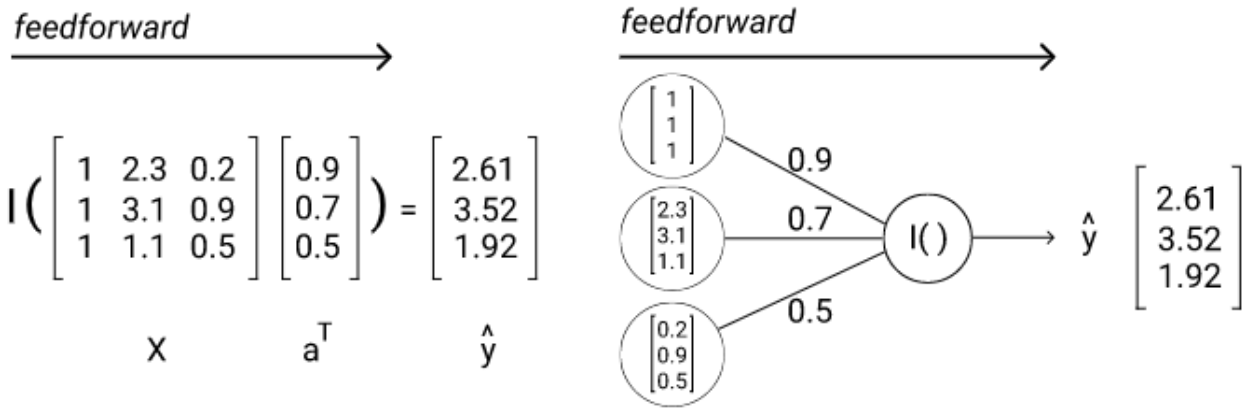
Soluciones

```
from sklearn.datasets import make_regression
import pandas as pd
data = make_regression(n_samples=100, n_features=3, random_state=1)

features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
```

I.1.6 Ajuste de una Red Neuronal de Regresión Lineal

Dado que las entradas de una capa de neuronas alimentan la siguiente capa de la única neurona de salida, llamamos a esta red una red **feedforward**. En el lenguaje de los grafos, una red de alimentación es un **grafo dirigido y acíclico.**

I.1 Representación de Redes Neuronales



Cómo montar una red

A continuación se presentan dos enfoques diferentes para entrenar un modelo de regresión lineal:

- Descenso gradual
- Mínimos cuadrados ordinarios

I.1 Representación de Redes Neuronales

El descenso de gradiente es la técnica más común para ajustar modelos de redes neuronales. En esta lección nos basaremos en la implementación de scikit-learn del descenso de gradiente.

Esta implementación está en la clase **SGDRegressor**. La utilizaremos de la misma manera que lo hacemos con la clase **LinearRegression**:

```
from sklearn.linear_model import SGDRegressor
lr = SGDRegressor()
lr.fit(X,y)
```

Ahora tenemos todo lo que necesitamos para implementar esta red. Como nos estamos centrando en la construcción de la intuición, entrenaremos y probaremos con el mismo conjunto de datos. En los escenarios de la vida real, siempre se quiere utilizar una técnica de validación cruzada de algún tipo.

I.1 Representación de Redes Neuronales

Instrucciones

1. Añade una columna llamada **bias** que contenga el valor 1 para cada fila del DataFrame de características
2. Importe SGDRegressor de sklearn.linear_model
3. Defina dos funciones:
   - train(features, labels): toma las features (características) DataFrame y los labels (etiquetas) de las series y realiza el ajuste del modelo
   - Utilice la clase **SGDRegressor** de scikit-learn para manejar el ajuste del modelo
   - Esta función debería devolver sólo una matriz NumPy 1D de pesos para el modelo de regresión lineal.
   - feedforward(features, weights): toma el DataFrame de features (características) y el array NumPy de weights (pesos)
   - Realiza la multiplicación matricial entre los features (100 filas por 4 columnas) y los weights (4 filas por 1 columna) y asigna el resultado a las predicciones
   - Predicciones de retorno (Return predictions). Omitiremos la implementación de la función de identidad ya que simplemente devuelve el mismo valor que se pasó
4. Descomente el código que hemos añadido para usted y ejecute las funciones train() y feedforward(). Las predicciones finales estarán en linear_predictions

I.1 Representación de Redes Neuronales

Soluciones

```
from sklearn.datasets import make_regression
import numpy as np
data = make_regression(n_samples=100, n_features=3, random_state=1)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])

# Uncomment after you've implemented train() and feedforward()
# train_weights = train(features, labels)
# linear_predictions = feedforward(features, train_weights)
from sklearn.linear_model import SGDRegressor
features["bias"] = 1

def train(features, labels):
    lr = SGDRegressor()
    lr.fit(features, labels)
    weights = lr.coef_
    return weights

def feedforward(features, weights):
    predictions = np.dot(features, weights.T)
    return predictions

train_weights = train(features, labels)
linear_predictions = feedforward(features, train_weights)
```


I.2.7 Generación de Datos de Clasificación

Para generar un conjunto de datos apto para la clasificación, podemos utilizar la función make_classification() de scikit-learn.

El siguiente código genera un conjunto de datos de clasificación con 4 características, 1000 observaciones y una semilla aleatoria de 1:

```
from sklearn.datasets import make_classification
class_data = make_classification(n_samples=1000, n_features=4, random_state=1)
```

La función **make_classification()** devuelve una tupla de dos objetos NumPy.

```
>> print(type(class_data))
tuple
```

I.1 Representación de Redes Neuronales

Al igual que con los datos generados por make_regression(), las características están en la primera matriz NumPy, y las etiquetas están en la segunda matriz NumPy:

```
# Features
class_features = class_data[0]
# Labels
class_labels = class_data[1]
```

Podemos entonces utilizar el constructor pandas.DataFrame() para crear DataFrames:

```
class_features = pd.DataFrame(class_features)
class_labels = pd.DataFrame(class_labels)
```

Vamos a generar algunos datos de clasificación para la red que estamos construyendo.

I.1 Representación de Redes Neuronales

Instrucciones

1. Genere un conjunto de datos para la clasificación que incluya lo siguiente:
   - Exactamente 100 observaciones
   - Exactamente 4 caracteristicas
   - La semilla aleatoria 1
2. Convertir el array NumPy de características generadas en un DataFrame de pandas y asignarlo a class_features
3. Convertir el array NumPy de etiquetas generadas en una serie de pandas, y asignar a class_labels

Soluciones

```
from sklearn.datasets import make_classification
class_data = make_classification(n_samples=100, n_features=4, random_state=1)
class_features = pd.DataFrame(class_data[0])
class_labels = pd.Series(class_data[1])
```

I.2.8 Implementar Redes Neuronales para Clasificación

En las pantallas anteriores, hemos reproducido la regresión lineal como un modelo de red neuronal feedforward y hemos aprendido sobre las funciones de activación no lineales. Ahora tenemos una mejor idea de lo que define a una red neuronal. Hasta ahora, sabemos que las redes neuronales necesitan lo siguiente:

- Una estructura de red (¿Cómo se conectan los nodos? ¿En qué dirección fluyen los datos y los cálculos?)
- Una función feedforward (¿Cómo se combinan los pesos de los nodos y los valores de observación?)
- Una función de activación (¿Qué transformaciones se realizan en los datos?)
- Una función de ajuste del modelo (¿Cómo se ajusta el modelo?)

Ahora exploraremos cómo construir una red neuronal que replique un modelo de regresión logística. Empezaremos con una rápida recapitulación.

I.1 Representación de Redes Neuronales

Clasificación Binaria y Regresión Logística

En la clasificación binaria, queremos encontrar un modelo que pueda diferenciar entre dos valores categóricos (normalmente 0 y 1). Los valores 0 y 1 no tienen ningún peso numérico y actúan como marcadores de posición numéricos para las dos categorías. Podemos intentar aprender la probabilidad de que una determinada observación pertenezca a una u otra categoría.

En el lenguaje de la probabilidad condicional, nos interesa la probabilidad de que una determinada observación x pertenezca a cada categoría: $P(y=0|x)=0.3P(y=1|x)=0.7$

Como el universo de posibilidades sólo está formado por estas dos categorías, las probabilidades de ambas deben sumar 1. Esto nos permite simplificar lo que queremos que aprenda un modelo de clasificación binaria:

- $P(y=1|x)=?$
- Si $P(y=1|x)>0,5$, queremos que el modelo lo asigne a la categoría 1
- Si $P(y=1|x)<0,5$, queremos que el modelo lo asigne a la categoría 0

I.1 Representación de Redes Neuronales

Implementación de un Modelo de Regresión Logística

Un modelo de regresión logística consta de dos componentes principales:

- Calcular la combinación lineal ponderada de pesos y características (como en un modelo de regresión lineal): $Xa^T \quad \sigma\left(Xa^T\right)$
- Aplicar una función de transformación para aplastar el resultado de manera que varíe entre 0 y 1: $\hat{y} = \sigma(Xa^T)$

Combinando estos dos pasos se obtiene la siguiente definición de un modelo de regresión logística:

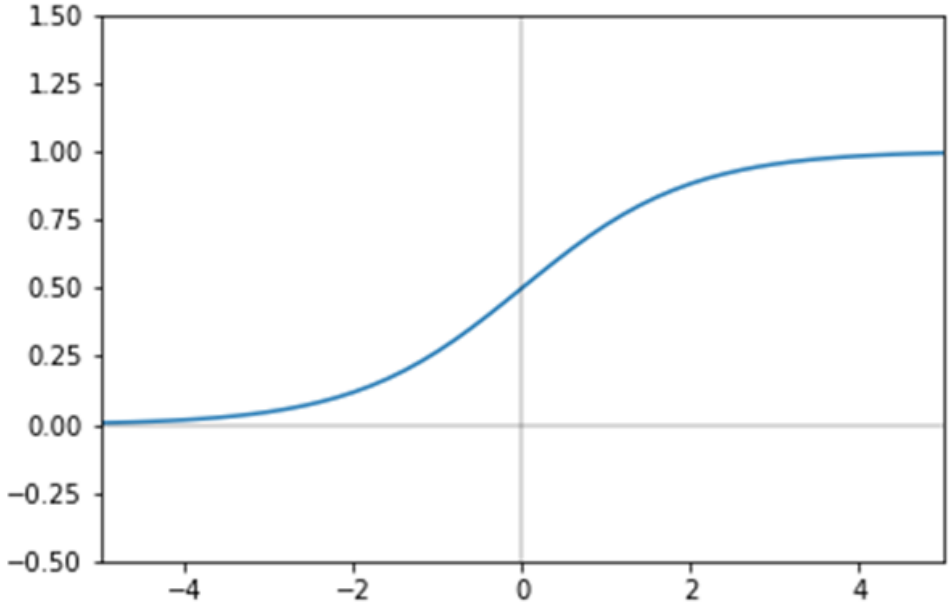
$$S(x) = \frac{1}{1+e^{-x}}$$

La literatura sobre redes neuronales suele referirse a esta función como la **función sigmoidea (sigmoid function).** Este es un gráfico de la función sigmoidea:

I.1 Representación de Redes Neuronales

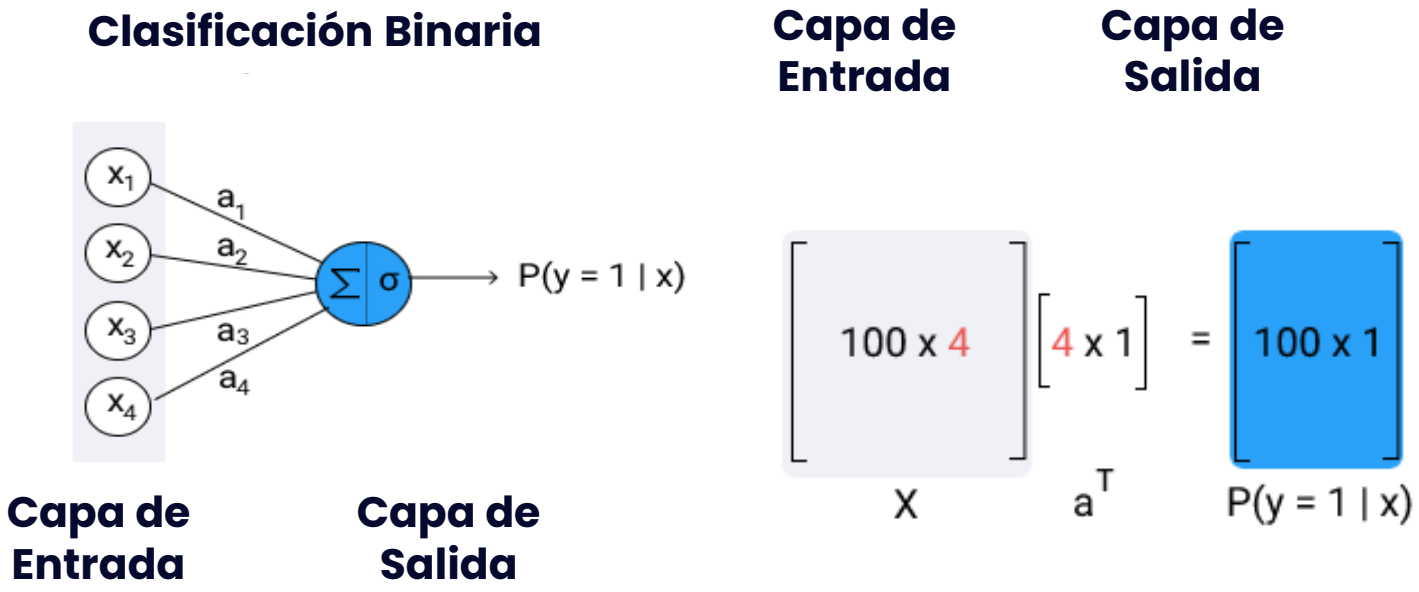
Verás que la función sigmoidea tiene asíntotas horizontales en 0 y 1, lo que significa que cualquier valor de entrada siempre dará un valor entre 0 y 1.

Para implementar una red que realice la clasificación, lo único que tenemos que cambiar de la red de regresión lineal que implementamos es la función de activación. En lugar de utilizar la función de identidad, tenemos que utilizar la función sigmoidea.



I.1 Representación de Redes Neuronales

Este es un diagrama de esta red:



I.1 Representación de Redes Neuronales

Instrucciones

1. Añade una columna llamada bias que contenga el valor 1 para cada fila del DataFrame class_features.
2. Defina tres funciones:

- log_train(class_features, class_labels): toma las series class_features DataFrame y class_labels y realiza el ajuste del modelo
- Utiliza la clase **SGDClassifier** de scikit-learn para manejar el ajuste del modelo.
- Esta función debería devolver una matriz NumPy 2D de pesos para el modelo de regresión logística.
- sigmoid(linear_combination): toma una matriz NumPy 2D y aplica la función sigmoid para cada valor: 11+e-x
- log_feedforward(class_features, log_train_weights): toma el DataFrame class_features y la matriz NumPy log_train_weights
- Realiza la multiplicación matricial entre class_features (100 filas por 5 columnas) y log_train_weights (1 fila por 5 columnas) transpuesta, y la asigna a linear_combination
- Utilizar la función sigmoide() para transformar linear_combinations y asignar el resultado a log_predictions
- Convierta cada valor de log_predictions en una etiqueta de clase:
- Si el valor es mayor o igual a 0,5, sobrescribir el valor a 1
- Si el valor es inferior a 0,5, sobrescribir el valor en 0
- Devuelve log_predictions

3. Quite los comentarios del código que hemos añadido y ejecute las funciones log_train() y log_feedforward(). Las predicciones finales estarán en log_predictions

I.1 Representación de Redes Neuronales

Soluciones

```
from sklearn.linear_model import SGDClassifier
from sklearn.datasets import make_classification

class_data = make_classification(n_samples=100, n_features=4,
random_state=1)
class_features = pd.DataFrame(class_data[0])
class_labels = pd.Series(class_data[1])
# Uncomment this code when you're ready to test your
functions.
# log_train_weights = log_train(class_features, class_labels)
# log_predictions = log_feedforward(class_features,
log_train_weights)
class_features["bias"] = 1

def log_train(class_features, class_labels):
    sg = SGDClassifier()
    sg.fit(class_features, class_labels)
    return sg.coef_

```

```
def sigmoid(linear_combination):
    return 1/(1+np.exp(-linear_combination))

def log_feedforward(class_features, log_train_weights):
    linear_combination = np.dot(class_features,
log_train_weights.T)
    log_predictions = sigmoid(linear_combination)
    log_predictions[log_predictions >= 0.5] = 1
    log_predictions[log_predictions < 0.5] = 0
    return log_predictions

log_train_weights = log_train(class_features, class_labels)
log_predictions = log_feedforward(class_features,
log_train_weights)
```

I.2 Funciones de Activación no Lineal

En la última misión, nos familiarizamos con los grafos computacionales y con la forma de representar los modelos de redes neuronales. También nos familiarizamos con la terminología de las redes neuronales como:

- Paso adelante
- Neuronas de entrada
- Neuronas de salida

En esta misión, profundizaremos en el papel que desempeñan las funciones de activación no lineales. Para motivar nuestra exploración, empecemos por reflexionar sobre el propósito de un modelo de aprendizaje automático.

El propósito de un modelo de aprendizaje automático es transformar las entradas de datos de entrenamiento al modelo (que son características) para aproximar los valores de salida de entrenamiento. Esto se logra mediante:

- Seleccionando un modelo específico para utilizar
- Encontrar los parámetros adecuados para este modelo que mejor funcionen
- Probar el modelo para saber si se generaliza a los nuevos datos

I.2 Funciones de Activación no Lineal

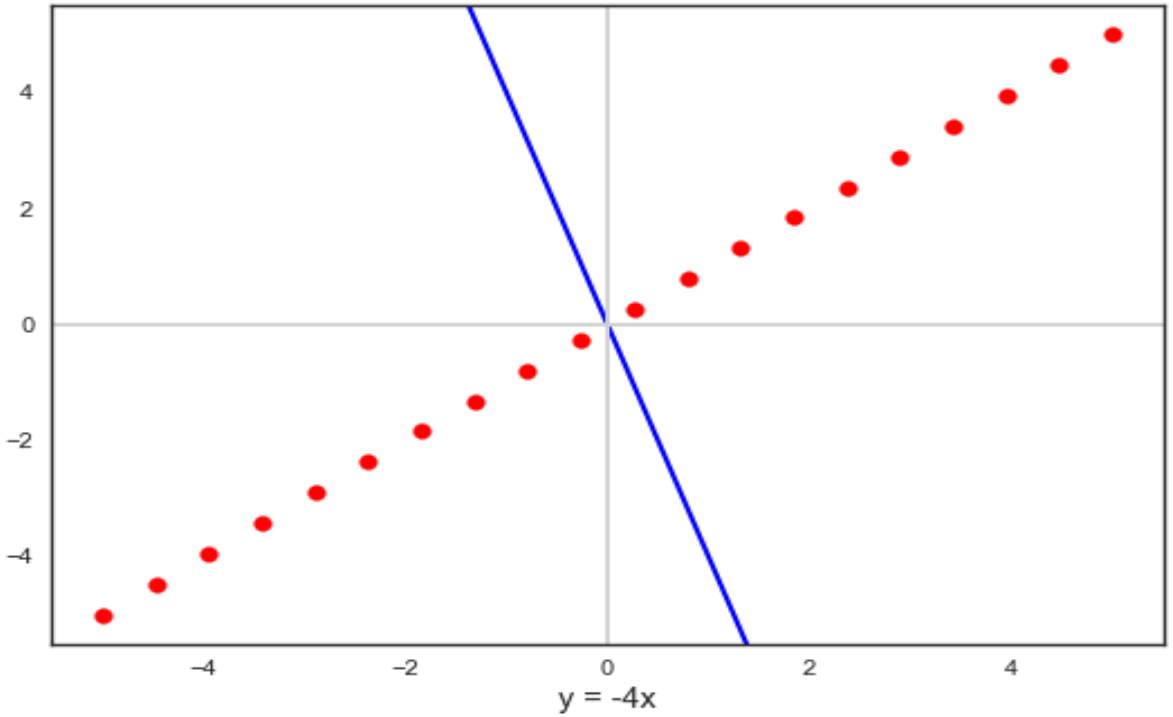
Regresión Lineal

Utilizamos la regresión lineal cuando pensamos que los valores de salida pueden aproximarse mejor mediante una combinación lineal de las características y los pesos aprendidos. Este modelo es un sistema lineal, porque cualquier cambio en el valor de salida es proporcional a los cambios en los valores de entrada.

$$\hat{y} = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$$

Cuando los valores objetivo **y** pueden ser aproximados por una combinación lineal de las características **x1** a xn, la regresión lineal es la opción ideal. Aquí hay un GIF que visualiza la expresabilidad potencial de un modelo de regresión lineal (imitando conceptualmente lo que hace el descenso de gradiente).

I.2 Funciones de Activación no Lineal



Veamos ahora una situación en la que los valores de salida no pueden aproximarse eficazmente utilizando una combinación lineal de los valores de entrada.

I.2 Funciones de Activación no Lineal

Regresión Logística

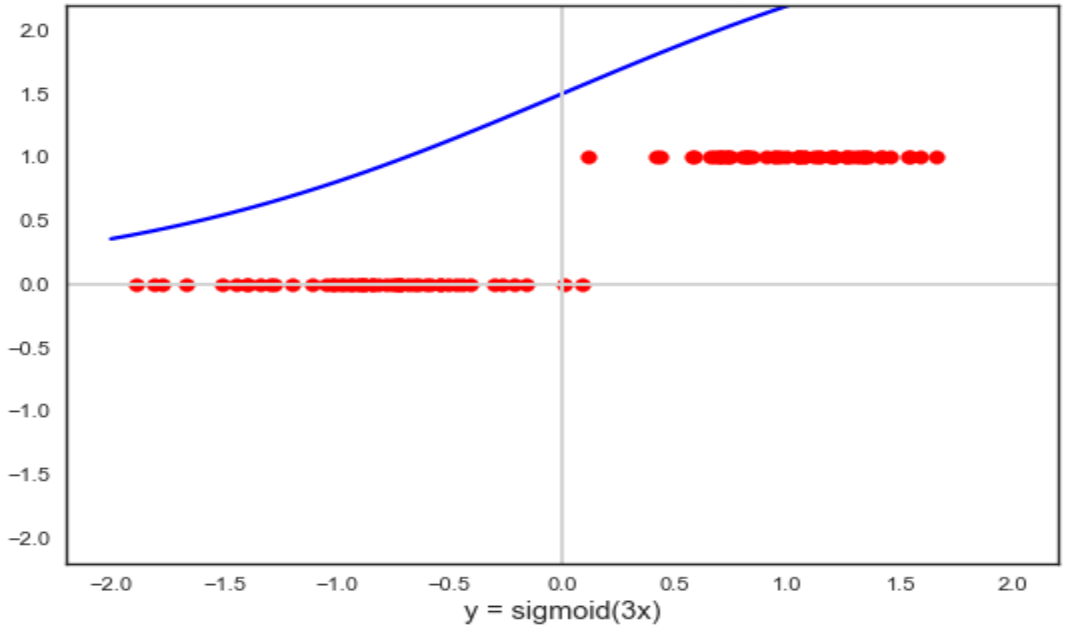
En un problema de clasificación binaria, los valores objetivo son 0 y 1 y la relación entre las características y los valores objetivo no es lineal. Esto significa que necesitamos una función que pueda realizar una transformación no lineal de las características de entrada.

La función sigmoidea es una buena elección, ya que todos sus valores de entrada se aplastan para que oscilen entre 0 y 1.

$$\hat{y} = \sigma(a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n)$$

La adición de la transformación sigmoidea ayuda al modelo a aproximarse a esta relación no lineal que subyace en las tareas comunes de clasificación binaria. El siguiente GIF muestra cómo cambia la forma del modelo de regresión logística a medida que aumentamos el peso único (imitando conceptualmente lo que hace el descenso de gradiente):

I.2 Funciones de Activación no Lineal



I.2 Funciones de Activación no Lineal

Red Neuronal

Los modelos de regresión logística aprenden un conjunto de pesos que inciden en la fase de combinación lineal y luego se alimentan a través de una única función no lineal (la función sigmoide). En esta misión, nos sumergiremos en las funciones de activación más utilizadas. Las tres funciones de activación más utilizadas en las redes neuronales son:

- La función sigmoidea
- La función ReLU
- La función tanh

Como ya hemos cubierto la función sigmoidea, nos centraremos en las dos últimas funciones.

I.2 Funciones de Activación no Lineal

1.2.1 Función de Activación ReLU

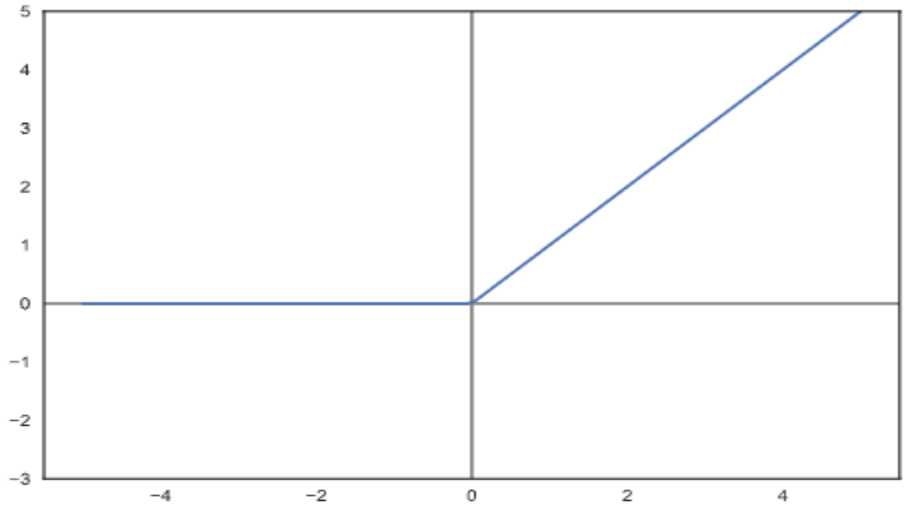
Empezaremos introduciendo la función de activación ReLU, que es una función de activación comúnmente utilizada en las redes neuronales para resolver problemas de regresión. ReLU significa unidad lineal rectificada y se define como sigue: ReLU(x)=max(0,x).

La llamada a la función max(0,x) devuelve el valor máximo entre 0 y x. Esto significa que:

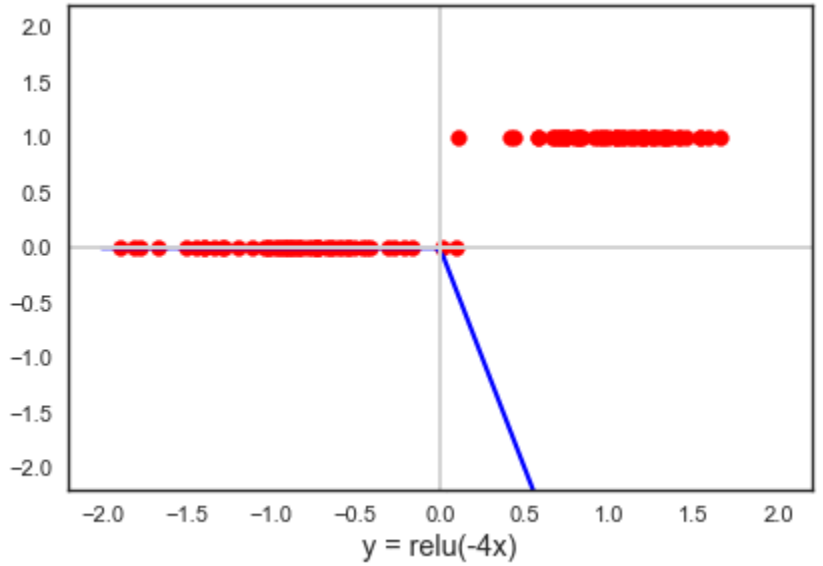
- Cuando x es menor que 0, se devuelve el valor 0
- Cuando x es mayor que 0, se devuelve el valor x

I.2 Funciones de Activación no Lineal

Aquí hay un gráfico de la función:



La función ReLU devuelve el componente positivo del valor de entrada. Visualicemos la expresividad de un modelo que realiza una combinación lineal de las características y pesos seguida de la transformación ReLU:



I.2 Funciones de Activación no Lineal

Hay varias formas de implementar la función ReLU en el código. Lo dejaremos como un ejercicio para que lo implementes.

Instrucciones

- Definir la función relu()
- Esta función debe poder trabajar con un solo valor o con una lista de valores
- Llama a la función relu(), pasa x y asigna el valor devuelto a relu_y
- Imprime tanto x como relu_y
- Generar un gráfico de líneas con x en el eje x y relu_y en el eje y

I.2 Funciones de Activación no Lineal

Soluciones

```
1  import matplotlib.pyplot as plt
2
3  import numpy as np
4  x = np.linspace(-2, 2, 20)
5  def relu(values):
6      return np.maximum(values, 0)
7
8  relu_y = relu(x)
9
10 print(x)
11 print(relu_y)
12
13 plt.plot(x, relu_y)
```


I.2 Funciones de Activación no Lineal

1.2.2 Activación Trigonométrica

La última función de activación comúnmente utilizada en las redes neuronales que discutiremos es la **función tanh** (también conocida como la función tangente hiperbólica). Empezaremos por repasar algo de trigonometría discutiendo la función tan (abreviatura de tangente) y luego trabajaremos hasta la **función tanh** (en la siguiente pantalla). Aunque no proporcionaremos aquí la profundidad necesaria para aprender trigonometría desde cero, recomendamos la **Serie de Trigonometría en Khan Academy (Trigonometry Series on Khan Academy)** si eres nuevo en la trigonometría.

¿Qué es la Trigonometría?

La trigonometría es la abreviatura de la geometría de los triángulos y proporciona fórmulas, marcos y modelos mentales para razonar sobre los triángulos. Los triángulos se utilizan ampliamente en las matemáticas teóricas y aplicadas, y se basan en el trabajo matemático realizado durante muchos siglos. Empecemos por definir claramente qué es un triángulo.

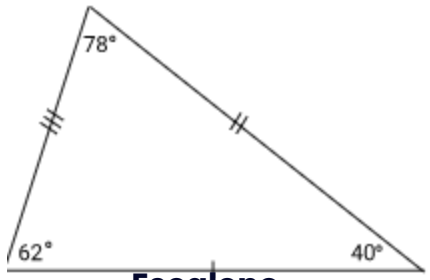
I.2 Funciones de Activación no Lineal

Un triángulo es un **polígono** que tiene las siguientes propiedades:

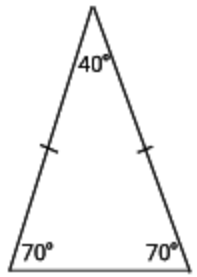
- 3 aristas
- 3 vértices
- Los ángulos entre aristas suman 180 grados

Los triángulos se pueden clasificar de dos formas principales: por los ángulos internos o por la longitud de las aristas. El siguiente diagrama describe los tres tipos de triángulos según las propiedades de la longitud de sus aristas:

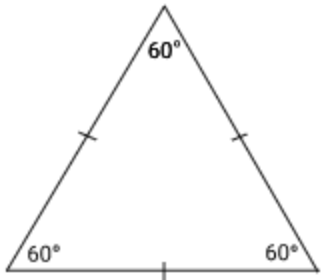
Clasificación de triángulos por longitud de las aristas



Escaleno
Las tres aristas tienen longitudes diferentes. Ángulos tienen diferentes medidas.



Isósceles
Dos aristas iguales

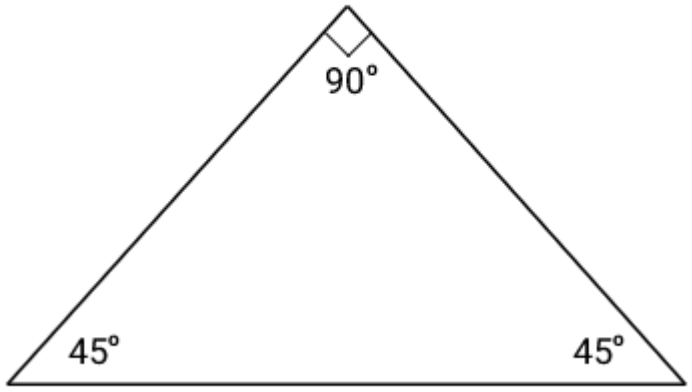


Equilátero
Todas las aristas iguales

I.2 Funciones de Activación no Lineal

Un triángulo importante que se clasifica por los ángulos internos es el triángulo rectángulo. En un triángulo rectángulo, uno de los ángulos es de 90 grados (también conocido como ángulo recto). La arista opuesta al ángulo recto se llama hipotenusa.

Triángulo de ángulo recto



Hipotenusa

Una función trigonométrica es una función que introduce un valor angular (normalmente representado como θ) y da como resultado algún valor. Estas funciones calculan relaciones entre las longitudes de los bordes. Aquí están las 3 primeras funciones trigonométricas:

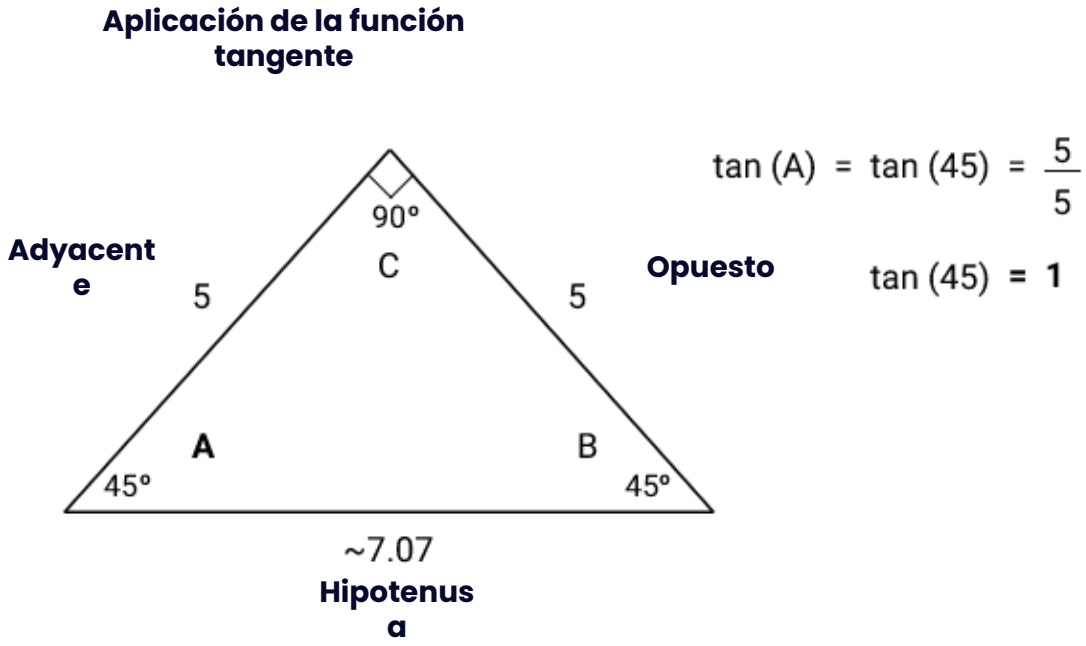
- $\sin(\theta) = \frac{opposite}{hypotenuse}$
- $\cos(\theta) = \frac{adjacent}{hypotenuse}$
- $\tan(\theta) = \frac{opposite}{adjacent}$

Definamos mejor estos términos:

- Hipotenusa describe la línea que no toca el ángulo recto
- Opuesta se refiere a la línea opuesta al ángulo
- Adyacente se refiere a la línea que toca el ángulo y que no es la hipotenusa

I.2 Funciones de Activación no Lineal

Este es un ejemplo de la función tangente.



Instrucciones

- Utiliza la función **numpy.tan()** para calcular la tangente de los valores de x y asigna el valor devuelto a tan_y
- Imprime tanto x como tan_y
- Genera un gráfico de líneas con x en el eje x y tan_y en el eje y

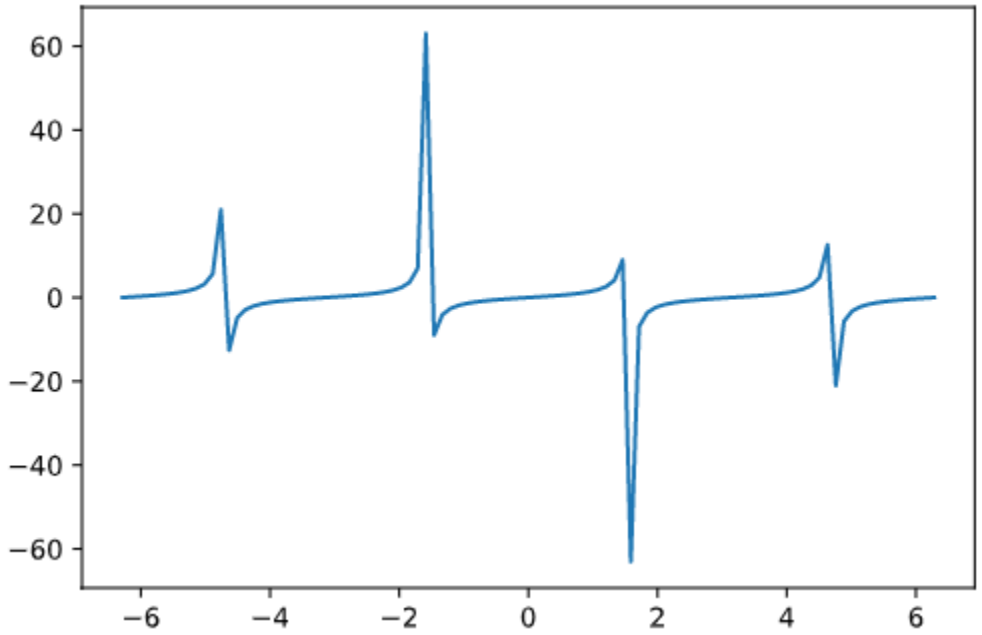
Soluciones

```
x = np.linspace(-2*np.pi, 2*np.pi, 100)
tan_y = np.tan(x)
plt.plot(x, tan_y)
```

I.2 Funciones de Activación no Lineal

1.2.3 Reflexión sobre la Función Tangente

La función tangente de la última pantalla generó el siguiente gráfico:



I.2 Funciones de Activación no Lineal

Los picos agudos periódicos que se ven en el gráfico se conocen como **asíntotas verticales (vertical asymptotes)**. En esos puntos, el valor no está definido, pero el límite se aproxima al infinito negativo o positivo (dependiendo de la dirección desde la que te aproximes al valor de x).

La clave de la gráfica es que la función tangente es **una función periódica** que se repite. Una función periódica es aquella que devuelve el mismo valor a intervalos regulares. Veamos una tabla con algunos valores de la función tangente:

x	tan(x)
-pi	0
0	0
pi	0

La función tangente se repite cada π, lo que se conoce como período. La función tangente no es conocida por ser utilizada como función de activación en redes neuronales (o cualquier modelo de aprendizaje automático en realidad) porque la naturaleza periódica no es un patrón que se encuentre en conjuntos de datos reales.

I.2 Funciones de Activación no Lineal

Aunque se han realizado algunos **experimentos con funciones periódicas (experiments with periodic functions)** como función de activación para redes neuronales, la conclusión general ha sido que las funciones periódicas como la tangente no ofrecen ninguna ventaja única para el modelado.

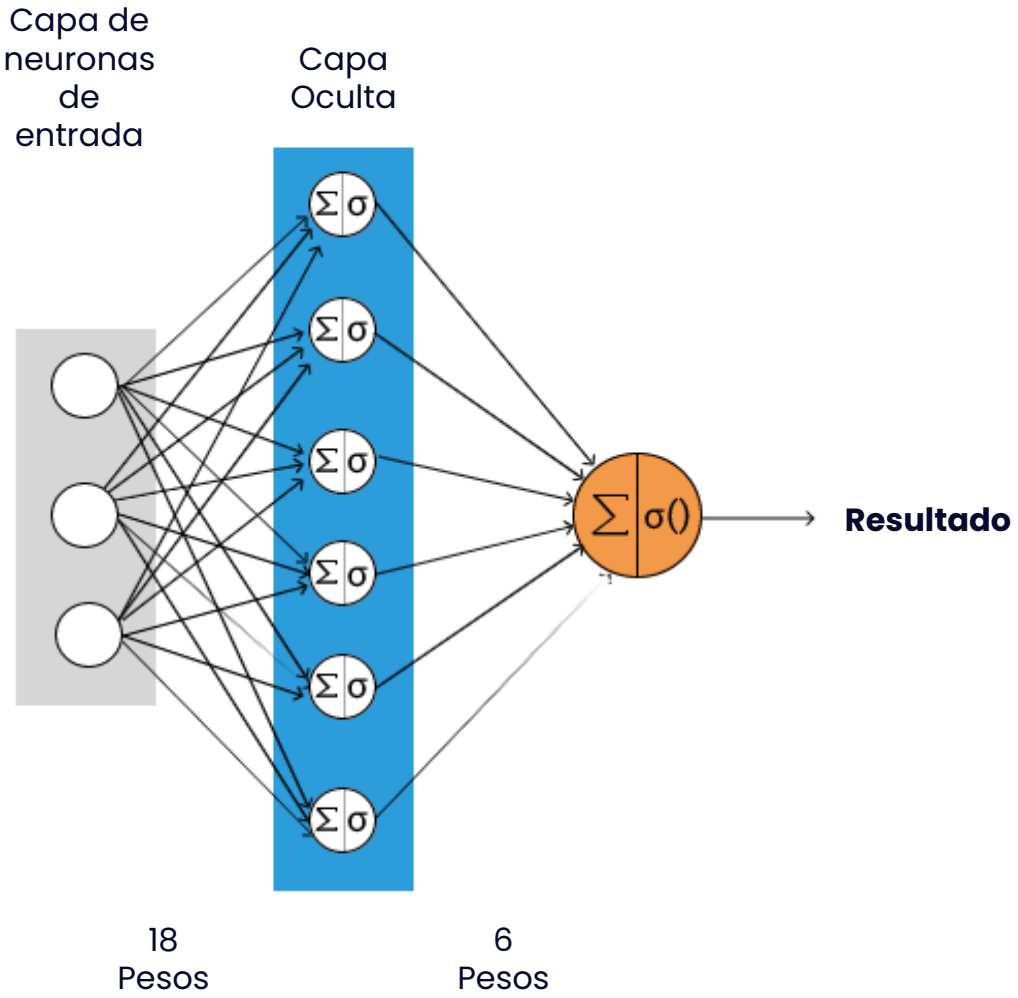
En general, las funciones de activación que se utilizan en las redes neuronales **son funciones crecientes (increasing functions)**. Una función creciente f es una función en la que f(x) siempre permanece igual o aumenta a medida que x aumenta.

Todas las funciones de activación que hemos visto (y veremos) en esta misión cumplen este criterio.

I.3 Capas Ocultas

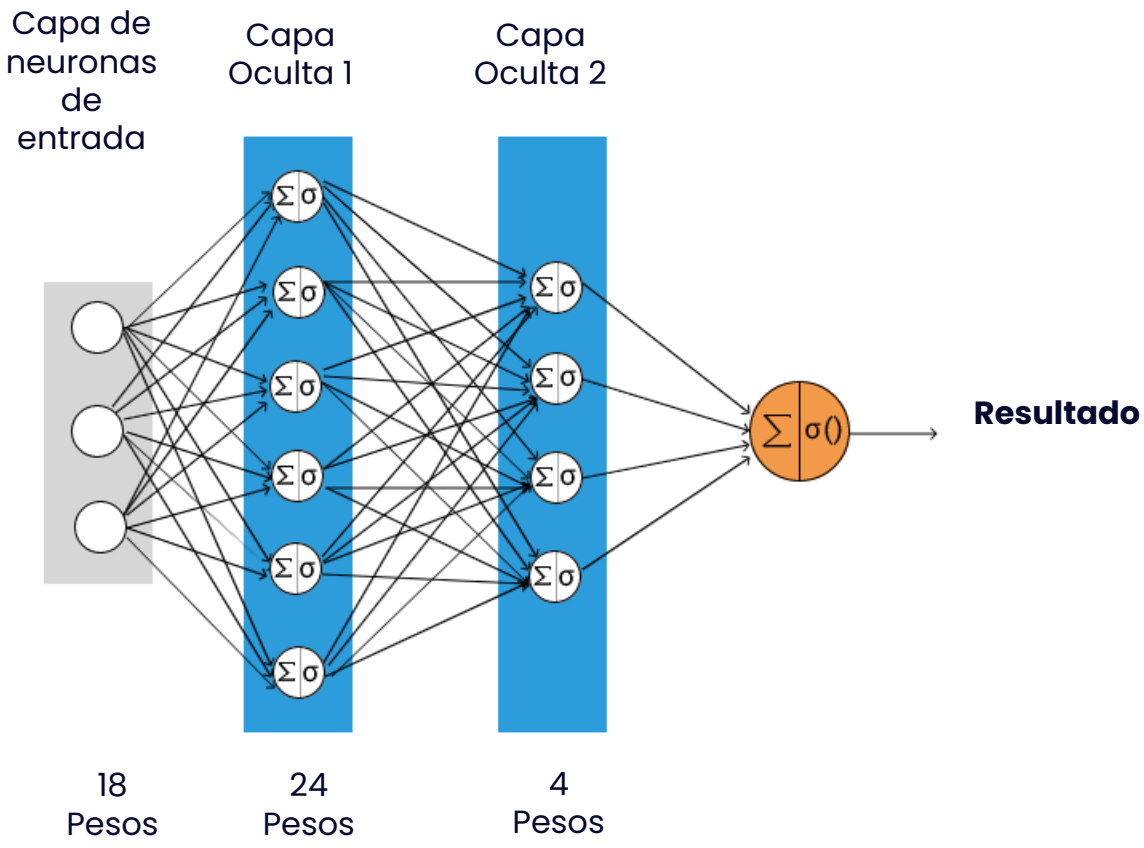
En las dos últimas misiones, hemos trabajado con **redes neuronales de una sola capa**. Estas redes tenían una sola capa de neuronas. Para hacer una predicción, una sola capa de neuronas en estas redes alimentaba directamente sus resultados a la(s) neurona(s) de salida.

En esta misión, exploraremos cómo las **redes multicapa** (también conocidas como **redes neuronales profundas**) son capaces de captar mejor la no linealidad de los datos. En una red neuronal profunda, la primera capa de neuronas de entrada alimenta una segunda capa intermedia de neuronas. Este es un diagrama que representa esta arquitectura:



I.3 Capas Ocultas

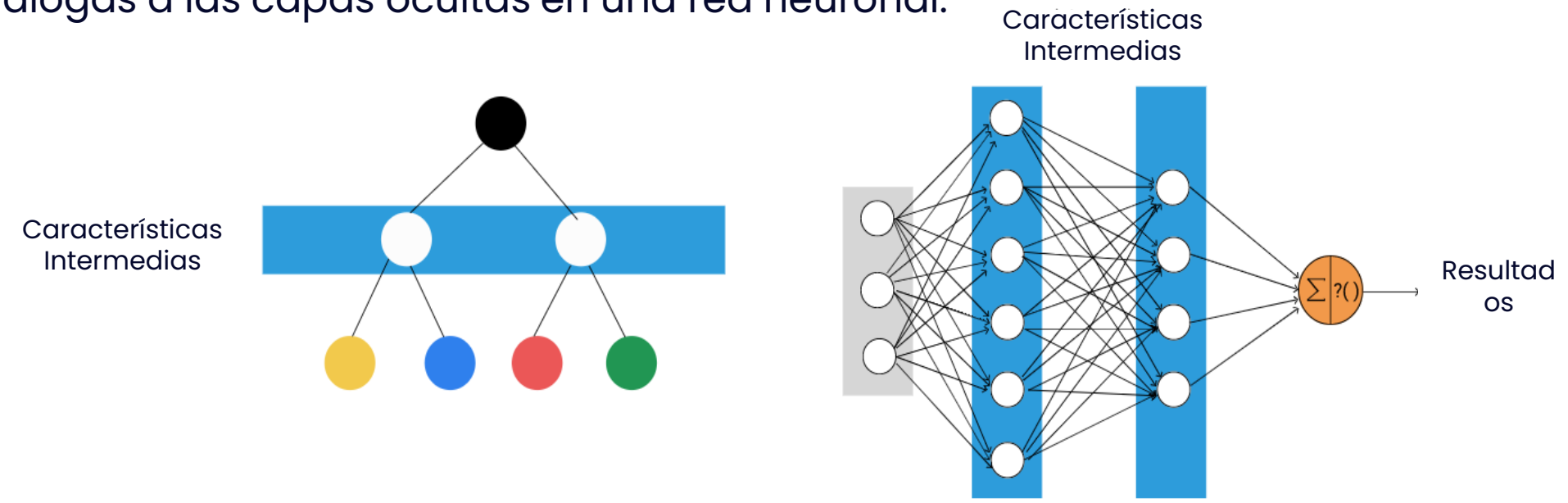
Incluimos las dos funciones que se utilizan para calcular cada neurona oculta y la neurona de salida para ayudar a aclarar cualquier confusión. Notarás que el número de neuronas de la segunda capa es mayor que el de la capa de entrada. La elección del número de neuronas en esta capa es un poco un arte y todavía no una ciencia en la literatura de redes neuronales. De hecho, podemos añadir más capas intermedias, y esto suele conducir a una mayor precisión del modelo (debido a una mayor capacidad de aprendizaje de la no linealidad).



I.3 Capas Ocultas

Las capas intermedias se conocen como capas ocultas, porque no están representadas directamente en los datos de entrada ni en las predicciones de salida. En cambio, podemos pensar en cada capa oculta como características intermedias que se aprenden durante el proceso de entrenamiento. Comparación con los modelos de árbol de decisión.

En realidad, esto es muy similar a cómo se estructuran los árboles de decisión. Las ramas y divisiones representan algunas características intermedias que son útiles para hacer predicciones y son análogas a las capas ocultas en una red neuronal:



I.3 Capas Ocultas

Cada una de estas capas ocultas tiene su propio conjunto de pesos y sesgos, que se descubren durante el proceso de entrenamiento. En los modelos de árbol de decisión, las características intermedias del modelo representan algo más concreto que podemos entender (rangos de características).

Los modelos de árboles de decisión se denominan **modelos de caja blanca (white box models)** porque pueden observarse y entenderse, pero no se pueden modificar fácilmente. Después de entrenar un modelo de árbol de decisión, podemos visualizar el árbol, interpretarlo y tener nuevas ideas para ajustar el modelo. Las redes neuronales, en cambio, están mucho más cerca de ser una caja negra. En un modelo de **caja negra (black box)**, podemos entender las entradas y las salidas, pero las características intermedias son realmente difíciles de interpretar y comprender. Más difícil aún, y quizás más importante, es entender cómo ajustar una red neuronal basándose en estas características intermedias.

En esta misión, aprenderemos cómo añadir más capas a una red y añadir más neuronas en las capas ocultas puede mejorar la capacidad del modelo para aprender relaciones más complejas.

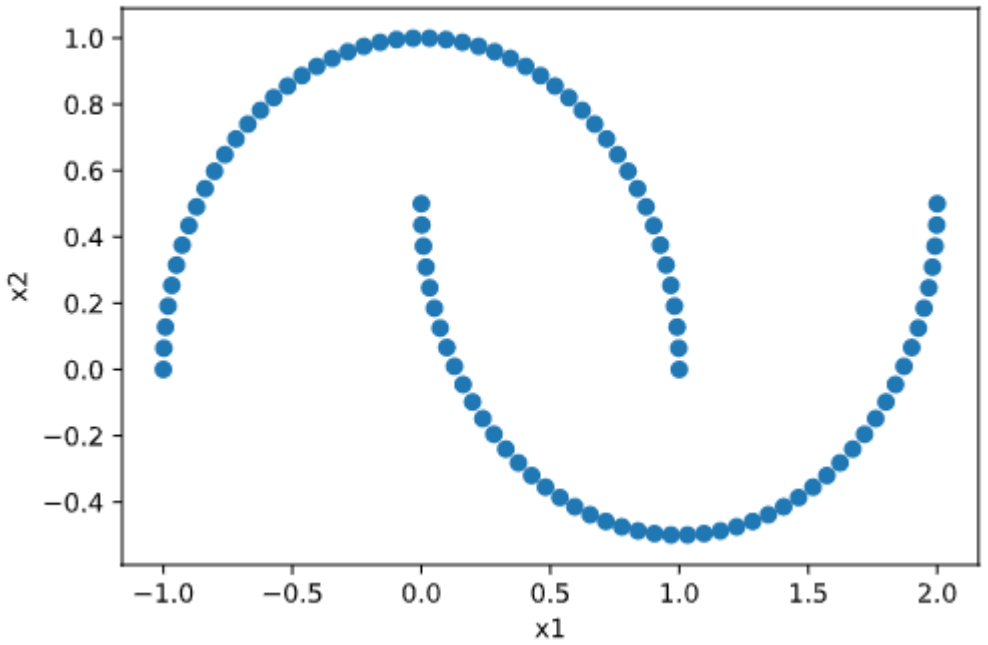
I.3 Capas Ocultas

I.3.1 Generación de datos con función de no linealidad

Para generar datos con no linealidad en las características (tanto entre las características como entre las características y la columna objetivo), podemos utilizar la función **make_moons()** de scikit-learn:

```
from sklearn.datasets import make_moons
data = make_moons()
```

Por defecto, make_moons() generará 100 filas de datos con 2 características. Aquí hay un gráfico que visualiza una característica contra la otra:



I.3 Capas Ocultas

Para hacer las cosas más interesantes, vamos a añadir un poco de **Ruido Gaussiano (Gaussian noise)**a los datos. El ruido gaussiano es un tipo de ruido estadístico que sigue la distribución gaussiana, y es una forma común de intentar recrear el ruido que a menudo se encuentra en los datos del mundo real.

Podemos utilizar el parámetro noise para especificar la desviación estándar del ruido gaussiano que queremos añadir a los datos. También establezcamos random_state a 3 para que los datos generados puedan ser recreados:

```
data = make_moons(random_state = 3, noise=0.04)
```

Al igual que en una misión anterior, podemos separar el objeto NumPy resultante en 2 dataframes de pandas:

```
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
print(features)

>>   0   1
0   1.919020    0.195719
1   0.510473    -0.380575
2   0.933035    0.332699
...

print(labels)
>>   0
0   1
1   1
2   0
...
```

I.3 Capas Ocultas

Instrucciones

- Utilice la función make_moons() para generar datos con no linealidad:
  - Generar 100 valores
  - Establezca la semilla aleatoria en 3
  - Establezca el parámetro de ruido en 0,04
- Convertir el array de NumPy de características generadas en un dataframe de pandas y asignarlo a las features (características)
- Convertir la matriz NumPy de etiquetas generadas en una serie de pandas y asignarla a las etiquetas
- Generar un gráfico de dispersión en 3D de los datos:
  - Crear un objeto matplotlib figure y establecer figsize en (8,8)
  - Cree y adjunte un objeto de eje único a esta figura utilizando la proyección 3d: ax = fig.add_subplot(111, projection='3d')
  - Generar un **gráfico de dispersión 3d (3d scatter plot)** con la primera columna de features (características) en el eje x, la segunda columna de features (características) en el eje y y los labels (etiqueta) en el eje z
  - Establezca las etiquetas 'x1', 'x2' e 'y', respectivamente

I.3 Capas Ocultas

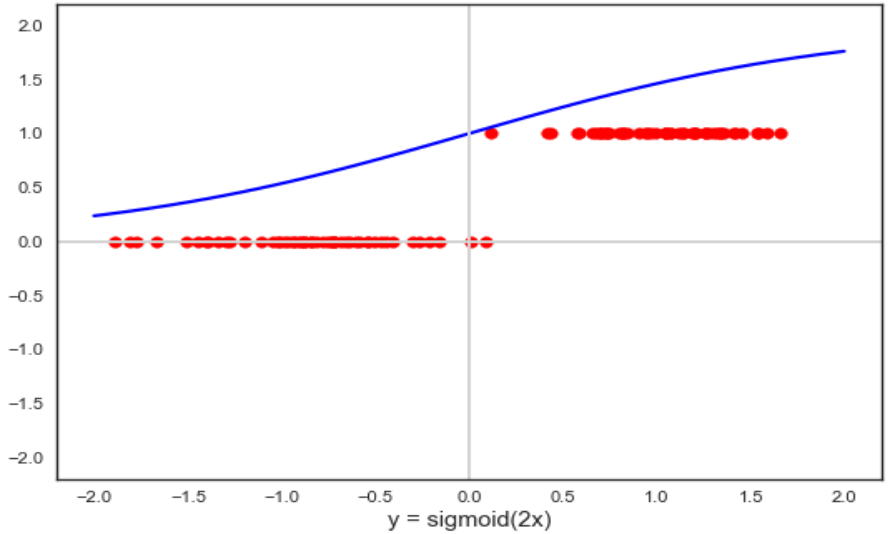
Soluciones

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import make_moons
import pandas as pd
import matplotlib.pyplot as plt
data = make_moons(100, random_state=3, noise=0.04)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(features[0], features[1], labels)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
```


I.3 Capas Ocultas

I.3.2 Capa oculta con una sola neurona

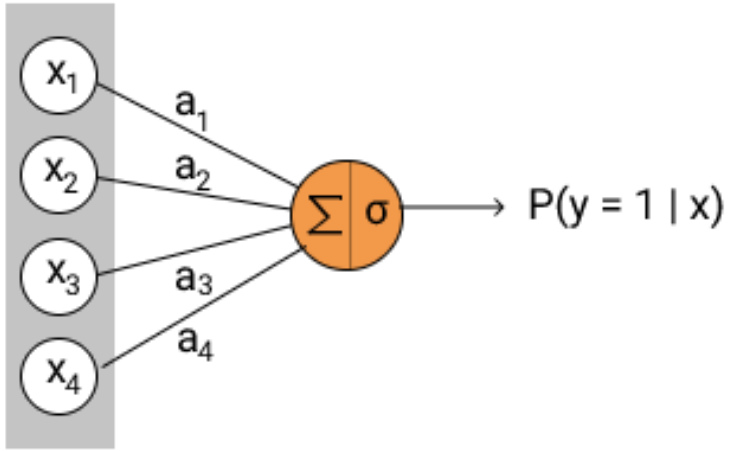
En la última misión, aprendimos cómo la adición de una función de activación no lineal ampliaba la gama de patrones que un modelo podía intentar aprender. El siguiente GIF muestra cómo la adición de la función sigmoide permite a un modelo de regresión logística capturar la no linealidad de manera más eficaz:

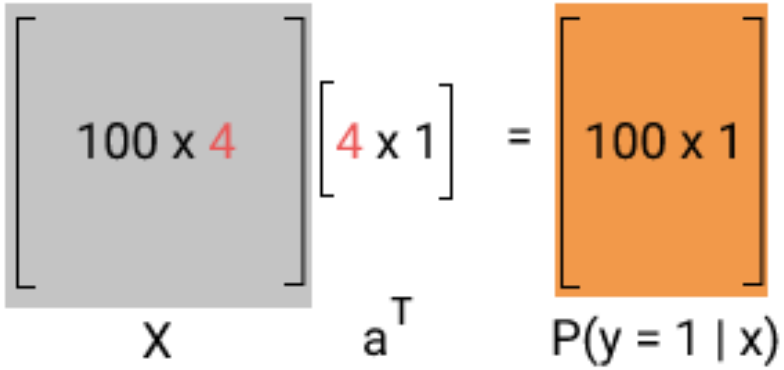


I.3 Capas Ocultas

Podemos pensar en un modelo de regresión logística como una red neuronal con una función de activación pero sin capas ocultas. Para hacer predicciones, se realiza una combinación lineal de las características y los pesos, seguida de una única transformación sigmoidea.

Clasificación Binaria



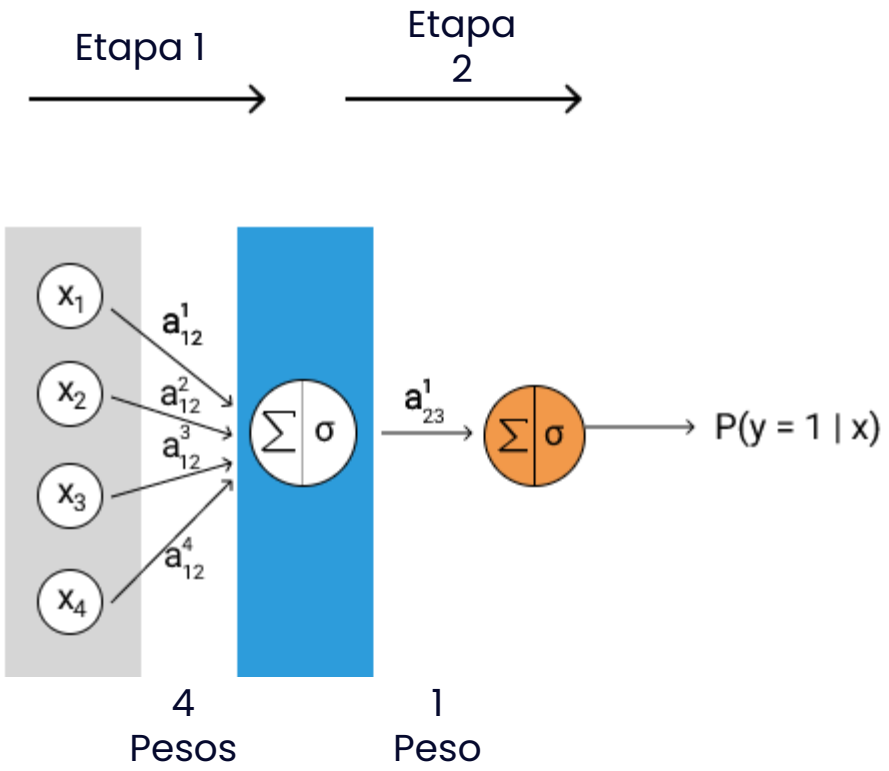


Capa de Entrada

Capa de Salida

I.3 Capas Ocultas

Para mejorar la capacidad expresiva, podemos añadir una capa oculta de neuronas entre la capa de entrada y la de salida. Este es un ejemplo en el que hemos añadido una sola capa oculta con una sola neurona entre la capa de entrada y la de salida:



Etapa 1

$$\sigma\left(\begin{bmatrix} 100 \times 4 \end{bmatrix}\begin{bmatrix} 4 \times 1 \end{bmatrix}\right) = \begin{bmatrix} 100 \times 1 \end{bmatrix}$$

X a_1^T L_1

↓

Etapa 2

$$\sigma\left(\begin{bmatrix} 100 \times 1 \end{bmatrix}\begin{bmatrix} 1 \times 1 \end{bmatrix}\right) = \begin{bmatrix} 100 \times 1 \end{bmatrix}$$

L_1 a_2^T L_2

I.3 Capas Ocultas

Esta red contiene dos conjuntos de pesos que se aprenden durante la fase de entrenamiento:

- 4 pesos entre la capa de entrada y la capa oculta
- 1 peso entre la capa oculta y la capa de salida

En la siguiente pantalla, aprenderemos a entrenar una red neuronal con una capa oculta utilizando scikit-learn. Compararemos este modelo con un modelo de regresión logística.

I.3 Capas Ocultas

I.3.3 Entrenamiento de una red neuronal con Scikit-learn

Scikit-learn contiene dos clases para trabajar con redes neuronales:

- **MLPClassifier**
- **MLPRegressor**

Vamos a centrarnos en la clase MLPClassifier. Como todas las clases de modelos en scikit-learn, MLPClassifier sigue el patrón estándar model.fit() y model.predict():

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
mlp.fit(X_train, y_train)
predictions = mlp.predict(X_test)
```


I.3 Capas Ocultas

Podemos especificar el número de neuronas ocultas que queremos utilizar en cada capa utilizando el parámetro hidden_layer_sizes. Este parámetro acepta una tupla en la que el valor del índice corresponde al número de neuronas de esa capa oculta. El parámetro se establece a la tupla (100) por defecto, que corresponde a cien neuronas en una sola capa oculta. El siguiente código especifica una capa oculta de seis neuronas:

```
mlp = MLPClassifier(hidden_layer_sizes=(6,))
```

Podemos especificar la función de activación (activation) que queremos que se utilice en todas las capas utilizando el parámetro de activación. Este parámetro sólo acepta los siguientes valores de cadena:

- 'identity': la función de identidad
- 'logistic': la función sigmoidea
- 'tanh': la función tangente hiperbólica (tanh)
- 'relu': la función ReLU

I.3 Capas Ocultas

Este es un modelo instanciado con la función de activación sigmoidea:

```
mlp = MLPClassifier(hidden_layer_sizes=(6,),
activation='logistic')
```

Aunque scikit-learn es fácil de usar cuando se aprenden nuevos conceptos, tiene algunas limitaciones cuando se trata de trabajar con redes neuronales en producción.

- En el momento de escribir esto, scikit-learn sólo admite el uso de la misma función de activación para todas las capas
- Scikit-learn también tiene dificultades para escalar a conjuntos de datos más grandes
  - Bibliotecas como **Theano** y **TensorFlow** permiten descargar algunos cálculos a la GPU para superar los cuellos de botella

I.3 Capas Ocultas

Instrucciones

- Entrene dos modelos diferentes utilizando scikit-learn en el conjunto de entrenamiento:
  - Un modelo de regresión logística estándar
  - Una red neuronal con:
    - Una sola capa oculta
    - Una sola neurona en la capa oculta
    - La función de activación sigmoidea
- Realice y asigne predicciones (a efectos de comprobación de respuestas, debe respetarse el orden):
  - Realice predicciones sobre el conjunto de pruebas utilizando el modelo de red neuronal y asígnelas a nn_predicciones
  - Realice predicciones sobre el conjunto de pruebas utilizando el modelo de regresión logística y asígnelas a log_predictions
- Calcule la puntuación de precisión (**accuracy score**) de log_predictions y asígnela a log_accuracy
- Calcule la puntuación de precisión (**accuracy score**) de nn_predicciones y asígnela a nn_precisión
- Imprima tanto log_accuracy como nn_accuracy

I.3 Capas Ocultas

Soluciones

```
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
np.random.seed(8)

data = make_moons(100, random_state=3, noise=0.04)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
features["bias"] = 1

shuffled_index = np.random.permutation(features.index
shuffled_data = features.loc[shuffled_index]
shuffled_labels = labels.loc[shuffled_index]
mid_length = int(len(shuffled_data)/2)
```

```
train_features = shuffled_data.iloc[0:mid_length]
test_features =
shuffled_data.iloc[mid_length:len(shuffled_data)]
train_labels = shuffled_labels.iloc[0:mid_length]
test_labels = shuffled_labels.iloc[mid_length:
len(labels)]
mlp = MLPClassifier(hidden_layer_sizes=(1,),
activation='logistic')
mlp.fit(train_features, train_labels)
nn_predictions = mlp.predict(test_features)

lr = LogisticRegression()
lr.fit(train_features, train_labels)
log_predictions = lr.predict(test_features)
nn_accuracy = accuracy_score(test_labels,
nn_predictions)
log_accuracy = accuracy_score(test_labels,
log_predictions)

print("Logistic Regression Model Accuracy: ",
log_accuracy)
print("Single Neuron Single Layer NN Model Accuracy: ",
nn_accuracy)
```


I.3 Capas Ocultas

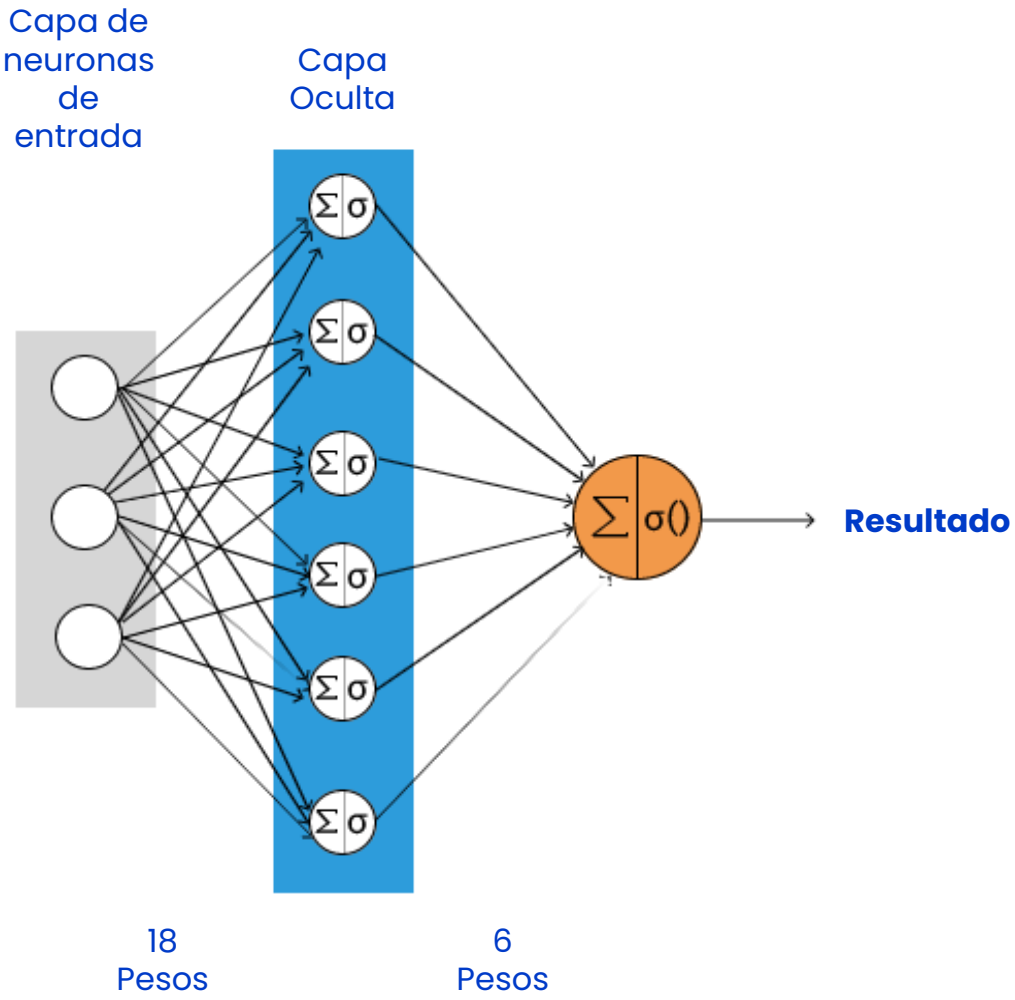
I.3.4 Capa oculta con múltiples neuronas

En la última pantalla, hemos entrenado un modelo de regresión logística y un modelo de red neuronal con una capa oculta que contiene una sola neurona. Aunque no recomendamos utilizar las puntuaciones de precisión para evaluar los modelos de clasificación en un entorno de producción, pueden ser útiles cuando estamos aprendiendo y experimentando porque son fáciles de entender.

El modelo de regresión logística se comportó mucho mejor (precisión del 88%) en comparación con el modelo de red neuronal con una capa oculta y una neurona (48%). Desgraciadamente, esta arquitectura de red no da al modelo mucha capacidad para captar la no linealidad de los datos, por lo que la regresión logística funcionó mucho mejor.

Veamos una red con una sola capa oculta de múltiples neuronas:

I.3 Capas Ocultas



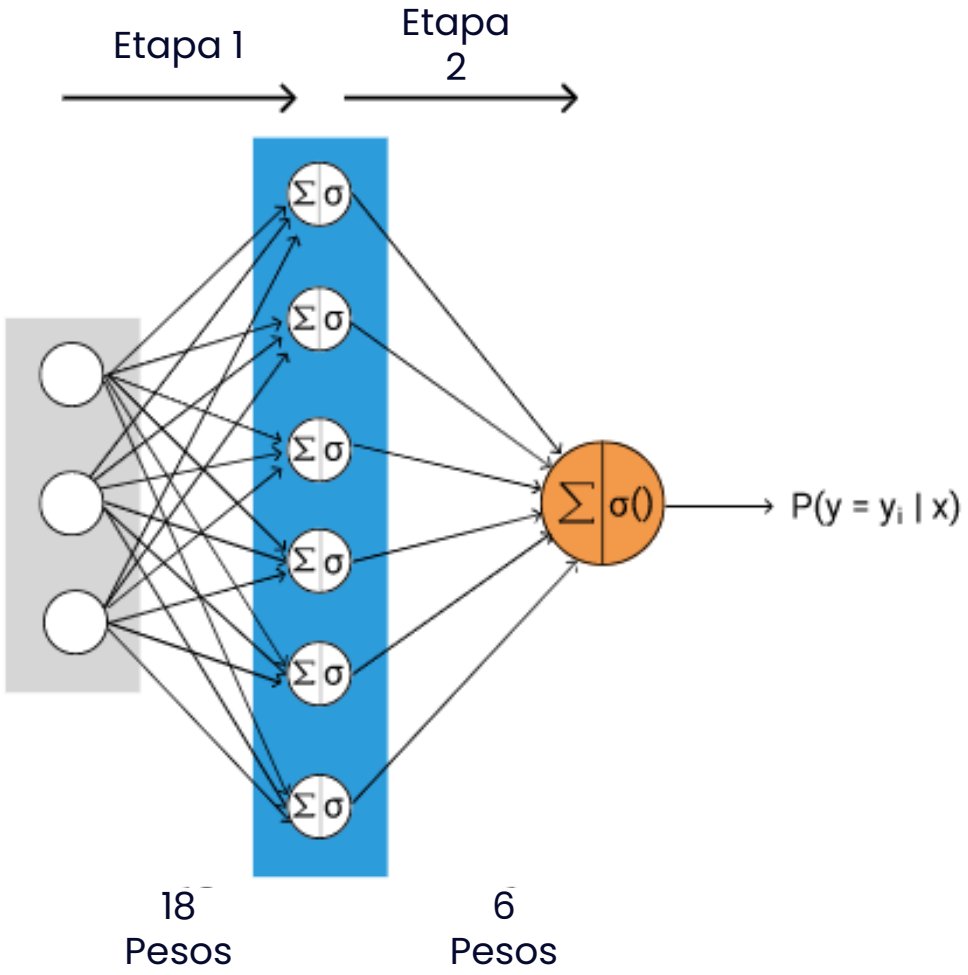
I.3 Capas Ocultas

Esta red tiene 3 neuronas de entrada, 6 neuronas en la única capa oculta y 1 neurona de salida. Verás que hay una flecha entre cada neurona de entrada y cada neurona oculta (3 x 6 = 18 conexiones), lo que representa un peso que hay que aprender durante el proceso de entrenamiento. Verás que también hay un peso que hay que aprender entre cada neurona oculta y la neurona de salida final (6 x 1 = 6 conexiones).

Dado que cada neurona tiene una conexión entre sí y todas las neuronas de la capa siguiente, esto se conoce como una **red totalmente conectada**. Por último, dado que el cálculo fluye de la izquierda (capa de entrada) a la derecha (capa oculta y luego a la capa de salida), podemos llamar a esta red una red totalmente conectada, de tipo feedforward.

Hay dos matrices de pesos (a1 y a2) que deben aprenderse durante el proceso de entrenamiento, una para cada etapa del cálculo. Veamos la representación en álgebra lineal de esta red.

I.3 Capas Ocultas



Etapa 1

$$\sigma\left(\begin{bmatrix} 100 \times 3 \\ X \end{bmatrix}\begin{bmatrix} 3 \times 6 \\ a_1^T \end{bmatrix}\right) = \begin{bmatrix} 100 \times 6 \\ L_1 \end{bmatrix}$$

↓

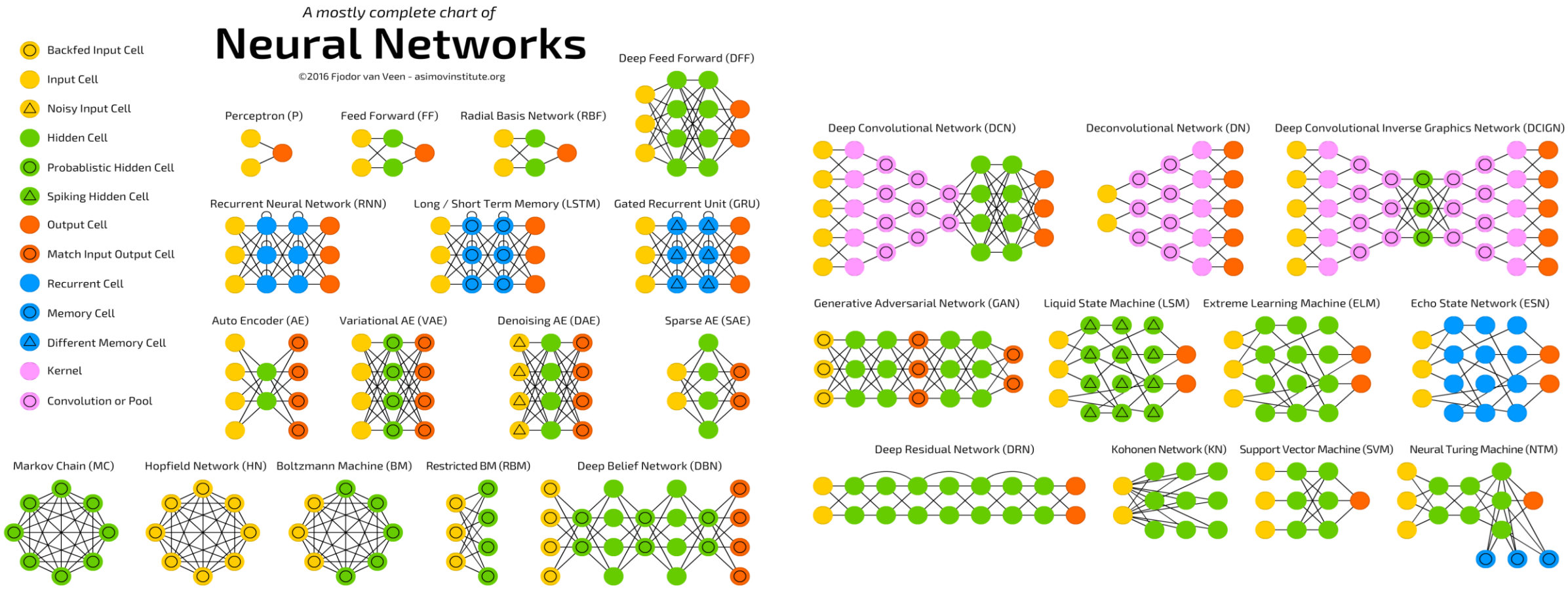
Etapa 2

$$\sigma\left(\begin{bmatrix} 100 \times 6 \\ L_1 \end{bmatrix}\begin{bmatrix} 6 \times 1 \\ a_2^T \end{bmatrix}\right) = \begin{bmatrix} 100 \times 1 \\ P(y = y_i \mid x) \end{bmatrix}$$

I.3 Capas Ocultas

Aunque hemos hablado de diferentes arquitecturas en este curso, una red neuronal profunda se reduce a una serie de multiplicaciones de matrices emparejadas con transformaciones no lineales. Estas son las ideas clave que subyacen en todas las arquitecturas de redes neuronales. Echa un vistazo a este diagrama conceptual del **Instituto Asimov (the Asimov Institute)** que muestra una variedad de arquitecturas de redes neuronales:

I.3 Capas Ocultas



I.3 Capas Ocultas

Instrucciones

- Cree la siguiente lista de recuentos de neuronas y asígnelos a neurons: [1, 5, 10, 15, 20, 25]
- Crear una lista vacía llamada accuracies (precisiones)
- Para cada valor de neurons:
  - Entrena una red neuronal:
    - Con el número de neuronas en la capa oculta fijado en el valor actual
    - Utilizando la función de activación sigmoidea en el conjunto de entrenamiento
  - Realiza predicciones en el conjunto de prueba y calcula el valor de precisión
  - Añade el valor de precisión a las accuracies (precisiones)
- Imprima las accuracies (precisiones)

I.3 Capas Ocultas

Soluciones

```
np.random.seed(8)
shuffled_index = np.random.permutation(features.index)
shuffled_data = features.loc[shuffled_index]
shuffled_labels = labels.loc[shuffled_index]
mid_length = int(len(shuffled_data)/2)
train_features = shuffled_data.iloc[0:mid_length]
test_features =
shuffled_data.iloc[mid_length:len(shuffled_data)]
train_labels = shuffled_labels.iloc[0:mid_length]
test_labels = shuffled_labels.iloc[mid_length: len(labels)]
neurons = [1, 5, 10, 15, 20, 25]
accuracies = []

for n in neurons:
    mlp = MLPClassifier(hidden_layer_sizes=(n,),
activation='logistic')
    mlp.fit(train_features, train_labels)
    nn_predictions = mlp.predict(test_features)
    accuracy = accuracy_score(test_labels, nn_predictions)
    accuracies.append(accuracy)

print(accuracies)
```


I.3 Capas Ocultas

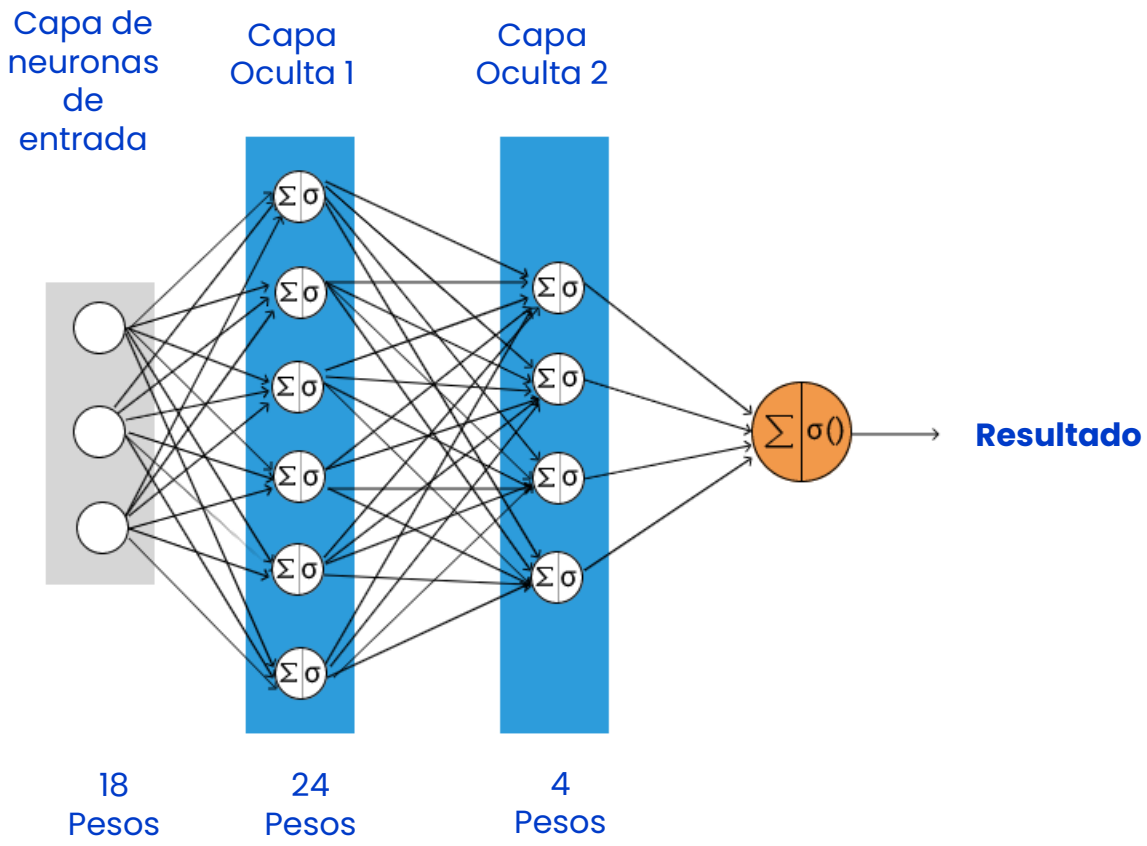
I.3.5 Capa Oculta Múltiple

Parece que la precisión de la predicción del conjunto de pruebas mejoró hasta el 0,86 cuando se utilizaron diez o quince neuronas en la capa oculta. A medida que aumentamos el número de neuronas en la capa oculta, la precisión mejoró enormemente entre los modelos:

```
print(accuracies)
> [0.48, 0.78, 0.86, 0.82, 0.84, 0.86]
```

Parece que la precisión de la predicción del conjunto de pruebas mejoró hasta el 0,86 cuando se utilizaron diez o quince neuronas en la capa oculta. A medida que aumentamos el número de neuronas en la capa oculta, la precisión mejoró enormemente entre los modelos:

I.3 Hidden Layers



I.3 Capas Ocultas

Para determinar el número de pesos entre las capas, multiplique el número de neuronas entre esas dos capas. Recuerda que estos pesos se representarán como matrices de pesos.

Para especificar el número de capas ocultas y el número de neuronas en cada capa oculta, cambiamos la tupla que pasamos al parámetro **hidden_layer_sizes**:

```
# Network with 2 hidden layers, 1 neuron in each hidden layer
mlp = MLPClassifier(hidden_layer_sizes=(1,1),
activation='logistic')

# Network with 3 hidden layers of varying neuron counts in
each hidden layer
mlp = MLPClassifier(hidden_layer_sizes=(2,6,10),
activation='logistic')
```

I.3 Capas Ocultas

El número de capas ocultas y el número de neuronas en cada capa oculta son hiperparámetros que actúan como mandos para el comportamiento del modelo. La optimización de los hiperparámetros de las redes neuronales está, por desgracia, fuera del alcance de este curso, ya que requiere una base matemática más sólida que planeamos proporcionar en futuros cursos.

Vamos a entrenar los siguientes modelos de redes neuronales:

- Modelo con dos capas ocultas, cada una con una neurona
- Modelo con dos capas ocultas, cada una con cinco neuronas
- Modelo con dos capas ocultas, cada una con diez neuronas
- Modelo con dos capas ocultas, cada una con quince neuronas
- Modelo con dos capas ocultas, cada una con veinte neuronas
- Modelo con dos capas ocultas, cada una con veinticinco neuronas

Cambiemos también la función de activación utilizada en las capas ocultas por la función ReLU.

I.3 Capas Ocultas

Las redes neuronales suelen tardar mucho en converger durante el proceso de entrenamiento y muchas bibliotecas tienen valores por defecto para el número de iteraciones de descenso de gradiente a ejecutar. Podemos aumentar el número de iteraciones del descenso de gradiente que se realiza durante el proceso de entrenamiento modificando el parámetro max_iter, que está fijado en 200 por defecto.

```
mlp = MLPClassifier(hidden_layer_sizes=(1,1),
activation='relu', max_iter=1000)
```

I.3 Capas Ocultas

Instrucciones

- Cree la siguiente lista de recuentos de neuronas y asígnelos a las neuronas - neurons: [1, 5, 10, 15, 20, 25]
- Cree una lista vacía llamada nn_accuracies
- Para cada valor de las neurons:
  - Entrene una red neuronal:
    - Con dos capas ocultas, cada una con el mismo número de neuronas (el valor actual en neuronas)
    - Usando la función de activación **relu**
    - Usando 1000 iteraciones de descenso de gradiente
    - En el conjunto de entrenamiento
  - Realiza predicciones en el conjunto de pruebas y calcula el valor de precisión
  - Añada el valor de precisión a nn_accuracies
- Imprimir nn_accuracies

I.3 Capas Ocultas

Soluciones

```
neurons = [1, 5, 10, 15, 20, 25]
nn_accuracies = []

for n in neurons:
    mlp = MLPClassifier(hidden_layer_sizes=(n,n),
activation='relu', max_iter=1000)

    mlp.fit(train_features, train_labels)
    nn_predictions = mlp.predict(test_features)

    accuracy = accuracy_score(test_labels, nn_predictions)
    nn_accuracies.append(accuracy)

print(nn_accuracies)
```


I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

En la última misión, aprendimos cómo la adición de capas ocultas de neuronas a una red neuronal puede mejorar su capacidad para capturar la no linealidad en los datos. Probamos diferentes modelos de redes neuronales en un conjunto de datos que generamos con no linealidad deliberada.

En este proyecto guiado, vamos a:

- Explorar por qué la clasificación de imágenes es una tarea difícil
- Observar las limitaciones de los modelos tradicionales de aprendizaje automático para la clasificación de imágenes
- Entrenar, probar y mejorar algunas redes neuronales profundas diferentes para la clasificación de imágenes

Como mencionamos en la primera misión de este curso, las redes neuronales profundas se han utilizado para alcanzar un rendimiento de vanguardia en tareas de clasificación de imágenes en la última década. Para algunas tareas de clasificación de imágenes, las redes neuronales profundas realmente se desempeñan tan bien o ligeramente mejor que el punto de referencia humano. Puede leer sobre la historia de las redes neuronales profundas **aquí.**

I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

Para terminar este curso, construiremos modelos que puedan clasificar los dígitos escritos a mano. Antes del año 2000, instituciones como la Oficina de Correos de los Estados Unidos utilizaban programas de reconocimiento de escritura a mano para leer direcciones, códigos postales, etc. En este **artículo** se detalla uno de sus enfoques, que consiste en preprocesar las imágenes manuscritas y luego alimentar un modelo de red neuronal:

¿Por qué la clasificación de imágenes es una tarea difícil?

En el campo del aprendizaje automático y el reconocimiento de patrones, la clasificación de imágenes (especialmente de texto manuscrito) se encuentra en el extremo más difícil del espectro. Esto se debe a varias razones.

En primer lugar, cada imagen de un conjunto de entrenamiento es altamente dimensional. Cada píxel de una imagen es una característica y una columna independiente. Esto significa que una imagen de 128 x 128 tiene 16384 características.

I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

En segundo lugar, las imágenes suelen reducirse a resoluciones inferiores y transformarse en escala de **grises** (sin color). Por desgracia, esto es una limitación de la potencia de cálculo. La resolución de una foto de 8 megapíxeles es de 3.264 por 2.448 píxeles, lo que supone un total de 7.990.272 rasgos (unos 8 millones). Las imágenes de esta resolución suelen reducirse a una escala de entre 128 y 512 píxeles en cualquier dirección para que el procesamiento sea mucho más rápido. Esto suele suponer una pérdida de detalles disponibles para el entrenamiento y la comparación de patrones.

En tercer lugar, las características de una imagen no tienen una relación lineal o no lineal obvia que pueda aprenderse con un modelo como la regresión lineal o logística. En escala de grises, cada píxel se representa simplemente como un valor de brillo que va de 0 a 256.

Este es un ejemplo de cómo se representa una imagen en las diferentes abstracciones que nos interesan:

I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

Imagen Uniforme en la Base de Datos

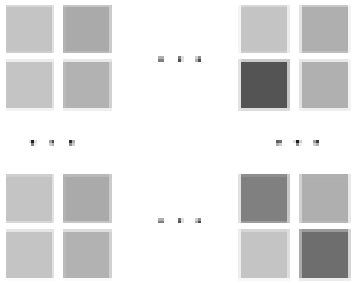


Imagen Renderizada

$$\begin{bmatrix} 50 & 90 & \dots & 70 & 90 \\ 50 & 82 & & 180 & 70 \\ \dots & & & \dots & \\ 50 & & \dots & 120 & 50 \\ 50 & & & 50 & 120 \end{bmatrix}$$

Valor de Pixeles

Miles de millones de columnas

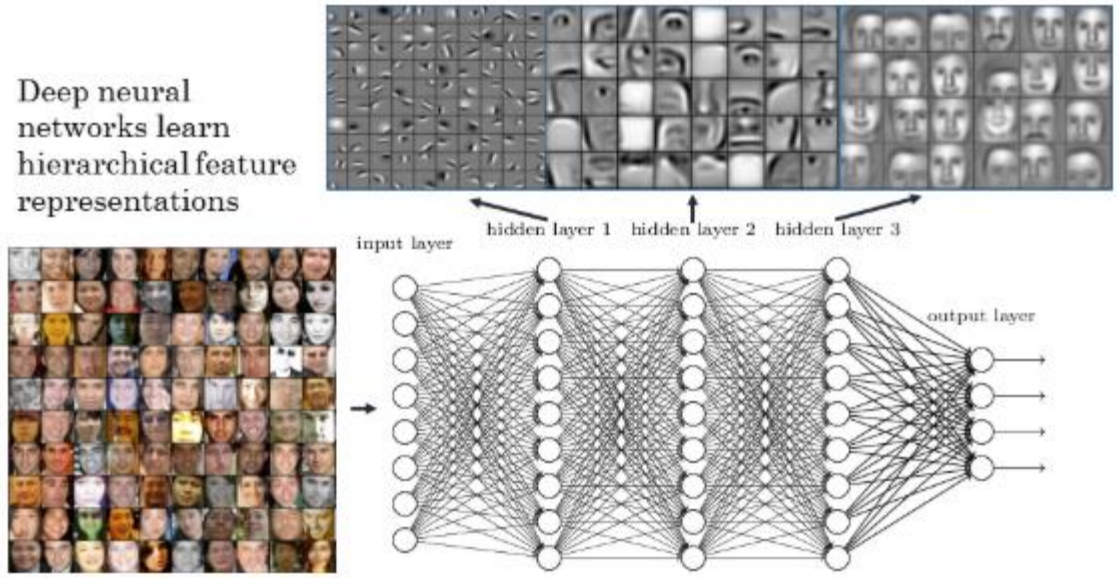
$$\begin{bmatrix} 50 & 90 & \dots & 70 & 90 & 50 & 82 & \dots & 50 & 120 \end{bmatrix}$$

Una sola observación en los datos

I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

¿Por qué el aprendizaje profundo es eficaz en la clasificación de imágenes?

El aprendizaje profundo es eficaz en la clasificación de imágenes debido a la capacidad de los modelos para aprender representaciones jerárquicas. A alto nivel, un modelo de aprendizaje profundo eficaz aprende representaciones intermedias en cada capa del modelo y las utiliza en el proceso de predicción. Aquí hay un diagrama que visualiza lo que representan los pesos en cada capa de una red neuronal convolucional, un tipo de red que se utiliza a menudo en la clasificación de imágenes y que, lamentablemente, está fuera del alcance de este curso, que fue entrenado para identificar rostros.



I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

Observarás que en la primera capa oculta la red aprendió a representar bordes y rasgos específicos de las caras. En la segunda capa oculta, los pesos parecían representar rasgos faciales de mayor nivel, como ojos y narices. Por último, los pesos de la última capa oculta representan rostros que podrían compararse. Cada capa sucesiva utiliza los pesos de las capas anteriores para intentar aprender representaciones más complejas.

En este proyecto guiado, exploraremos la eficacia de las redes neuronales profundas y feedforward para clasificar imágenes.

Scikit-learn contiene una serie de **conjuntos de datos (datasets)** precargados con la biblioteca, dentro del espacio de nombres **sklearn.datasets**. La **función load_digits() (load_digits() function)** devuelve una copia del **hand-written digits dataset** (conjunto de datos de dígitos escritos a mano) de la UCI.

I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

Como los marcos de datos son una representación tabular de los datos, cada imagen se representa como una fila de valores de píxeles. Para visualizar una imagen a partir del marco de datos, tenemos que remodelar la imagen a sus dimensiones originales (28 x 28 píxeles). Para visualizar la imagen, tenemos que volver a dar forma a estos valores de píxeles en 28 por 28 y trazarlos en una cuadrícula de coordenadas.

Para remodelar la imagen, necesitamos convertir un ejemplo de entrenamiento en un array de numpy (excluyendo la columna de etiquetas (label)) y pasar el resultado a la función **numpy.reshape()**

```
first_image = train.iloc[0]
first_image = first_image.drop('label', axis=1)
np_image = first_image.values
np_image = np_image.reshape(28,28)
```

Ahora que los datos tienen la forma adecuada, podemos visualizarlos utilizando la función **pyplot.imshow()**:

```
plt.imshow(np_image, cmap='gray_r')
```


I.4 Proyecto Guiado: Construcción de un Clasificador de Dígitos Escritos a Mano

Para mostrar múltiples imágenes en una figura de matplotlib, podemos utilizar la función equivalente **axes.imshow().** Usemos lo que hemos aprendido para mostrar imágenes de ambas clases.

Instrucciones

- Importar **load_digits()** del paquete sklearn.datasets
- Transformar el array 2D de NumPy en un dataframe de pandas
- Utilizar matplotlib para visualizar algunas de las imágenes del conjunto de datos
  - Generar una cuadrícula de dispersión, con 2 filas y 4 columnas
  - En la primera fila:
    - Mostrar las imágenes correspondientes a las filas 0, 100, 200 y 300
  - En la segunda fila:
    - Mostrar las imágenes correspondientes a las filas 1000, 1100, 1200, y 1300

II. Proyecto de Aprendizaje Automático

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

En este curso, recorreremos el ciclo de vida completo de la ciencia de datos, desde la limpieza de datos y la selección de características hasta el aprendizaje automático. Nos centraremos en la **modelización del crédito**, un conocido problema de la ciencia de datos que se centra en la modelización del **riesgo crediticio** de un prestatario. El crédito ha desempeñado un papel clave en la economía durante siglos y alguna forma de crédito ha existido desde el inicio del comercio. Trabajaremos con datos de préstamos financieros de **Lending Club**. Lending Club es un mercado de préstamos personales que pone en contacto a prestatarios que buscan un préstamo con inversores que quieren prestar dinero y obtener una rentabilidad. Puedes leer más sobre su mercado aquí.

Cada prestatario completa una solicitud exhaustiva, proporcionando su historial financiero anterior, el motivo del préstamo, y más. Lending Club evalúa la puntuación de crédito de cada prestatario utilizando datos históricos anteriores y su propio proceso de ciencia de datos para asignar un tipo de interés al prestatario. El tipo de interés es el porcentaje adicional al importe del préstamo solicitado que el prestatario tiene que devolver. Puedes leer más sobre el tipo de interés que asigna Lending Club **aquí.** Lending Club también intenta verificar toda la información que el prestatario proporciona, pero no puede verificar toda la información (normalmente por razones de regulación).

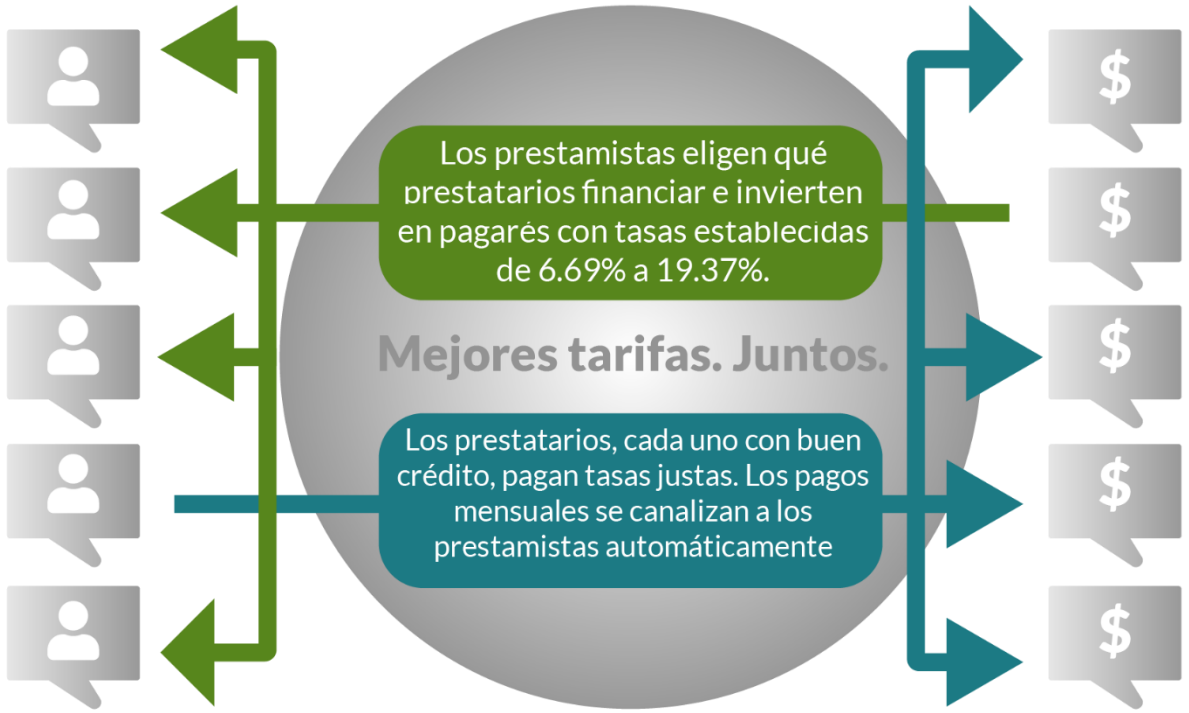
II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Un tipo de interés más alto significa que el prestatario es un riesgo y que es más improbable que devuelva el préstamo. Mientras que un tipo de interés más bajo significa que el prestatario tiene un buen historial crediticio y es más probable que devuelva el préstamo. Los tipos de interés van desde el 5,32% hasta el 30,99% y cada prestatario recibe **una calificación (grade)** según el tipo de interés que se le haya asignado. Si el prestatario acepta el tipo de interés, el préstamo aparece en el mercado de Lending Club.

A los inversores les interesa sobre todo recibir un rendimiento de sus inversiones. Los préstamos aprobados aparecen en el sitio web de Lending Club, donde los inversores cualificados pueden consultar los préstamos aprobados recientemente, la puntuación crediticia del prestatario, la finalidad del préstamo y otros datos de la solicitud. Una vez que están preparados para respaldar un préstamo, seleccionan la cantidad de dinero que quieren financiar. Una vez que la cantidad solicitada de un préstamo está totalmente financiada, el prestatario recibe el dinero que solicitó menos la **comisión de apertura (origination fee)** que cobra Lending Club.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

El prestatario realizará pagos mensuales a Lending Club en 36 o 60 meses. Lending Club redistribuye estos pagos a los inversores. Esto significa que los inversores no tienen que esperar a que se pague la totalidad del importe para ver un retorno en dinero. Si un préstamo se liquida completamente a tiempo, los inversores obtienen una rentabilidad que corresponde al tipo de interés que el prestatario tuvo que pagar además de la cantidad solicitada. **Muchos préstamos no se pagan completamente a tiempo y algunos prestatarios no pagan el préstamo (default)**.



Este es un diagrama de **Bible Money Matters** que resume el proceso:

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Mientras que Lending Club tiene que ser extremadamente inteligente y riguroso con su modelado crediticio, los inversores en Lending Club tienen que ser igualmente inteligentes a la hora de determinar qué préstamos tienen más probabilidades de ser pagados. Al principio, uno puede preguntarse por qué los inversores invierten dinero en cualquier cosa que no sea un préstamo de bajo interés. El incentivo que tienen los inversores para respaldar los préstamos con mayor interés es, bueno, el mayor interés. Si los inversores creen que el prestatario puede devolver el préstamo, incluso si tiene un historial financiero débil, entonces los inversores pueden ganar más dinero a través de la mayor cantidad adicional que el prestatario tiene que pagar.

La mayoría de los inversores utilizan una estrategia de cartera para invertir pequeñas cantidades en muchos préstamos, con mezclas saludables de préstamos de bajo, medio y alto interés. En este curso, nos centraremos en la mentalidad de un inversor conservador que sólo quiere invertir en los préstamos que tienen una buena oportunidad de ser pagados a tiempo. Para ello, primero tendremos que entender las características del conjunto de datos y luego experimentar con la construcción de modelos de aprendizaje automático que predigan de forma fiable si un préstamo se pagará o no.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

II.2.1 Descripción de Datos

Lending Club publica periódicamente los datos de todas las solicitudes de préstamo aprobadas y rechazadas en **su sitio web.** Puede seleccionar diferentes rangos de años para descargar los conjuntos de datos (en formato CSV) para los préstamos aprobados y rechazados.

También encontrará un **diccionario de datos** (en formato XLS) que contiene información sobre los diferentes nombres de las columnas al final de la página. Le recomendamos que descargue el diccionario de datos para poder consultarlo siempre que quiera saber más sobre lo que representa una columna en los conjuntos de datos. Aquí tienes un enlace al archivo del diccionario de datos alojado en **Google Drive**.

Antes de entrar en los conjuntos de datos, vamos a familiarizarnos con el diccionario de datos. La hoja **LoanStats** describe los conjuntos de datos de préstamos aprobados y **RejectStats** describe los conjuntos de datos de préstamos rechazados. Dado que las solicitudes rechazadas no aparecen en el mercado de Lending Club y no están disponibles para la inversión, nos centraremos en los préstamos aprobados.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Los conjuntos de datos de préstamos aprobados contienen información sobre los préstamos actuales, los préstamos completados y los préstamos impagados. Definamos ahora el planteamiento del problema para este proyecto de aprendizaje automático:

- ¿Podemos construir un modelo de aprendizaje automático que pueda predecir con precisión si un prestatario pagará su préstamo a tiempo o no?

Antes de empezar a realizar el aprendizaje automático, tenemos que definir qué características queremos utilizar y qué columna representa el objetivo que queremos predecir. Empecemos por leer y explorar el conjunto de datos.

II.2.2 Lectura en Pandas

En esta lección, nos centraremos en los datos de préstamos aprobados de 2007 a 2011, ya que un buen número de los préstamos ya han finalizado. En los conjuntos de datos de los años posteriores, muchos de los préstamos están vigentes y todavía se están pagando.

Si completamos lo siguiente, podremos reducir el tamaño del conjunto de datos para facilitar su uso

- Eliminar la columna desc:
  - Que contiene una larga explicación de texto para cada préstamo
- Eliminar la columna url:
  - Que contiene un enlace a cada préstamo en Lending Club al que sólo se puede acceder con una cuenta de inversor
- Eliminar todas las columnas que contienen más del 50% de valores perdidos:
  - Lo que nos permite movernos más rápido ya que podemos pasar menos tiempo tratando de llenar estos valores

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

En primer lugar, vamos a leer el conjunto de datos en un Dataframe para que podamos empezar a explorar los datos y las características restantes.

Instrucciones

- Leer loans_2007.csv en un DataFrame llamado loans_2007 y utilizar la función de impresión para mostrar la primera fila del Dataframe
- Utilice la función print para:
  - Mostrar la primera fila de loans_2007
  - El número de columnas de loans_2007

II.2.3 Primer Grupo de Columnas

El Dataframe contiene muchas columnas y puede ser engorroso tratar de explorarlas todas a la vez. Separemos las columnas en 3 grupos de 18 columnas y utilicemos el **data dictionary (diccionario de datos)** para familiarizarnos con lo que representa cada columna. A medida que entienda cada característica, busque cualquier característica que:

- Revelar información del futuro (después de que el préstamo ya haya sido financiado)
- No afectan a la capacidad del prestatario para devolver el préstamo (por ejemplo, un valor de identificación generado aleatoriamente por Lending Club)
- Necesitan ser limpiados y están mal formateados
- Requieren más datos o mucho procesamiento para convertirse en una función útil
- Contienen información redundante

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Debemos prestar especial atención a la fuga de datos, ya que puede hacer que nuestro modelo se sobreajuste. Esto se debe a que el modelo utiliza datos sobre la columna objetivo que no estarían disponibles cuando utilizamos el modelo en futuros préstamos. Le animamos a que se tome su tiempo para entender cada columna, porque una mala comprensión podría hacerle cometer errores en el proceso de análisis de datos y de modelización. A medida que avanza por el diccionario, tenga en cuenta que debemos seleccionar una de las columnas como la columna objetivo que queremos utilizar para la fase de aprendizaje automático.

En esta pantalla y en las siguientes, vamos a centrarnos sólo en las columnas que tenemos que eliminar de la consideración. Luego, podemos volver atrás y diseccionar más las columnas que decidimos mantener.

Para facilitar este proceso, creamos una tabla que contiene el nombre, el tipo de datos, el valor de la primera fila y la descripción del diccionario de datos para las primeras 18 filas.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Nombre	dtype	Primer valor	Descripción
id	Object	1077501	Una identificación única asignada a LC para la lista de préstamos.
member_id	Float64	1.2966e+06	Un LC único asignado Id para el miembro prestatario.
loan_amnt	float64	5000	El monto indicado del préstamo solicitado por el prestatario
funded_amnt	float64	5000	El monto total comprometido con ese préstamo en ese momento
funded_amnt_inv	float64	49750	El importe total comprometido por los inversores para ese préstamo en ese momento
term	object	36 months	El número de pagos del préstamo. Los valores están en meses y pueden ser 36 o 60
int_rate	object	10.65%	Tasa de interés del préstamo
installment	float64	162.87	El pago mensual adeudado por el prestatario si el préstamo se origina.
grade	object	B	LC assigned loan grade
sub_grade	object	B2	LC assigned loan subgrade
emp_title	object	NaN	El título de trabajo proporcionado por el Prestatario al solicitar el préstamo
emp_length	object	10+ years	Duración del empleo en años. Los valores posibles están entre 0 y 10 donde 0 significa menos de un año y 10 significa diez o más años
home_ownership	object	RENT	El estado de propiedad de la vivienda proporcionado por el prestatario durante el registro. Nuestros valores son: ALQUILER, PROPIEDAD, HIPOTECA, OTROS
annual_inc	float64	24000	Los ingresos anuales auto informaron proporcionados por el prestatario durante el registro
verification_status	object	Verified	Indica si los ingresos fueron verificados por LC, no verificados, o si la fuente de ingresos fue verificada
issue_d	object	Dec-2011	El mes en que se financió el préstamo
loan_status	object	Charged Off	Estado actual del préstamo
pymnt_plan	object	n	Indica si se ha establecido un plan de pago para el préstamo
purpose	object	car	Una categoría proporcionada por el prestatario para la solicitud de préstamo

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Tras analizar cada columna, podemos concluir que es necesario eliminar las siguientes características:

- id: campo generado aleatoriamente por Lending Club sólo con fines de identificación única
- member_id: también es un campo generado aleatoriamente por Lending Club con fines de identificación única
- funded_amnt: filtra datos del futuro (después de que el préstamo haya empezado a financiarse)
- funded_amnt_inv: también filtra datos del futuro (después de que el préstamo haya empezado a financiarse)
- grade: contiene información redundante como la columna del tipo de interés (int_rate)
- sub_grade: también contiene información redundante como la columna del tipo de interés (int_rate)
- emp_title: requiere otros datos y mucho procesamiento para ser potencialmente útil
- issue_d: filtra datos del futuro (después de que el préstamo ya esté completamente financiado)

Recordemos que Lending Club asigna un grado y un subgrado en función del tipo de interés del prestatario. Mientras que los valores de grade y sub_grade son categóricos, la columna int_rate contiene valores continuos, que son más adecuados para el aprendizaje automático.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Ahora vamos a eliminar estas columnas del Dataframe antes de pasar al siguiente grupo de columnas.

Utilice el método Dataframe **drop** para eliminar las siguientes columnas del Dataframe loans_2007:

- Id
- member_id
- funded_amnt
- funded_amnt_inv
- grade
- sub_grade
- emp_title
- issue_d

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

II.2.4 Segundo grupo de columnas

Veamos ahora las siguientes 18 columnas:

Name	dtype	First Value	Description
title	object	Computer	El título del préstamo proporcionado por el prestatario
zip_code	object	860xx	Los primeros 3 números del código postal proporcionados por el prestatario en la solicitud de préstamo
addr_state	object	AZ	El estado proporcionado por el prestatario en la solicitud de préstamo
dti	float64	27.65	Una relación calculada utilizando los pagos mensuales totales de la deuda del prestatario sobre las obligaciones totales de la deuda, excluyendo la hipoteca y el préstamo LC solicitado, dividido por los ingresos mensuales autoinformados del prestatario
delinq_2yrs	float64	0	El número de incidencias vencidas de morosidad de más de 30 días en el archivo de crédito del prestatario durante los últimos 2 años
earliest_cr_line	object	janv-85	El mes en que se abrió la primera línea de crédito reportada del prestatario
inq_last_6mths	float64	1	El número de consultas en los últimos 6 meses (excluyendo consultas de automóviles e hipotecas)
open_acc	float64	3	El número de líneas de crédito abiertas en el archivo de crédito del prestatario
pub_rec	float64	0	Número de registros públicos despectivos
revol_bal	float64	13648	Saldo rotatorio total del crédito
revol_util	object	83.7%	Tasa de utilización de la línea rotatoria, o la cantidad de crédito que el prestatario está utilizando en relación con todo el crédito renovable disponible
total_acc	float64	9	El número total de líneas de crédito actualmente en el archivo de crédito del prestatario
initial_list_status	object	f	El estado inicial de listado del préstamo. Los valores posibles son – W, F
out_prncp	float64	0	Capital pendiente restante por el monto total financiado
out_prncp_inv	float64	0	Capital pendiente restante por una parte del monto total financiado por los inversores
total_pymnt	float64	5863.16	Pagos recibidos hasta la fecha por el importe total financiado
total_pymnt_inv	float64	5833.84	Pagos recibidos hasta la fecha por una parte del importe total financiado por los inversores
total_rec_prncp	float64	5000	Principal recibido hasta la fecha

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Dentro de este grupo de columnas, tenemos que eliminar las siguientes columnas:

- zip_code: redundante con la columna addr_state, ya que sólo son visibles los 3 primeros dígitos del código postal de 5 cifras (que sólo puede utilizarse para identificar el estado en el que vive el prestatario)
- out_prncp: filtra datos del futuro, (después de que el préstamo ya haya empezado a pagarse)
- out_prncp_inv: también filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- total_pymnt: también filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- total_pymnt_inv: también filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- total_rec_prncp: también filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)

Las columnas out_prncp y out_prncp_inv describen el importe principal pendiente de un préstamo, que es el importe restante que el prestatario aún debe. Estas dos columnas, así como la columna total_pymnt, describen las propiedades del préstamo una vez que se ha financiado completamente y se ha empezado a pagar. Esta información no está disponible para un inversor antes de que el préstamo esté totalmente financiado y no queremos incluirla en nuestro modelo.

Sigamos adelante y eliminemos estas columnas del **Dataframe**.

.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Utilice el método Dataframe **drop** para eliminar las siguientes columnas del Dataframe loans_2007 :

- zip_code
- out_prncp
- out_prncp_inv
- total_pymnt
- total_pymnt_inv
- total_rec_prncp

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

II.2.5 Tercer grupo de columnas

Pasemos ahora al último grupo de características:

Name	dtype	First Value	Description
total_rec_int	float64	863.16	Intereses recibidos hasta la fecha
total_rec_late_fee	float64	0	Cargos por mora recibidos hasta la fecha
recoveries	float64	0	Cargo posterior a la recuperación bruta
collection_recovery_fee	float64	0	Cargo posterior a la tarifa de cobro
last_pymnt_d	object	janv-15	El mes pasado se recibió el pago
last_pymnt_amnt	float64	171.62	Último monto total de pago recibido
last_credit_pull_d	object	juin-16	El mes más reciente LC retiró crédito por este préstamo
collections_12_mths_ex_med	float64	0	Número de colecciones en 12 meses, excluidas las colecciones médicas
policy_code	float64	1	disponibles públicamente policy_code=1 nuevos productos no disponibles públicamente policy_code=2
application_type	object	INDIVIDUAL	Indica si el préstamo es una solicitud individual o una solicitud conjunta con dos coprestatarios
acc_now_delinq	float64	0	El número de cuentas en las que el prestatario está ahora en mora.
chargeoff_within_12_mths	float64	0	Número de cargos en un plazo de 12 meses
delinq_amnt	float64	0	El monto vencido adeudado por las cuentas en las que el prestatario está ahora en mora.
pub_rec_bankruptcies	float64	0	Número de quiebras de registro público
tax_liens	float64	0	Número de gravámenes fiscales

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

En el último grupo de columnas, tenemos que eliminar las siguientes columnas:

- total_rec_int: filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- total_rec_late_fee: filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- recoveries: filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- collection_recovery_fee: filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- last_pymnt_d: filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)
- last_pymnt_amnt: filtra datos del futuro, (después de que el préstamo haya empezado a pagarse)

Todas estas columnas filtran datos del futuro, lo que significa que describen aspectos del préstamo después de que éste haya sido financiado en su totalidad y haya comenzado a ser pagado por el prestatario.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Instrucciones

Utilice el método Dataframe **drop** para eliminar las siguientes columnas del Dataframe loans_2007:

- total_rec_int
- total_rec_late_fee
- Recoveries
- collection_recovery_fee
- last_pymnt_d
- last_pymnt_amnt

Utilice la función print para mostrar la primera fila de loans_2007 y el número de columnas de loans_2007.

Al familiarizarnos con las columnas del conjunto de datos, hemos podido reducir el número de columnas de 52 a 32 columnas. Ahora tenemos que decidir la columna objetivo que queremos utilizar para la modelización.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

We should use the **loan_status** column, since it's the only column that directly describes if a loan was paid off on time, had delayed payments, or was defaulted on the borrower. Currently, this column contains text values and we need to convert it to a numerical value for training a model. Let's explore the different values in this column and come up with a strategy for converting the values in this column.

Instrucciones

- Utilice el método de la serie **value_counts** para devolver la frecuencia de los valores únicos en la columna loan_status
- Mostrar la frecuencia de cada valor único utilizando la función de print

II.2.6 Clasificación Binaria

Hay 8 valores posibles para la columna loan_status. Puedes leer sobre la mayoría de los diferentes estados de los préstamos en el **sitio web de Lending Clube**. Los dos valores que comienzan con "No cumple con la política de crédito" no se explican por desgracia. Una rápida búsqueda en Google nos lleva a explicaciones de la comunidad de prestamistas **aquí.**

Hemos recopilado la explicación de cada columna, así como los recuentos en el marco de datos, en la siguiente tabla:

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Loan Status	Count	Significado
Fully Paid	33136	El préstamo ha sido totalmente pagado
Charged Off	5634	Préstamo para el cual ya no hay una expectativa razonable de pagos adicionales
Does not meet the credit policy. Status: Fully Paid	1988	Si bien el préstamo se pagó, la solicitud de préstamo de hoy ya no cumpliría con la política de crédito y no se aprobaría en el mercado
Does not meet the credit policy. Status: Charged Off	761	Si bien el préstamo fue cobrado, la solicitud de préstamo de hoy ya no cumpliría con la política de crédito y no sería aprobada en el mercado.
In Grace Period	20	El préstamo está vencido pero aún en el período de gracia de 15 días
Late (16-30 days)	8	El préstamo no se ha pagado en 16 a 30 días (retraso en el pago actual)
Late (31-120 days)	24	El préstamo no se ha pagado en 31 a 120 días (retraso en el pago actual)
Current	961	El préstamo está al día en los pagos actuales
Default	3	El préstamo está incumplido y no se ha realizado ningún pago durante más de 121 días

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Desde la perspectiva del inversor, lo que nos interesa es tratar de predecir si los préstamos se pagarán a tiempo. Sólo los valores Fully Paid (Préstamo Pagado) y Charged Off (Préstamo Castigado/No Pagado) describen el resultado final del préstamo. Los demás valores describen los préstamos que aún están en curso y en los que todavía no se sabe si el prestatario pagará el préstamo a tiempo o no. Mientras que el Default (estado de impago) se asemeja al estado de "Charged Off", a los ojos de Lending Club, los préstamos "charged off" no tienen prácticamente ninguna posibilidad de ser devueltos, mientras que los "default" tienen una pequeña posibilidad.

Como nos interesa poder predecir en cuál de estos dos valores caerá un préstamo, podemos tratar el problema como una clasificación binaria. Eliminemos todos los préstamos que no contengan el estado de Fully Paid (Préstamo Pagado) or Charged Off («Totalmente Pagado" o " Préstamo Castigado/No Pagado"). Una vez eliminados los estados del préstamo, transformemos los valores de Fully Paid en 1 para el caso positivo y los valores de Charged Off en 0 para el caso negativo. Aunque hay varias formas de transformar todos los valores de una columna, utilizaremos **el método Dataframe replace (Dataframe method replace)**. Según la documentación, podemos pasar al método replace un diccionario de mapeo anidado con el siguiente formato:

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

```
mapping_dict = {
    "date": {
        "january": 1,
        "february": 2,
        "march": 3
    }
}
df = df.replace(mapping_dict)
```

Por último, hay que tener en cuenta el **desequilibrio de clases** entre los casos positivos y los negativos. Mientras que hay 33.136 préstamos que han sido totalmente pagados, sólo hay 5.634 que fueron imputados. Este desequilibrio de clases es un problema común en la clasificación binaria y, durante el entrenamiento, el modelo acaba teniendo un fuerte sesgo hacia la predicción de la clase con más observaciones en el conjunto de entrenamiento y rara vez predice la clase con menos observaciones. Cuanto más fuerte sea el desequilibrio, más sesgado estará el modelo. Hay varias formas de abordar este desequilibrio de clases, que exploraremos más adelante.

II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Instrucciones

- Eliminar todas las filas de loans_2007 que contengan valores distintos de Fully Paid o Charged Off para la columna loan_status
- Utilice el método Dataframe **replace** para sustituir
  - Fully Paid (Préstamo Pagado) por 1
  - Charged off (Préstamo Castigado/No Pagado) con 0

II.2.7 Remover columnas de valor único

Para terminar esta lección, busquemos cualquier columna que contenga un solo valor único y eliminémosla. Estas columnas no serán útiles para el modelo ya que no añaden ninguna información a cada solicitud de préstamo. Además, la eliminación de estas columnas reducirá el número de columnas que tendremos que explorar en el futuro.

Tendremos que calcular el número de valores únicos de cada columna y eliminar las columnas que sólo contienen un valor único. Mientras que el método Series unique devuelve los valores únicos en una columna, también cuenta el objeto Pandas missing value nan como un valor:

```
# Returns 0 and nan.
unique_values = loans['tax_liens'].unique()
```

Como estamos tratando de encontrar columnas que contengan un valor único verdadero, primero debemos eliminar los valores nulos y luego calcular el número de valores únicos:

```
non_null = loans_2007['tax_liens'].dropna()
unique_non_null = non_null.unique()
num_true_unique = len(unique_non_null)
```


II.1 Recorrido por el Proyecto de Aprendizaje Automático: Depuración de Datos

Instrucciones

- Elimine cualquier columna de loans_2007 que contenga un único valor:
  - Cree una lista vacía, drop_columns para llevar la cuenta de las columnas que desea eliminar
  - Para cada columna:
    - Utilice el método **dropna** de la serie para eliminar cualquier valor nulo y luego utilice el método **unique** de la serie para devolver el conjunto de valores únicos no nulos
    - Utilice la función len() para devolver el número de valores en ese conjunto
    - Añada la columna a drop_columns si sólo contiene un valor único
  - Utilice el método Dataframe **drop** para eliminar las columnas de drop_columns de loans_2007
- Utilizar la función print para mostrar drop_columns y saber cuáles se han eliminado

II.2.1 Recapitulación

Puede haber aprendido a eliminar todas las columnas que contenían información redundante, que no eran útiles para el modelado, que requerían demasiado procesamiento para ser útiles o que filtraban información del futuro. Después de exportar el Dataframe a un archivo CSV llamado filtered_loans_2007.csv para diferenciar el archivo con el loans_2007.csv. En esta lección, prepararemos los datos para el aprendizaje automático centrándonos en el manejo de los valores perdidos, la conversión de las columnas categóricas en columnas numéricas y la eliminación de cualquier otra columna extraña que encontremos a lo largo de este proceso.

Las matemáticas subyacentes a la mayoría de los modelos de aprendizaje automático asumen que los datos son numéricos y no contienen valores perdidos. Para reforzar este requisito, **scikit-learn** devolverá un error si se intenta entrenar un modelo utilizando datos que contienen valores perdidos o valores no numéricos cuando se trabaja con modelos como la regresión lineal y la regresión logística.

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Empecemos por calcular el número de valores que faltan e ideemos una estrategia para manejarlos. Luego, nos centraremos en las columnas categóricas.

Podemos devolver el número de valores perdidos en el Dataframe de la siguiente manera:

- Primero utilizando el método de Pandas Dataframe **isnull** para devolver un Dataframe que contenga valores booleanos:
  - True si el valor original es nulo
  - Falso si el valor original no es nulo
- A continuación, utilizando el método Pandas Dataframe **sum** para calcular el número de valores nulos en cada columna

```
null_counts = df.isnull().sum()
```

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Instrucciones

- Leer en filtered_loans_2007.csv como un Dataframe y asignarlo a los loans (préstamos)
- Utilice los métodos isnull y sum para devolver el número de valores nulos en cada columna. Asigne el objeto Series resultante a null_counts
- Utilice la función de print para mostrar las filas de null_counts que son mayores que cero

Soluciones

```
1 import pandas as pd
2 loans = pd.read_csv('filtered_loans_2007.csv')
3 null_counts = loans.isnull().sum()
4 print(null_counts[null_counts>0])
```


II.2.2 Tratamiento de los valores perdidos

En la pantalla anterior obtuvimos una serie que muestra cuántos valores perdidos tiene cada columna con valores perdidos:

```
emp_length              1036
title                     11
revol_util                50
last_credit_pull_d         2
pub_rec_bankruptcies     697
```

Aunque la mayoría de las columnas no tienen valores perdidos, dos columnas tienen cincuenta o menos filas con valores perdidos, y dos columnas, emp_length y pub_rec_bankruptcies, contienen un número relativamente alto de valores perdidos.

El conocimiento del dominio nos dice que la duración del empleo se utiliza con frecuencia para evaluar el riesgo de un posible prestatario, por lo que mantendremos esta columna a pesar de su número relativamente elevado de valores perdidos.

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Inspeccionemos los valores de la columna pub_rec_bankruptcies.

```
print(loans.pub_rec_bankruptcies.value_counts(normalize=Tr
ue, dropna=False))
```

```
0.0    0.939438
 1.0    0.042456
NaN     0.017978
 2.0    0.000129
Name: pub_rec_bankruptcies, dtype: float64
```

Vemos que esta columna ofrece muy poca variabilidad, casi el 94% de los valores están en la misma categoría. Probablemente no tenga mucho valor predictivo. Vamos a eliminarla. Además, eliminaremos las filas restantes que contienen valores nulos.

Esto significa que mantendremos las siguientes columnas y sólo eliminaremos las filas que contengan valores perdidos para ellas:

- emp_length
- Title
- revol_útil
- last_credit_pull_d

Después de eliminar las filas que contienen valores perdidos, elimine la columna pub_rec_bankruptcies por completo.

Utilicemos la estrategia de eliminar primero la columna pub_rec_bankruptcies y luego eliminar todas las filas que contengan valores perdidos para cubrir estos dos casos. De este modo, sólo eliminamos las filas que contienen valores perdidos para las columnas emp_length, title y revol_util, pero no la columna pub_rec_bankruptcies.

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Instrucciones

- Utilice el **método drop (drop method)** para eliminar la columna pub_rec_bankruptcies de los préstamos
- Utilice el **método dropna (dropna method)** para eliminar todas las filas de los préstamos que contengan algún valor perdido
- Utilice el atributo dtypes seguido del método value_counts() para devolver los recuentos de cada tipo de dato de columna.
- Utilice la función print para mostrar estos recuentos

Soluciones

```
loans = loans.drop("pub_rec_bankruptcies", axis=1)
loans = loans.dropna(axis=0)
print(loans.dtypes.value_counts())
```


II.2.3 Columnas de texto

Mientras que las columnas numéricas se pueden utilizar de forma nativa con scikit-learn, las columnas de objetos que contienen texto necesitan ser convertidas a tipos de datos numéricos. Devolvamos un nuevo dataframe que contenga sólo las columnas de objetos para poder explorarlas en mayor profundidad. Se puede utilizar el método de dataframe **select_dtypes** para seleccionar sólo las columnas de un determinado tipo de datos:

```
float_df = df.select_dtypes(include=['float'])
```

Seleccionemos sólo las columnas de los objetos y luego mostremos una fila de muestra para tener una mejor idea de cómo están formateados los valores de cada columna.

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Instrucciones

- Utilizar el método dataframe select_dtypes para seleccionar sólo las columnas de tipo object (objeto) de los loans (préstamos) y asignar al Dataframe resultante object_columns_df
- Mostrar la primera fila de object_columns_df utilizando la función print

Soluciones

```
1 object_columns_df = loans.select_dtypes(include=
  ["object"])
2 print(object_columns_df.iloc[0])
```


II.2.4 Conversión de columnas de texto

Algunas de las columnas parecen representar valores categóricos, pero deberíamos confirmarlo comprobando el número de valores únicos en esas columnas:

- **home_ownership:** estado de propiedad de la vivienda, sólo puede ser 1 de 4 valores categóricos según el diccionario de datos
- **verification_status:** indica si los ingresos fueron verificados por Lending Club
- **emp_length:** número de años que el prestatario estaba empleado en el momento de la solicitud
- **term:** número de pagos del préstamo, ya sea 36 o 60
- **addr_state:** estado de residencia del prestatario
- **purpose:** categoría proporcionada por el prestatario para la solicitud de préstamo
- **title:** título del préstamo facilitado por el prestatario

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

También hay dos columnas que representan valores numéricos y que hay que convertir:

- int_rate: tipo de interés del préstamo en %
- revol_util: tasa de utilización de la línea rotatoria o la cantidad de crédito que el prestatario está utilizando en relación con todo el crédito disponible, lea más **aquí**

Basándonos en los valores de la primera fila para el purpose(propósito) y el title(título), parece que estas columnas podrían reflejar la misma información. Exploremos los recuentos de valores únicos por separado para confirmar si esto es cierto.

Por último, algunas de las columnas contienen valores de fecha que requerirían una buena cantidad de ingeniería de características para que fueran potencialmente útiles:

- earliest_cr_line: El mes en que se abrió la primera línea de crédito del prestatario
- last_credit_pull_d: El mes más reciente en que Lending Club retiró el crédito para este préstamo

Dado que estas características de fecha requieren algo de ingeniería de características para fines de modelado, eliminemos estas columnas de fecha del marco de datos.

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

II.2.5 Primeras 5 columnas categóricas

Exploremos los recuentos de valores únicos de las columnas que parecen contener valores categóricos.

Instrucciones

- Mostrar los recuentos de valores únicos para las siguientes columnas: home_ownership, verification_status, emp_lenght, term, addr state columns:
  - Guarda estos nombres de columnas en una lista llamada cols
  - Utiliza un bucle **"for"** para iterar sobre las cols:
    - Utilice la función de **print** combinada con el método Series value_counts para mostrar los recuentos de valores únicos de cada columna

Soluciones

```
cols = ['home_ownership', 'verification_status',
'emp_length', 'term', 'addr_state']
for c in cols:
    print(loans[c].value_counts())
```

II.2.6 El motivo del préstamo

Las columnas home_ownership, verification_status, emp_length, term y addr_state contienen múltiples valores discretos. Deberíamos limpiar la columna emp_length y tratarla como numérica, ya que los valores tienen ordenación (2 años de empleo son menos de 8 años).

En primer lugar, veamos los recuentos de valores únicos de las columnas purpose (propósito) y title (título) para saber qué columna queremos conservar.

Instrucciones

Utilice el método value_counts y la función print para mostrar los valores únicos en las siguientes columnas:

- Title
- purpose

Soluciones

```
print(loans["title"].value_counts())
print(loans["purpose"].value_counts())
```

II.2.7 Columnas categóricas

Las columnas home_ownership, verification_status, emp_length y term contienen cada una unos valores categóricos discretos. Deberíamos codificar estas columnas como variables ficticias y mantenerlas.

Parece que las columnas purpose (propósito) y title (título) contienen información que se solapa, pero mantendremos la columna purpose (propósito) ya que contiene algunos valores discretos. Además, la columna de title (título) tiene problemas de calidad de datos, ya que muchos de los valores se repiten con ligeras modificaciones (por ejemplo, Debt Consolidation and Debt Consolidation Loan and debt consolidation - Consolidación de deuda y Préstamo de consolidación de deuda y consolidación de deuda).

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Podemos utilizar el siguiente mapeo para limpiar la columna emp_length:

- "10+ years": 10
- "9 years": 9
- "8 years": 8
- "7 years": 7
- "6 years": 6
- "5 years": 5
- "4 years": 4
- "3 years": 3
- "2 years": 2
- "1 year": 1
- "< 1 year": 0
- "n/a": 0

Hemos sido precavidos con los mapeos de más de 10 años, < 1 año y n/a. Suponemos que las personas que pueden haber trabajado más de 10 años sólo han trabajado realmente durante 10 años. También asumimos que las personas que han trabajado menos de un año o si la información no está disponible que han trabajado durante 0. Esta es una heurística general pero no es perfecta.

Por último, la columna addr_state contiene muchos valores discretos, y tendríamos que añadir 49 columnas de variables ficticias para utilizarla en la clasificación. Esto haría nuestro marco de datos mucho más grande y podría ralentizar la velocidad de ejecución del código. Vamos a eliminar esta columna de la consideración.

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Instrucciones

- Eliminar las columnas last_credit_pull_d, addr_state, title y earliest_cr_line de los loans (préstamos)
- Convertir las columnas int_rate y revol_util en columnas float mediante:
  - Utilizando el accesorio str seguido del método de cadena rstrip para eliminar el signo de porcentaje (%) de la derecha:
    - loans['int_rate'].str.rstrip('%') devuelve una nueva Serie con % eliminado del lado derecho de cada valor
  - En el objeto Series resultante, utilice el método astype para convertirlo al tipo float
  - Asigne la nueva serie de valores flotantes a las respectivas columnas del marco de datos
- Utilice el método replace para limpiar la columna emp_length

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Soluciones

```
mapping_dict = {
    "emp_length": {
        "10+ years": 10,
        "9 years": 9,
        "8 years": 8,
        "7 years": 7,
        "6 years": 6,
        "5 years": 5,
        "4 years": 4,
        "3 years": 3,
        "2 years": 2,
        "1 year": 1,
        "< 1 year": 0,
loans = loans.drop(["last_credit_pull_d",
"earliest_cr_line", "addr_state", "title"], axis=1)
loans["int_rate"] =
loans["int_rate"].str.rstrip("%").astype("float")
loans["revol_util"] =
loans["revol_util"].str.rstrip("%").astype("float")
loans = loans.replace(mapping_dict)
```

II.2.8 Variables ficticias (Dummy)

Ahora vamos a codificar las columnas home_ownership, verification_status, purpose y term como variables dummy para poder utilizarlas en nuestro modelo. Primero tenemos que utilizar el método Pandas **get_dummies** para devolver un nuevo Dataframe que contenga una nueva columna para cada variable ficticia:

```
# Returns a new Dataframe containing 1 column for each
dummy variable.
dummy_df = pd.get_dummies(loans[["term",
"verification_status"]])
```

A continuación, podemos utilizar el método **concat** para añadir estas columnas ficticias al Dataframe original:

```
loans = pd.concat([loans, dummy_df], axis=1)
```

Y a continuación, suelte las columnas originales por completo utilizando el método de la caída (Drop Method):

```
loans = loans.drop(["verification_status", "term"],
axis=1)
```

II.2 Recorrido por el Proyecto de Aprendizaje Automático: Preparación de las Características

Instrucciones

- Codifique las columnas home_ownership, verification_status, purpose y term como valores enteros:
  - Utilice la función **get_dummies** para devolver un Dataframe que contenga las columnas ficticias
  - Utilice el método **concat** para añadir estas columnas ficticias a los loans (préstamos)
  - Elimine las columnas originales no ficticias (home_ownership, verification_status, purpose y term) de los loans (préstamos)

Soluciones

```
1 cat_columns = ["home_ownership", "verification_status",
  "purpose", "term"]
2 dummy_df = pd.get_dummies(loans[cat_columns])
3 loans = pd.concat([loans, dummy_df], axis=1)
4 loans = loans.drop(cat_columns, axis=1)
```

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

II.3.1 Recapitular

Nuestro objetivo es generar características a partir de los datos, que podemos introducir en un algoritmo de aprendizaje automático. El algoritmo hará predicciones sobre si un préstamo se pagará o no a tiempo, lo cual está contenido en la columna loan_status del conjunto de datos limpio.

Cuando preparamos los datos, eliminamos las columnas que tenían problemas de fuga de datos, que contenían información redundante o que requerían un procesamiento adicional para convertirse en características útiles. Limpiamos las características que tenían problemas de formato y convertimos las columnas categóricas en variables ficticias.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

En la última lección, nos dimos cuenta de que hay un desequilibrio de clases en nuestra columna objetivo, loan_status. Hay aproximadamente 6 veces más préstamos que fueron pagados a tiempo (caso positivo, etiqueta de 1) que los que no lo fueron (caso negativo, etiqueta de 0). Los desequilibrios pueden causar problemas con muchos algoritmos de aprendizaje automático, donde parecen tener una alta precisión, pero en realidad no están aprendiendo de los datos de entrenamiento. Debido a su potencial para causar problemas, debemos tener en cuenta el desequilibrio de clases cuando construyamos modelos de aprendizaje automático.

Después de toda nuestra limpieza de datos, terminamos con el archivo csv llamado clean_loans_2007.csv. Vamos a leer este archivo en un marco de datos y ver un resumen del trabajo que hicimos.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Instrucciones

- Leer clean_loans_2007.csv en un Dataframe llamado loans
- Utilice el método info() y la función print para mostrar un resumen del conjunto de datos

Soluciones

```
1 import pandas as pd
2 loans = pd.read_csv("clean_loans_2007.csv")
3 print(loans.info())
```


II.3.2 Cómo hacer una métrica del error

Antes de sumergirnos en la predicción de loan_status con aprendizaje automático, volvamos a nuestros primeros pasos cuando empezamos a limpiar el conjunto de datos de Lending Club. Quizá recuerdes la pregunta original que queríamos responder:

- ¿Podemos construir un modelo de aprendizaje automático que pueda predecir con precisión si un prestatario pagará su préstamo a tiempo o no?

Establecimos que este es un problema de clasificación binaria y convertimos la columna loan_status en 0s y 1s como resultado. Antes de entrar en materia y seleccionar un algoritmo para aplicarlo a los datos, debemos seleccionar una métrica de error.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Una métrica de error nos ayudará a averiguar cuándo nuestro modelo está funcionando bien y cuándo está funcionando mal. Para relacionar las métricas de error con la pregunta original que queríamos responder, digamos que estamos utilizando un modelo de aprendizaje automático para predecir si debemos o no financiar un préstamo en la plataforma de Lending Club. Nuestro objetivo es ganar dinero: queremos financiar suficientes préstamos que se paguen a tiempo para compensar nuestras pérdidas por los préstamos que no se pagan. Una métrica de errores nos ayudará a determinar si nuestro algoritmo nos hará ganar o perder dinero.

En este caso, nos preocupan principalmente los falsos positivos y los falsos negativos. Ambos son diferentes tipos de errores de clasificación. Con un falso positivo, predecimos que un préstamo se pagará a tiempo, pero en realidad no es así. Esto nos cuesta dinero, ya que financiamos préstamos que nos hacen perder dinero. Con un falso negativo, predecimos que un préstamo no se pagará a tiempo, pero en realidad se pagaría a tiempo. Esto nos hace perder dinero potencial, ya que no financiamos un préstamo que realmente se habría pagado.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

He aquí un diagrama para simplificar los conceptos:

loan_status actual	prediction	error type
0	1	False Positive
1	1	True positive
0	0	True negative
1	0	False Negative

En las columnas loan_status y prediction, un 0 significa que el préstamo no se pagaría a tiempo, y un 1 significa que sí.

Como estamos viendo este problema desde el punto de vista de un inversor conservador, tenemos que tratar los falsos positivos de forma diferente a los falsos negativos. Un inversor conservador querría minimizar el riesgo y evitar los falsos positivos en la medida de lo posible. Estaría más seguro de perder oportunidades (falsos negativos) que de financiar un préstamo arriesgado (falsos positivos).

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Calculemos los falsos positivos y los verdaderos positivos en Python. Podemos usar múltiples condicionales, separados por un & para seleccionar elementos en un array de NumPy que cumplan ciertas condiciones. Por ejemplo, si tuviéramos una matriz llamada predictions (predicciones), podríamos seleccionar los elementos de las predictions (predicciones) que sean iguales a 1 y donde los elementos de préstamos ["loan_status"] (estado del prestamo) en la misma posición también sean iguales a 1 utilizando esto:

```
tp_filter = (predictions == 1) & (loans["loan_status"] == 1)
predictions[tp_filter]
```

El código anterior nos dará todos los elementos de las predictions que son verdaderos positivos, es decir, los casos en los que predijimos que el préstamo se pagaría a tiempo y realmente se pagó a tiempo. Utilizando la función len para encontrar el número de elementos, podemos encontrar el número de verdaderos positivos.

Utilizando el diagrama anterior como referencia, es posible calcular las otras 3 cantidades que hemos mencionado: falsos positivos, verdaderos negativos y falsos negativos.

Hemos generado algunas predicciones automáticamente y se almacenan en un array de NumPy llamado predictions.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Instrucciones

- Encontrar el número de verdaderos negativos
- Encuentre el número de elementos en los que predictions son 0, y la entrada correspondiente en loans["loan_status"] también es 0
- Asignar el resultado a tn
- Encontrar el número de verdaderos positivos
- Encuentre el número de elementos en los que predictions son 1, y la entrada correspondiente en loans["loan_status"] también es 1
- Asignar el resultado a tp
- Encontrar el número de falsos negativos
- Encuentre el número de elementos en los que predictions son 0, y la entrada correspondiente en loans["loan_status"] también es 1
- Asignar el resultado a fn
- Encontrar el número de falsos positivos
- Encuentre el número de elementos en los que predictions son 1, y la entrada correspondiente en loans["loan_status"] también es 0
- Asignar el resultado a fp

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Soluciones

```
import pandas as pd
# False positives.
fp_filter = (predictions == 1) & (loans["loan_status"] == 0)
fp = len(predictions[fp_filter])

# True positives.
tp_filter = (predictions == 1) & (loans["loan_status"] == 1)
tp = len(predictions[tp_filter])

# False negatives.
fn_filter = (predictions == 0) & (loans["loan_status"] == 1)
fn = len(predictions[fn_filter])

# True negatives
tn_filter = (predictions == 0) & (loans["loan_status"] == 0)
tn = len(predictions[tn_filter])
```


II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

II.3.3 Desequilibrio de clases

Ya hemos mencionado que existe un importante desequilibrio de clases en la columna loan_status. Hay 6 veces más préstamos que fueron pagados a tiempo (1), que préstamos que no fueron pagados a tiempo (0). Esto provoca un problema importante cuando utilizamos la precisión como métrica. Debido al desequilibrio de clases, un clasificador puede predecir 1 para cada fila, y aún así tener una alta precisión. Este es un diagrama que ilustra el concepto:

loan_status actual	prediction
0	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

En el diagrama anterior, nuestras predicciones tienen una precisión del 85,7%: hemos identificado correctamente loan_status en el 85,7% de los casos. Sin embargo, hemos hecho esto prediciendo 1 para cada fila. Lo que esto significa es que realmente perderemos dinero. Digamos que prestamos 1000 dólares de media a cada prestatario. Cada prestatario nos devuelve el 10% de interés. Obtendremos un beneficio previsto de 100 dólares por cada préstamo. En el diagrama anterior, en realidad perderíamos dinero:

loan_status actual	prediction	profit/loss
0	1	-1000
1	1	100
1	1	100
1	1	100
1	1	100
1	1	100

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Como puedes ver, ganamos 600 dólares en intereses de los prestatarios que nos devolvieron el dinero, pero perdimos 1.000 dólares con el único prestatario que nunca nos devolvió el dinero, así que en realidad acabamos perdiendo 400 dólares en total, aunque nuestro modelo es técnicamente preciso.

Por eso es importante tener siempre en cuenta las clases desequilibradas en los modelos de aprendizaje automático y ajustar la métrica de error en consecuencia. En este caso, no queremos usar la precisión y deberíamos usar en su lugar métricas que nos digan el número de falsos positivos y falsos negativos.

Esto significa que deberíamos optimizar para:

- Una **alta recuperación (recall)** (tasa de verdaderos positivos)
- **Baja caída** (tasa de falsos positivos) (**fall-out**)

Podemos calcular la tasa de falsos positivos y la tasa de verdaderos positivos, utilizando los números de verdaderos positivos, verdaderos negativos, falsos negativos y falsos positivos.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

La tasa de falsos positivos es el número de falsos positivos dividido por el número de falsos positivos más el número de verdaderos negativos. Esto divide todos los casos en los que pensamos que un préstamo se pagaría por todos los préstamos que no se pagaron:

```
fpr = fp / (fp + tn)
```

La tasa de verdaderos positivos es el número de verdaderos positivos dividido por el número de verdaderos positivos más el número de falsos negativos. Esto divide todos los casos en los que pensamos que un préstamo se pagaría por todos los préstamos que se pagaron:

```
tpr = tp / (tp + fn)
```

$$FPR = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

$$TPR = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Las formas sencillas de pensar en cada término son:

- False Positive Rate (Tasa de falsos positivos): "el porcentaje de los préstamos que no deberían financiarse y que yo financiaría"
- True Positive Rate (Tasa de verdaderos positivos): "el porcentaje de préstamos que deberían financiarse y que yo financiaría"

Por lo general, si reducimos la tasa de falsos positivos, la tasa de verdaderos positivos también bajará. Esto se debe a que si queremos reducir el riesgo de falsos positivos, no pensaríamos en financiar préstamos más arriesgados en primer lugar.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Instrucción

- Calcular la tasa de falsos positivos de las predictions (predicciones)
- Calcule el número de falsos positivos y divídalo por el número de falsos positivos más el número de verdaderos negativos
- Asignar a fpr
- Calcular la tasa de verdaderos positivos de las predictions (predicciones)
- Calcular el número de verdaderos positivos y dividirlo por el número de verdaderos positivos más el número de falsos negativos
- Asignar a tpr
- Imprimir fpr y tpr para verificar

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

```
import pandas as pd
import numpy

# Predict that all loans will be paid off on time.
predictions = pd.Series(numpy.ones(loans.shape[0]))
# False positives.
fp_filter = (predictions == 1) & (loans["loan_status"] == 0)
fp = len(predictions[fp_filter])

# True positives.
tp_filter = (predictions == 1) & (loans["loan_status"] == 1)
tp = len(predictions[tp_filter])

# False negatives.
fn_filter = (predictions == 0) & (loans["loan_status"] == 1)
fn = len(predictions[fn_filter])
# True negatives
tn_filter = (predictions == 0) & (loans["loan_status"] == 0)
tn = len(predictions[tn_filter])

# Rates
tpr = tp / (tp + fn)
fpr = fp / (fp + tn)

print(tpr)
print(fpr)
```

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

II.3.4 Regresión Logistica

En la última pantalla, puede haber notado que tanto fpr como tpr eran 1. Esto se debe a que hemos predicho 1 para cada fila. Esto significa que hemos identificado correctamente todos los préstamos buenos (tasa de verdaderos positivos), pero también hemos identificado incorrectamente todos los préstamos malos (tasa de falsos positivos). Ahora que hemos configurado las métricas de error, podemos pasar a hacer predicciones utilizando un algoritmo de aprendizaje automático.

Como vimos en la primera pantalla de la misión, nuestro conjunto de datos depurado contiene 41 columnas, todas ellas de tipo int64 o float64. No hay valores nulos en ninguna de las columnas. Esto significa que ahora podemos aplicar cualquier algoritmo de aprendizaje automático a nuestro conjunto de datos. La mayoría de los algoritmos no pueden tratar con valores no numéricos o faltantes, por lo que tuvimos que hacer mucha limpieza de datos.

Para ajustar los modelos de aprendizaje automático, utilizaremos la biblioteca **Scikit-learn**. Aunque hemos construido nuestras propias implementaciones de algoritmos en misiones anteriores, es más fácil y rápido utilizar algoritmos que alguien ya ha escrito y ajustado para obtener un alto rendimiento.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Un buen primer algoritmo para aplicar a los problemas de clasificación binaria es la regresión logística (**logistic regression)**, por las siguientes razones:

- Es rápido de entrenar y podemos iterar más rápidamente
- Es menos propenso al sobreajuste que otros modelos más complejos como los árboles de decisión
- Es fácil de interpretar

Instrucciones

- Cree un marco de datos llamado features que contenga sólo las columnas de características
- Elimine la columna loan_status
- Cree una serie denominada target que contenga sólo la columna target (loan_status)
- Utilice el método **fit** de lr para ajustar una regresión logística a las features (características) y al target (objetivo)
- Utilice el método **predict** de lr para hacer predicciones sobre las features (características). Asigne las predicciones a predictions

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Soluciones

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
cols = loans.columns
train_cols = cols.drop("loan_status")
features = loans[train_cols]
target = loans["loan_status"]
lr.fit(features, target)
predictions = lr.predict(features)
```


II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

II.3.6 Validación cruzada

Aunque generamos predicciones en la última pantalla, esas predicciones estaban sobreajustadas. Estaban sobreajustadas porque generamos predicciones utilizando los mismos datos con los que entrenamos nuestro modelo. Cuando usamos esto para evaluar un error, obtenemos una representación irreal de la precisión del algoritmo, porque ya "conoce" las respuestas correctas. Es como pedirle a alguien que memorice un montón de ecuaciones físicas y luego pedirle que introduzca números en las ecuaciones. Puede decir la respuesta correcta, pero no puede explicar un concepto para el que no ha memorizado una ecuación.

Para obtener una representación realista de la exactitud del modelo, vamos a realizar una validación cruzada k-fold (**k-fold cross validation**). Podemos utilizar la función **cross_val_predict() function** del paquete **sklearn.model_selection**. Este es el aspecto del flujo de trabajo:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict

# Instantiate model object.
lr = LogisticRegression()
# Make predictions using 3-fold cross-validation.
cross_val_predict(lr, features, target, cv=3)
```

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Una vez que hemos validado las predicciones de forma cruzada, podemos calcular la tasa de verdaderos positivos y la tasa de falsos positivos.

Instrucciones

- Generar predicciones de validación cruzada para las (características)
- Llame a **cross_val_predict** utilizando lr, features (características) y target (objetivo)
- Establezca el parámetro cv en 3, para que se realice una validación cruzada triple
- Asigne las predicciones a predictions
- Utilice la clase **Series** para convertir predictions en un objeto pandas Series
- Calcule la tasa de verdaderos positivos y la tasa de falsos positivos
- Asigne la tasa de verdaderos positivos a tpr
- Asignar la tasa de falsos positivos a fpr
- Mostrar fpr y tpr para evaluarlos

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Soluciones

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict
lr = LogisticRegression()
predictions = cross_val_predict(lr, features, target, cv=3)
predictions = pd.Series(predictions)
# False positives.
fp_filter = (predictions == 1) & (loans["loan_status"] == 0)
fp = len(predictions[fp_filter])

# True positives.
tp_filter = (predictions == 1) & (loans["loan_status"] == 1)
tp = len(predictions[tp_filter])
# False negatives.
fn_filter = (predictions == 0) & (loans["loan_status"] == 1)
fn = len(predictions[fn_filter])

# True negatives
tn_filter = (predictions == 0) & (loans["loan_status"] == 0)
tn = len(predictions[tn_filter])
# Rates
tpr = tp  / (tp + fn)
fpr = fp  / (fp + tn)
print(tpr)
print(fpr)
```


II.3.7 Penalización del clasificador

Como se puede ver en la última pantalla, nuestros fpr y tpr están alrededor de lo que esperaríamos si el modelo estuviera prediciendo todos los unos. Podemos mirar las primeras filas de predicciones para confirmar:

```
0    1
1    1
2    1
3    1
4    1
5    1
6    1
7    1
8    1
9    1
```

Por desgracia, aunque no estemos utilizando la precisión como métrica de error, el clasificador sí lo hace, y no tiene en cuenta el desequilibrio de las clases. Hay algunas formas de conseguir que un clasificador corrija el desequilibrio de las clases. Las dos formas principales son:

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

- Utilizar el sobremuestreo y el submuestreo para garantizar que el clasificador reciba una entrada con un número equilibrado de cada clase
- Decirle al clasificador que penalice las clasificaciones erróneas de la clase menos prevalente más que la otra clase

Primero veremos el sobremuestreo y el submuestreo. Se trata de tomar una muestra que contenga el mismo número de filas en las que loan_status (estado de préstamo) es 0 y en las que loan_status es 1. De este modo, el clasificador se ve obligado a realizar predicciones reales, ya que predecir todos los 1 o todos los 0 sólo dará lugar a una precisión del 50% como máximo.

El inconveniente de esta técnica es que, al tener que preservar una proporción igual, tiene que:

- Desechar muchas filas de datos. Si quisiéramos un número igual de filas en las que el estado del préstamo es 0 y en las que el estado del préstamo es 1, una forma de hacerlo es eliminar las filas en las que el estado del préstamo es 1
- Copiar las filas varias veces. Una forma de igualar los 0s y 1s es copiar las filas donde loan_status es 0
- Generar datos falsos. Una forma de igualar los 0s y 1s es generar nuevas filas donde loan_status es 0

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Por desgracia, ninguna de estas técnicas es fácil. El segundo método que mencionamos antes, decirle al clasificador que penalice más ciertas filas, es mucho más fácil de implementar usando scikit-learn.

Podemos hacerlo estableciendo el parámetro class_weight como balanced al crear la instancia LogisticRegression. Esto le dice a scikit-learn que penalice la clasificación errónea de la clase minoritaria durante el proceso de entrenamiento. La penalización significa que el clasificador de regresión logística presta más atención a la clasificación correcta de las filas donde loan_status es 0. Esto disminuye la precisión cuando loan_status es 1, pero aumenta la precisión cuando loan_status es 0.

Al establecer el parámetro class_weight como equilibrado, la penalización se establece como inversamente proporcional a las frecuencias de las clases. Puede leer más sobre el parámetro **aquí**.

Esto significaría que para el clasificador, clasificar correctamente una fila donde loan_status es 0 es 6 veces más importante que clasificar correctamente una fila donde loan_status es 1.

Podemos repetir el procedimiento de validación cruzada que realizamos en la última pantalla, pero con el parámetro class_weight ajustado a equilibrado.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Instrucciones

- Crear una instancia de **LogisticRegression**
  - Recuerde establecer class_weight a balanced
  - Asigne la instancia a lr
- Genere predicciones de validación cruzada para las features
  Llame **cross_val_predict()** usando lr, features, y target
  - Asigne las predicciones a predictions
- Utilizar la clase de **Series** para convertir las predictions en Series de Pandas, as we did in the last screen
- La conversión a objetos Series permite aprovechar el filtrado booleano y las operaciones aritméticas de pandas
- Calcular la tasa de verdaderos positivos y la tasa de falsos positivos
  - Asignar la tasa de verdaderos positivos a tpr
  - Asignar la tasa de falsos positivos a fpr
- Imprima fpr y tpr para evaluarlos

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Soluciones

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict
lr = LogisticRegression(class_weight="balanced")
predictions = cross_val_predict(lr, features, target, cv=3)
predictions = pd.Series(predictions)

# False positives.
fp_filter = (predictions == 1) & (loans["loan_status"] == 0)
fp = len(predictions[fp_filter])

# True positives.
tp_filter = (predictions == 1) & (loans["loan_status"] == 1)
tp = len(predictions[tp_filter])

# False negatives.
fn_filter = (predictions == 0) & (loans["loan_status"] == 1)
fn = len(predictions[fn_filter])

# True negatives
tn_filter = (predictions == 0) & (loans["loan_status"] == 0)
tn = len(predictions[tn_filter])

# Rates
tpr = tp / (tp + fn)
fpr = fp / (fp + tn)

print(tpr)
print(fpr)
```


II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

II.3.8 Penalidades manuales

Hemos mejorado significativamente la tasa de falsos positivos en la última pantalla al equilibrar las clases, lo que ha reducido la tasa de verdaderos positivos. Nuestra tasa de verdaderos positivos se sitúa ahora en torno al 66%, y nuestra tasa de falsos positivos en torno al 39% . Desde el punto de vista de un inversor conservador, es tranquilizador que la tasa de falsos positivos sea más baja, porque significa que podremos hacer un mejor trabajo para evitar los malos préstamos que si financiamos todo. Sin embargo, sólo decidiríamos financiar el 66% del total de los préstamos (tasa de verdaderos positivos), por lo que rechazaríamos inmediatamente una buena cantidad de préstamos.

Podemos intentar reducir aún más la tasa de falsos positivos asignando una penalización más dura por clasificar mal la clase negativa. Mientras que al establecer class_weight en balanced se establecerá automáticamente una penalización basada en el número de 1s y 0s en la columna, también podemos establecer una penalización manual. En la última pantalla, la penalización que scikit-learn impuso por clasificar mal un 0 habría sido de alrededor de 5,89 (ya que hay 5,89 veces más 1s que 0s).

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

También podemos especificar una penalización manualmente si queremos ajustar más los índices. Para ello, tenemos que pasar un diccionario de valores de penalización al parámetro class_weight:

```
penalty = {
    0: 10,
    1: 1
}
lr = LogisticRegression(class_weight=penalty)
```

El diccionario anterior impondrá una penalización de 10 por clasificar mal un 0 y una penalización de 1 por clasificar mal un 1.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Instrucciones

Modifica el código de la última pantalla para cambiar el parámetro class_weight de la cadena "balanced" al diccionario:

```
penalty = {
    0: 10,
    1: 1
}
```

¡Recuerda imprimir los valores fpr y tpr al final!

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

II. Bosque Aleatorio

Parece que la asignación de penalizaciones manuales redujo la tasa de falsos positivos al 9% y, por tanto, disminuyó nuestro riesgo. Hay que tener en cuenta que esto se produce a expensas de la tasa de verdaderos positivos. Aunque tenemos menos falsos positivos, también estamos perdiendo oportunidades de financiar más préstamos y potencialmente ganar más dinero. Dado que estamos enfocando esto como un inversor conservador, esta estrategia tiene sentido, pero vale la pena tener en cuenta las compensaciones.

Aunque podríamos ajustar más las penalizaciones, es mejor pasar a probar un modelo diferente ahora mismo, para obtener mayores ganancias potenciales en la tasa de falsos positivos. Siempre podemos volver atrás e iterar sobre las penalizaciones más adelante.

Probemos un algoritmo más complejo, el bosque aleatorio. Aprendimos sobre los bosques aleatorios en una misión anterior y construimos nuestro propio modelo. Los bosques aleatorios son capaces de trabajar con datos no lineales y aprender condicionales complejos. Las regresiones logísticas sólo pueden trabajar con datos lineales. El entrenamiento de un algoritmo de bosque aleatorio puede permitir una mayor precisión debido a las columnas que se correlacionan de forma no lineal con loan_status.

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Para ello podemos utilizar la clase **RandomForestClassifer** de scikit-learn.

Instrucciones

- Modifique el código de la última pantalla y cambie la **LogisticRegression** por un modelo **RandomForestClassifer**
- Establezca el valor del argumento de la palabra clave random_state en 1, para que las predicciones no varíen debido al azar
- Establezca el valor del argumento de la palabra clave class_weight en balanced, para evitar problemas con las clases desequilibradas
- Recuerde imprimir los valores fpr y tpr al final

II.3 Recorrido por el Proyecto de Aprendizaje Automático: Hacer Predicciones

Soluciones

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import cross_val_predict
rf = RandomForestClassifier(class_weight="balanced",
random_state=1)
predictions = cross_val_predict(rf, features, target,
cv=3)
predictions = pd.Series(predictions)

# False positives.
fp_filter = (predictions == 1) & (loans["loan_status"]
== 0)
fp = len(predictions[fp_filter])

# True positives.`
tp_filter = (predictions == 1) & (loans["loan_status"]
== 1)
tp = len(predictions[tp_filter])

# False negatives.
fn_filter = (predictions == 0) & (loans["loan_status"]
== 1)
fn = len(predictions[fn_filter])

# True negatives
tn_filter = (predictions == 0) & (loans["loan_status"]
== 0)
tn = len(predictions[tn_filter])
```

```
# Rates
tpr = tp / (tp + fn)
fpr = fp / (fp + tn)

print(tpr)
print(fpr)
```


Puntos Clave

Por desgracia, el uso de un clasificador de bosque aleatorio no mejoró nuestra tasa de falsos positivos. Es probable que el modelo esté demasiado centrado en la clase 1, y que siga prediciendo mayoritariamente 1s. Podríamos arreglar esto aplicando una penalización más dura para las clasificaciones erróneas de 0s.

Por último, nuestro mejor modelo tenía una tasa de falsos positivos de casi el 9% y una tasa de verdaderos positivos de casi el 24%. Para un inversor conservador, esto significa que gana dinero siempre que el tipo de interés sea lo suficientemente alto como para compensar las pérdidas derivadas del incumplimiento del 9% de los prestatarios. Además, el grupo del 24% de los prestatarios debe ser lo suficientemente grande como para ganar suficiente dinero en intereses para compensar las pérdidas.

Si hubiéramos elegido aleatoriamente los préstamos a financiar, los prestatarios habrían incumplido el 14,5% de ellos, y nuestro modelo es mejor que eso, aunque estamos excluyendo más préstamos de lo que haría una estrategia aleatoria. Teniendo en cuenta esto, todavía hay bastante margen de mejora:

Puntos Clave

- Podemos ajustar más las penalizaciones
- Podemos probar otros modelos además del bosque aleatorio y la regresión logística
- Podemos utilizar algunas de las columnas que descartamos para generar mejores características
- Podemos ensamblar varios modelos para obtener predicciones más precisas
- Podemos ajustar los parámetros del algoritmo para conseguir un mayor rendimiento

III. Fundamentos de Kaggle

Fundamentos de Kaggle

Aprende cómo empezar y participar en las competiciones de Kaggle con nuestro curso de Fundamentos de Kaggle. Kaggle es un sitio de competiciones de ciencia de datos donde puedes inscribirte para competir con otros científicos de datos y equipos de ciencia de datos para producir el análisis más preciso de un conjunto de datos en particular. La competencia en Kaggle es fuerte, y quedar entre los primeros clasificados en una competición le dará derecho a presumir.

En este curso, competirás en la competición "Titanic" de Kaggle para construir un modelo simple de aprendizaje automático y hacer tu primera presentación en Kaggle. También aprenderás a seleccionar el mejor algoritmo y a ajustar tu modelo para obtener el mejor rendimiento. Trabajarás con múltiples algoritmos como la regresión logística, los vecinos más cercanos y los bosques aleatorios para intentar encontrar el modelo que obtenga la mejor puntuación y te otorgue el mejor rango.

Fundamentos de Kaggle

A lo largo de este curso, aprenderás varios consejos y trucos para competir en las competiciones de Kaggle que te ayudarán a obtener un buen puesto. También aprenderá más sobre los flujos de trabajo de aprendizaje automático eficaces, y sobre cómo utilizar un cuaderno Jupyter para las competiciones de Kaggle.

Al final del curso, tendrás un proyecto de aprendizaje automático completado y el conocimiento que necesitas para sumergirte en otras competiciones de Kaggle y demostrar tus habilidades al mundo.

Al final de este curso, usted será capaz de:

- Construir un modelo simple de aprendizaje automático y hacer su primera presentación en Kaggle
- Crear nuevas características y seleccionar las de mejor rendimiento para mejorar su puntuación
- Trabajar con múltiples algoritmos, incluyendo regresión logística, k vecinos más cercanos y bosque aleatorio
- Cómo seleccionar el mejor algoritmo y ajustar su modelo para obtener el mejor rendimiento

III.1 Cómo Empezar a Usar Kaggle

III.1.1 Introducción a Kaggle

Kaggle es un sitio en el que la gente crea algoritmos y compite contra profesionales del aprendizaje automático de todo el mundo. Tu algoritmo gana la competición si es el más preciso en un conjunto de datos concreto. Kaggle es una forma divertida de practicar tus habilidades de aprendizaje automático.

En esta misión y en las siguientes, vamos a aprender a competir en las competiciones de Kaggle. En esta misión introductoria aprenderemos a:

- Acercarse a una competición de Kaggle
- Explorar los datos de la competición y aprender sobre el tema de la competición
- Preparar los datos para el aprendizaje automático
- Entrenar un modelo
- Medir la precisión de tu modelo
- Preparar y hacer su primera presentación en Kaggle

III.1 Cómo Empezar a Usar Kaggle

Este curso presupone que tienes conocimientos de Python y de la biblioteca pandas. Si necesita aprender sobre estos, le recomendamos nuestros cursos sobre **Python** y **pandas**.

Kaggle ha creado una serie de concursos diseñados para los principiantes. El más popular de estos concursos, y el que vamos a ver, trata de predecir qué pasajeros sobrevivieron al hundimiento del Titanic.

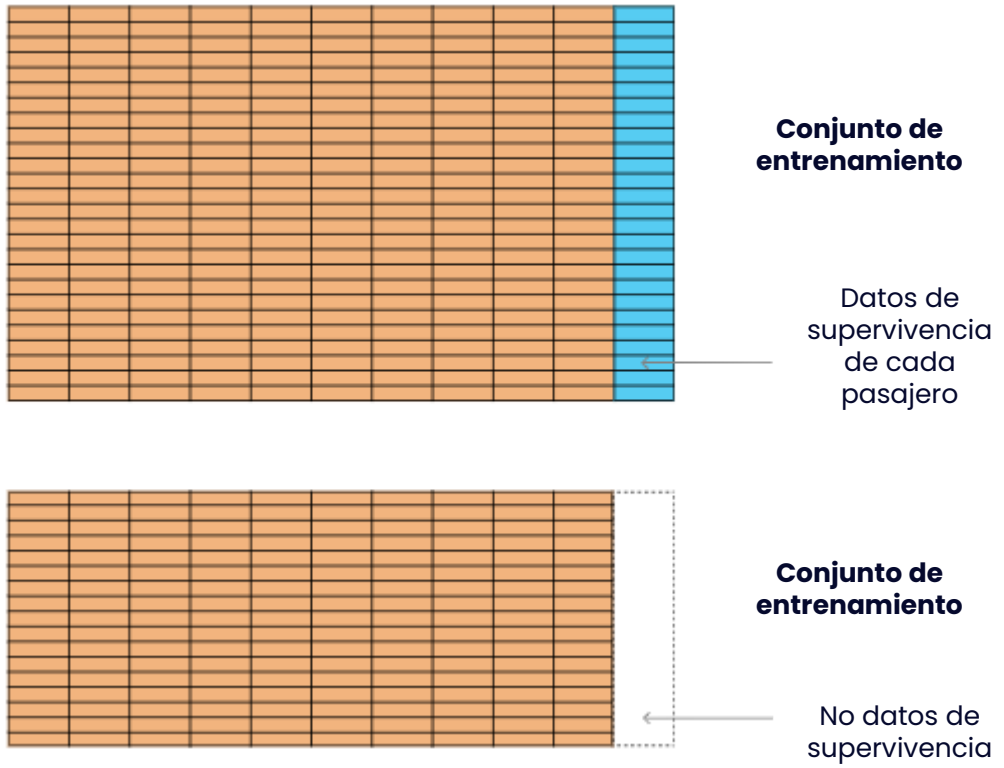
En este concurso, tenemos un conjunto de datos con diferente información sobre los pasajeros a bordo del Titanic, y queremos ver si podemos utilizar esa información para predecir si esas personas sobrevivieron o no. Antes de empezar a ver esta competición específica, tomemos un momento para entender cómo funcionan las competiciones de Kaggle.

III.1 Cómo Empezar a Usar Kaggle

Cada competición de Kaggle tiene dos archivos de datos clave con los que trabajarás: **un conjunto de entrenamiento (Training Set)** y un **conjunto de pruebas (Testing Set)**.

El conjunto de entrenamiento contiene datos que podemos utilizar para entrenar nuestro modelo. Tiene una serie de columnas de características que contienen varios datos descriptivos, así como una columna de los valores objetivo que estamos tratando de predecir: en este caso, la supervivencia (Survival).

El conjunto de pruebas contiene todas las mismas columnas de características, pero falta la columna de valores objetivo. Además, el conjunto de prueba suele tener menos observaciones (filas) que el conjunto de entrenamiento.



III.1 Cómo Empezar a Usar Kaggle

Esto es útil porque queremos la mayor cantidad de datos posible para entrenar nuestro modelo. Una vez que hayamos entrenado nuestro modelo en el conjunto de entrenamiento, utilizaremos ese modelo para hacer predicciones sobre los datos del conjunto de prueba, y enviaremos esas predicciones a Kaggle.

En esta competición, los dos archivos se llaman test.csv y train.csv. Comenzaremos utilizando la biblioteca pandas para leer ambos archivos e inspeccionar su tamaño.

Instrucciones

- Usar pandas.read_csv() para importar train.csv y asignarlo a la variable train
- Utilice DataFrame.shape para calcular el número de filas y columnas en train, y asigne el resultado a train_shape
- Haz clic en ***Run*** para ejecutar tu código, y utiliza el inspector de variables para ver las cuatro variables que acabas de crear

III.1 Cómo Empezar a Usar Kaggle

Soluciones

```
import pandas as pd

test = pd.read_csv("test.csv")
test_shape = test.shape
train = pd.read_csv("train.csv")
train_shape = train.shape
```


III.1 Cómo Empezar a Usar Kaggle

III.1.2 Exploración de Datos

Los archivos que hemos leído en la pantalla anterior están disponibles en la **página de datos del concurso Titanic en Kaggle**. Esa página también tiene un diccionario de datos, que explica las distintas columnas que componen el conjunto de datos. A continuación se muestran las descripciones contenidas en ese diccionario de datos:

- PassengerID - Una columna añadida por Kaggle para identificar cada fila y facilitar los envíos
- Survived - Si el pasajero sobrevivió o no y el valor que estamos prediciendo (0=No, 1=Sí)
- Pclass - La clase del billete que compró el pasajero (1=1ª, 2=2ª, 3=3ª)
- Sex - El sexo del pasajero
- Edad - La edad del pasajero en años
- SibSp - El número de hermanos o cónyuges que el pasajero tenía a bordo del Titanic
- Parch - El número de padres o hijos que el pasajero tenía a bordo del Titanic
- Ticket - El número de billete del pasajero
- Fare - La tarifa que pagó el pasajero
- Cabin - El número de cabina del pasajero
- Embarked - El puerto donde embarcó el pasajero (C=Cherbourg, Q=Queenstown, S=Southampton)

III.1 Cómo Empezar a Usar Kaggle

La página de datos en Kaggle tiene algunas notas adicionales sobre algunas de las columnas. Siempre vale la pena explorar esto en detalle para obtener una comprensión completa de los datos.

Las primeras 2 filas de los datos están abajo:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85

III.1 Cómo Empezar a Usar Kaggle

El tipo de aprendizaje automático que vamos a realizar se llama **clasificación**, porque cuando hacemos predicciones estamos clasificando a cada pasajero como superviviente o no. Más concretamente, estamos realizando una **clasificación binaria**, lo que significa que sólo hay dos estados diferentes que estamos clasificando.

En cualquier ejercicio de aprendizaje automático, es muy importante pensar en el tema que estamos prediciendo. A este paso lo llamamos adquirir conocimiento del dominio, y es uno de los determinantes más importantes para el éxito en el aprendizaje automático.

En este caso, es importante comprender la catástrofe del **Titanic** y, concretamente, qué variables podrían afectar al resultado de la supervivencia. Cualquiera que haya visto la película Titanic recordaría que las mujeres y los niños tenían preferencia en los botes salvavidas (al igual que en la vida real). También recordaría la gran disparidad de clases de los pasajeros.

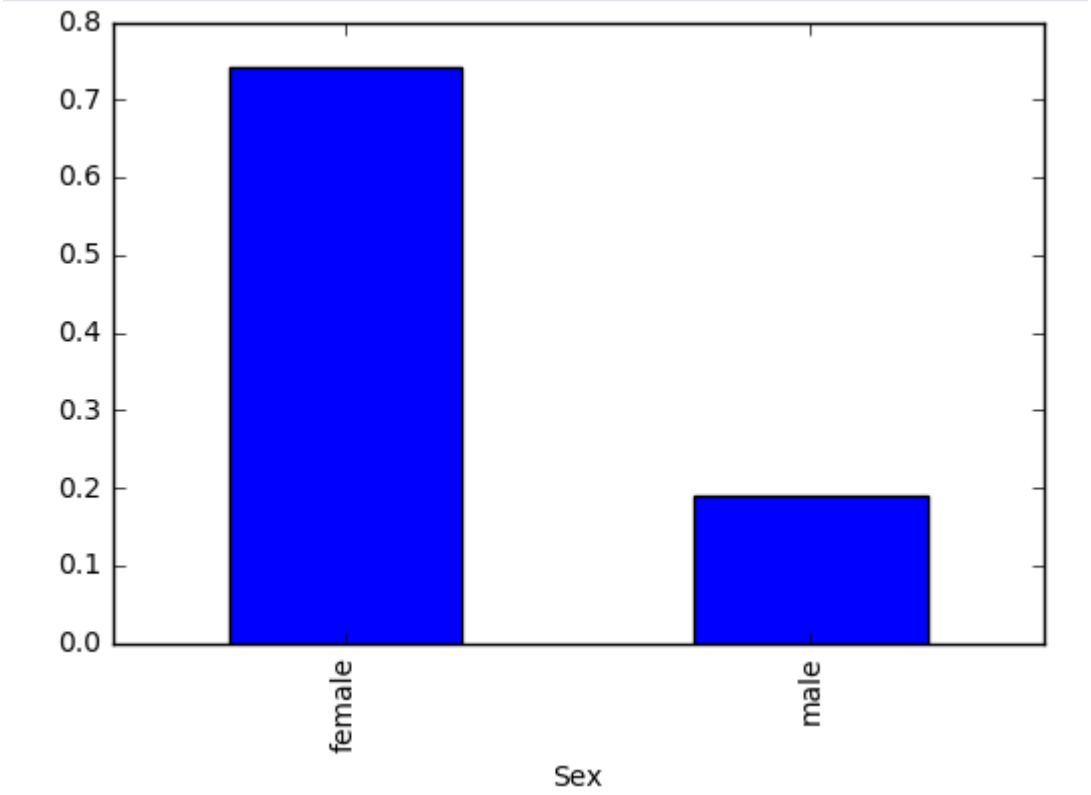
III.1 Cómo Empezar a Usar Kaggle

Esto indica que la Age (edad), el Sex (sexo) y la PClass pueden ser buenos predictores de la supervivencia. Empezaremos por explorar el Sex y la PClass visualizando los datos.

Como la columna Survived (Sobrevivientes) contiene 0 si el pasajero no sobrevivió y 1 si lo hizo, podemos segmentar nuestros datos por sexo y calcular la media de esta columna. Podemos utilizar DataFrame.pivot_table() para hacerlo fácilmente:

```
sex_pivot = train.pivot_table(index="Sex",values="Survived")
sex_pivot.plot.bar()
plt.show()
```

El gráfico resultante tendrá el siguiente aspecto:



III.1 Cómo Empezar a Usar Kaggle

Podemos ver inmediatamente que las mujeres sobrevivieron en una proporción mucho mayor que los hombres. Hagamos lo mismo con la columna Pclass.

Instrucciones

- Utilice DataFrame.pivot_table() para pivotar el marco de datos de train:
  - Utilice "Pclass" para el parámetro de índice
  - Utilice "Survived" para el parámetro de valores
  - Utilice DataFrame.plot.bar() para trazar la tabla pivotante

```
import matplotlib.pyplot as plt

sex_pivot = train.pivot_table(index="Sex",values="Survived")
sex_pivot.plot.bar()
plt.show()
class_pivot = train.pivot_table(index="Pclass",values="Survived")
class_pivot.plot.bar()
plt.show()
```


III.1 Cómo Empezar a Usar Kaggle

III.1.3 Exploración y conversión de la columna de edad

Las columnas Sex y PClass son lo que llamamos características categóricas. Esto significa que los valores representan algunas opciones distintas (por ejemplo, si el pasajero es hombre o mujer).

Echemos un vistazo a la columna Age utilizando Series.describe().

```
print(train["Age"].describe())
```

```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```


III.1 Cómo Empezar a Usar Kaggle

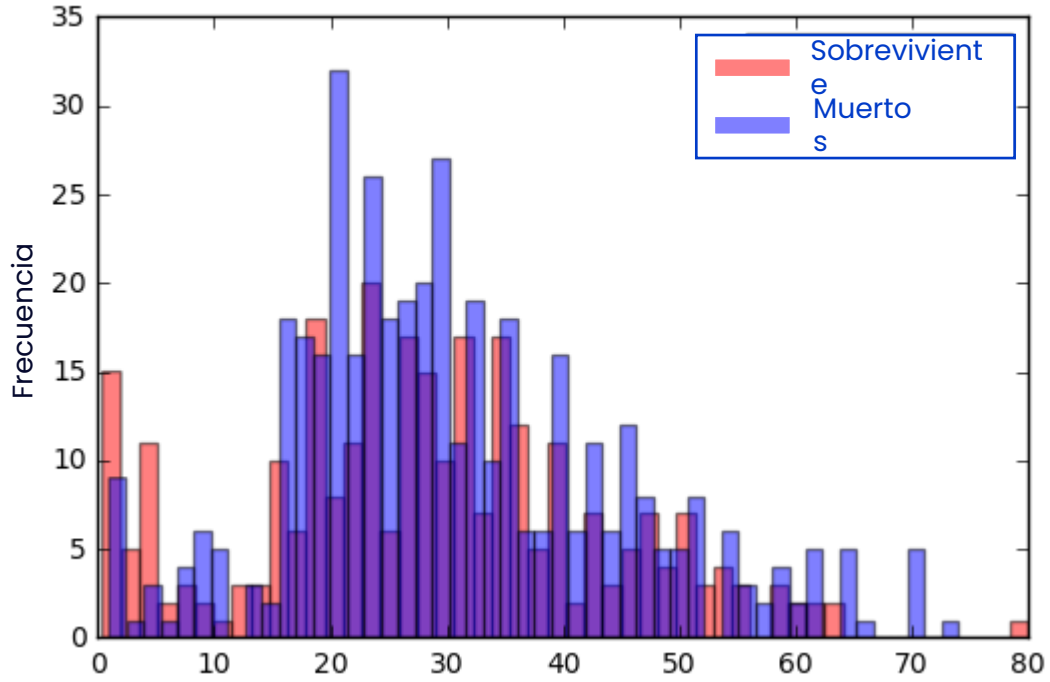
La columna Age contiene números que van de 0,42 a 80,0 (si miras la página de datos de Kaggle, nos informa de que Age es fraccionaria si el pasajero es menor de uno). La otra cosa a tener en cuenta aquí es que hay 714 valores en esta columna, menos que las 891 filas que descubrimos que tenía el conjunto de datos de train anteriormente en esta misión, lo que indica que tenemos algunos valores perdidos.

Todo esto significa que la columna Age debe ser tratada de forma ligeramente diferente, ya que se trata de una columna numérica continua. Una forma de ver la distribución de los valores en un conjunto numérico continuo es utilizar histogramas. Podemos crear dos histogramas para comparar visualmente los que sobrevivieron frente a los que murieron en diferentes rangos de edad:

```
survived = train[train["Survived"] == 1]
died = train[train["Survived"] == 0]
survived["Age"].plot.hist(alpha=0.5,color='red',bins=50)
died["Age"].plot.hist(alpha=0.5,color='blue',bins=50)
plt.legend(['Survived','Died'])
plt.show()
```

III.1 Cómo Empezar a Usar Kaggle

El gráfico resultante tendrá el siguiente aspecto:



La relación aquí no es sencilla, pero podemos ver que en algunos rangos de edad sobrevivieron más pasajeros, donde las barras rojas son más altas que las azules.

Para que esto sea útil para nuestro modelo de aprendizaje automático, podemos separar esta característica continua en una característica categórica dividiéndola en rangos. Podemos utilizar la pandas.cut() function para ayudarnos.

III.1 Cómo Empezar a Usar Kaggle

La función pandas.cut() tiene dos parámetros requeridos - la columna que deseamos cortar, y una lista de números que definen los límites de nuestros cortes. También vamos a utilizar el parámetro opcional labels, que toma una lista de etiquetas para los recipientes resultantes. Esto nos facilitará la comprensión de nuestros resultados.

Antes de modificar esta columna, debemos tener en cuenta dos cosas. En primer lugar, cualquier cambio que hagamos en los datos de train, debemos aplicarlo también a los datos de prueba, de lo contrario no podremos utilizar nuestro modelo para hacer predicciones para nuestros envíos. En segundo lugar, tenemos que acordarnos de gestionar los valores perdidos que hemos observado anteriormente.

En el ejemplo siguiente, creamos una función que:

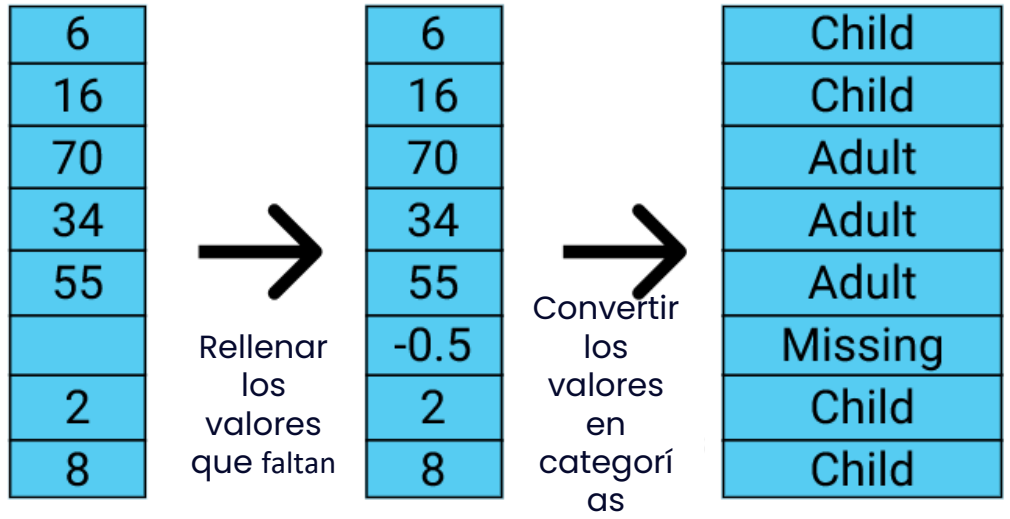
- Utiliza el pandas.fillna() method para rellenar todos los valores perdidos con -0.5
- Corta la columna Age en tres segmentos: Missing, Child y Adult usando pandas.cut()

III.1 Cómo Empezar a Usar Kaggle

A continuación, utilizamos esa función en los train and test dataframes.

```
def process_age(df,cut_points,label_names):
    df["Age"] = df["Age"].fillna(-0.5)
    df["Age_categories"] = pd.cut(df["Age"],cut_points,labels=label_names)
    return df

cut_points = [-1,0,18,100]
label_names = ["Missing","Child","Adult"]

train = process_age(train,cut_points,label_names)
test = process_age(test,cut_points,label_names)
```

El siguiente diagrama muestra cómo la función convierte los datos:



Tenga en cuenta que la lista "cut_points" tiene un elemento más que la lista "labels_names", ya que necesita definir el límite superior del último segmento.

III.1 Cómo Empezar a Usar Kaggle

Instrucciones

Cree las listas cut_points y label_names para dividir la columna Age en seis categorías:

- Missing, de -1 a 0
- Infant, de 0 a 5
- Child, de 5 a 12
- Teenager, de 12 a 18
- Young Adult, de 18 a 35
- Adult, de 35 a 60
- Senior, de 60 a 100
- Aplicar la función process_age() en el marco de datos(dataframe) train asignando el resultado a train
- Aplicar la función process_age() en el marco de datos(dataframe) test , asignando el resultado a test
- Utilize DataFrame.pivot_table() para hacer pivotar el marco de datos de train por la columna Age_categories
- Use DataFrame.plot.bar() para trazar la tabla pivotante

III.1 Cómo Empezar a Usar Kaggle

Soluciones

```
def process_age(df,cut_points,label_names):
    df["Age"] = df["Age"].fillna(-0.5)
    df["Age_categories"] = pd.cut(df["Age"],cut_points,labels=label_names)
    return df
cut_points = [-1,0,5,12,18,35,60,100]
label_names = ["Missing","Infant","Child","Teenager","Young Adult","Adult","Senior"]

train = process_age(train,cut_points,label_names)
test = process_age(test,cut_points,label_names)

pivot = train.pivot_table(index="Age_categories",values='Survived')
pivot.plot.bar()
plt.show()
```

III.1 Cómo Empezar a Usar Kaggle

III.1.4 Preparación de los datos para el aprendizaje automático

Hasta ahora hemos identificado tres columnas que pueden ser útiles para predecir la supervivencia :

- Sex
- Pclass
- Age, o más concretamente nuestro recién creado Age_categories

Antes de construir nuestro modelo, tenemos que preparar estas columnas para el aprendizaje automático. La mayoría de los algoritmos de aprendizaje automático no pueden entender las etiquetas de texto, así que tenemos que convertir nuestros valores en números.

Además, tenemos que tener cuidado de no implicar ninguna relación numérica donde no la hay. Si pensamos en los valores de la columna Pclass, sabemos que son 1, 2 y 3. Puede confirmarlo ejecutando el siguiente código:

```
train["Pclass"].value_counts()
```


III.1 Cómo Empezar a Usar Kaggle

Aunque la clase de cada pasajero tiene ciertamente algún tipo de relación ordenada, la relación entre cada clase no es la misma que la relación entre los números 1, 2 y 3. Por ejemplo, la clase 2 no "vale" el doble que la clase 1, y la clase 3 no "vale" el triple que la clase 1.

Para eliminar esta relación, podemos crear columnas ficticias para cada valor único en Pclass.

Pclass
3
1
3
1
3
3
1
3
3
2

Pclass_1	Pclass_2	Pclass_3
0	0	1
1	0	0
0	0	1
1	0	0
0	0	1
0	0	1
1	0	0
0	0	1
0	0	1
0	1	0

III.1 Cómo Empezar a Usar Kaggle

En lugar de hacerlo manualmente, podemos utilizar la pandas.get_dummies() function que generará las columnas que se muestran en el diagrama anterior.

El siguiente código crea una función para crear las columnas ficticias para la columna Pclass y las añade al marco de datos original. A continuación, aplica esa función a los marcos de datos de train y de test.

```
def create_dummies(df,column_name):
    dummies = pd.get_dummies(df[column_name],prefix=column_name)
    df = pd.concat([df,dummies],axis=1)
    return df

train = create_dummies(train,"Pclass")
test = create_dummies(test,"Pclass")
```

Utilicemos esa función para crear columnas ficticias tanto para el Sex como para el Age_categories columns.

III.1 Cómo Empezar a Usar Kaggle

Instrucciones

- Utilice la función create_dummies() para crear variables ficticias para la columna Sex:
  - En el marco de datos de train
  - En el marco de datos de test
- Utilice la función create_dummies() para crear variables ficticias para la columna Age_categories:
  - En el marco de datos de train
  - En el marco de datos de test

Soluciones

```
def create_dummies(df,column_name):
    dummies = pd.get_dummies(df[column_name],prefix=column_name)
    df = pd.concat([df,dummies],axis=1)
    return df

train = create_dummies(train,"Pclass")
test = create_dummies(test,"Pclass")
train = create_dummies(train,"Sex")
test = create_dummies(test,"Sex")

train = create_dummies(train,"Age_categories")
test = create_dummies(test,"Age_categories")
```

III.1 Cómo Empezar a Usar Kaggle

III.1.5 Su primer aprendizaje automático

Ahora que nuestros datos han sido preparados, estamos listos para entrenar nuestro primer modelo. El primer modelo que utilizaremos se llama **Regresión Logística**, que suele ser el primer modelo que se entrena al realizar la clasificación.

Utilizaremos la biblioteca **scikit-learn**, ya que tiene muchas herramientas que facilitan el aprendizaje automático. El flujo de trabajo de scikit-learn consiste en cuatro pasos principales:

- Instanciar (o crear) el modelo de aprendizaje automático específico que desea utilizar
- Ajustar el modelo a los datos de entrenamiento
- Utilizar el modelo para hacer predicciones
- Evaluar la precisión de las predicciones

III.1 Cómo Empezar a Usar Kaggle

Cada modelo en scikit-learn se implementa como una clase separada y el primer paso es identificar la clase de la que queremos crear una instancia. En nuestro caso, queremos utilizar la **LogisticRegression class.**

Empezaremos por ver los dos primeros pasos. En primer lugar, tenemos que importar la clase:

```
from sklearn.linear_model import LogisticRegression
```

A continuación, creamos un objeto **LogisticRegression** :

```
lr = LogisticRegression()
```

III.1 Cómo Empezar a Usar Kaggle

Por último, utilizamos el LogisticRegression.fit() method para entrenar nuestro modelo. El método .fit() acepta dos argumentos: X e Y. X debe ser una matriz bidimensional (como un marco de datos) de las características con las que deseamos entrenar nuestro modelo, e Y debe ser una matriz unidimensional (como una serie) de nuestro objetivo, o la columna que deseamos predecir.

```
columns = ['Pclass_2', 'Pclass_3', 'Sex_male']
lr.fit(train[columns], train['Survived'])
```

El código anterior ajusta (o entrena) nuestro modelo LogisticRegression utilizando tres columnas: Pclass_2, Pclass_3 y Sex_male.

Vamos a entrenar nuestro modelo utilizando todas las columnas que hemos creado en la pantalla anterior.

III.1 Cómo Empezar a Usar Kaggle

Instrucciones

- Instanciar un objeto LogisticRegression llamado lr
- Utilice LogisticRegression.fit() para ajustar el modelo en el conjunto de datos de train utilizando:
  - Las columnas contenidas en columns como primer parámetro (X)
  - La columna Survived como el segundo parámetro (Y)

Soluciones

```
columns = ['Pclass_1', 'Pclass_2', 'Pclass_3', 'Sex_female', 'Sex_male',
       'Age_categories_Missing','Age_categories_Infant',
       'Age_categories_Child', 'Age_categories_Teenager',
       'Age_categories_Young Adult', 'Age_categories_Adult',
       'Age_categories_Senior']

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train[columns], train["Survived"])
```

III.1.6 Datos divididos

Felicitaciones, ¡has entrenado tu primer modelo de aprendizaje automático! Nuestro siguiente paso es averiguar la precisión de nuestro modelo, y para ello tendremos que hacer algunas predicciones.

Si lo recuerdas, tenemos un marco de datos de prueba (test) que podríamos utilizar para hacer predicciones. Podríamos hacer predicciones en ese conjunto de datos, pero como no tiene la columna Survived tendríamos que enviarlo a Kaggle para averiguar nuestra precisión. Esto se convertiría rápidamente en una molestia si tuviéramos que enviarlo para averiguar la precisión cada vez que optimizáramos nuestro modelo.

También podríamos ajustar y predecir en nuestro marco de datos de entrenamiento (train), sin embargo, si hacemos esto hay una alta probabilidad de que nuestro modelo se sobreajuste, lo que significa que se desempeñará bien porque estamos probando en los mismos datos que hemos entrenado, pero luego se desempeñan mucho peor en los nuevos datos no vistos.

III.1 Cómo Empezar a Usar Kaggle

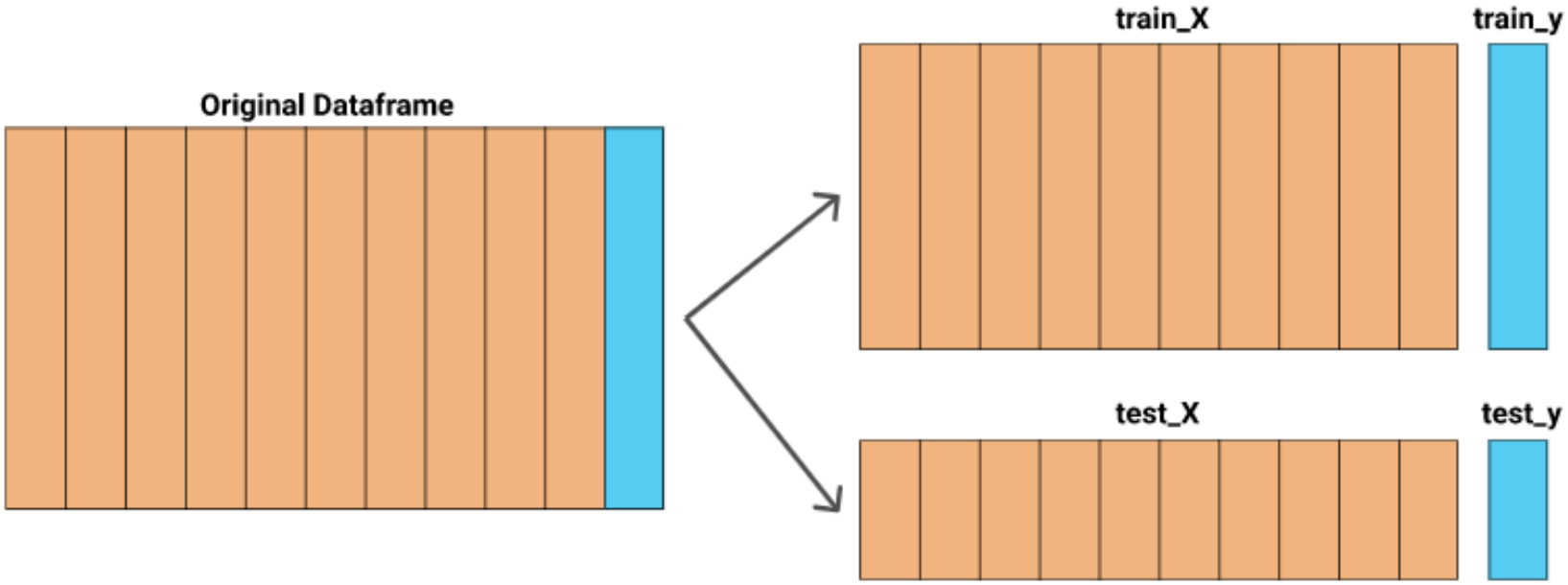
En cambio, podemos dividir nuestro marco de datos de entrenamiento (train) en dos:

- Una parte para entrenar nuestro modelo (a menudo el 80% de las observaciones)
- Una parte para hacer predicciones y probar nuestro modelo (a menudo el 20% de las observaciones)

La convención en el aprendizaje automático es llamar a estas dos partes entrenamiento (train) y prueba (test). Esto puede llegar a ser confuso, ya que ya tenemos nuestro marco de datos de prueba (test) que eventualmente vamos a utilizar para hacer predicciones para presentar a Kaggle. Para evitar confusiones, a partir de ahora, vamos a llamar a estos datos de "prueba" de Kaggle datos retenidos, que es el nombre técnico dado a este tipo de datos utilizados para las predicciones finales.

III.1 Cómo Empezar a Usar Kaggle

La biblioteca scikit-learn tiene una práctica función model_selection.train_test_split() function que podemos utilizar para dividir nuestros datos. train_test_split() acepta dos parámetros, X e Y, que contienen todos los datos que queremos entrenar y probar, y devuelve cuatro objetos: train X, train Y, test X, test Y:



III.1 Cómo Empezar a Usar Kaggle

Esta es la sintaxis para crear estos cuatro objetos:

```
from sklearn.model_selection import train_test_split

columns = ['Pclass_2', 'Pclass_3', 'Sex_male']

all_X = train[columns]
all_y = train['Survived']

train_X, test_X, train_y, test_y = train_test_split(
    all_X, all_y, test_size=0.2,random_state=0)
```

Verás que hay otros dos parámetros que utilizamos: test_size, que nos permite controlar en qué proporciones se dividen nuestros datos, y random_state. La función train_test_split() aleatoriza las observaciones antes de dividirlas, y establecer una semilla aleatoria (**random seed**) significa que nuestros resultados serán reproducibles, lo cual es importante si estás colaborando, o necesitas producir resultados consistentes cada vez (lo cual requiere nuestro verificador de respuestas).

III.1 Cómo Empezar a Usar Kaggle

Instrucciones

- Utilice la función model_selection.train_test_split() para dividir el marco de datos de entrenamiento (train) utilizando los siguientes parámetros:
  - test_size de 0.2
  - random_state de 0
  - Asigne los cuatro objetos devueltos a train_X, test_X, train_y y test_y

III.1 Cómo Empezar a Usar Kaggle

Soluciones

```
holdout = test # from now on we will refer to this
               # dataframe as the holdout data

from sklearn.model_selection import train_test_split

columns = ['Pclass_1', 'Pclass_2', 'Pclass_3', 'Sex_female', 'Sex_male',
       'Age_categories_Missing','Age_categories_Infant',
       'Age_categories_Child', 'Age_categories_Teenager',
       'Age_categories_Young Adult', 'Age_categories_Adult',
       'Age_categories_Senior']
all_X = train[columns]
all_y = train['Survived']

train_X, test_X, train_y, test_y = train_test_split(
    all_X, all_y, test_size=0.20,random_state=0)
```


III.1 Cómo Empezar a Usar Kaggle

III.1.7 Hacer predicciones y medir su exactitud

Ahora que tenemos los datos divididos en conjuntos de entrenamiento y de prueba, podemos ajustar nuestro modelo de nuevo en nuestro conjunto de entrenamiento y, a continuación, utilizar ese modelo para hacer predicciones en nuestro conjunto de prueba.

Una vez que hemos ajustado nuestro modelo, podemos utilizar el método LogisticRegression.predict() method para hacer predicciones.

El método predict() toma un único parámetro X, una matriz bidimensional de características para las observaciones que deseamos predecir. X debe tener exactamente las mismas características que la matriz que hemos utilizado para ajustar nuestro modelo. El método devuelve una matriz unidimensional de predicciones.

```
lr = LogisticRegression()
lr.fit(train_X, train_y)
predictions = lr.predict(test_X)
```

III.1 Cómo Empezar a Usar Kaggle

Hay varias formas de medir la precisión de los modelos de aprendizaje automático, pero cuando se compite en las competiciones de Kaggle, hay que asegurarse de utilizar el mismo método que Kaggle utiliza para calcular la precisión para esa competición específica.

En este caso, la **sección de evaluación para la competición Titanic en Kaggle** nos dice que nuestra puntuación calculada como "el porcentaje de pasajeros predichos correctamente". Esta es, con mucho, la forma más común de precisión para la clasificación binaria.

Como ejemplo, imaginemos que estamos prediciendo un pequeño conjunto de datos de cinco observaciones.

Our model's prediction	The actual value	Correct
0	0	Yes
1	0	No
0	1	No
1	1	Yes
1	1	Yes

III.1 Cómo Empezar a Usar Kaggle

En este caso, nuestro modelo predijo correctamente tres de los cinco valores, por lo que la precisión basada en este conjunto de predicciones sería del 60%.

De nuevo, scikit-learn tiene una función muy útil que podemos utilizar para calcular la precisión: metrics.accuracy_score(). La función acepta dos parámetros, y_true e y_pred, que son los valores reales y nuestros valores predichos respectivamente, y devuelve nuestra puntuación de precisión.

Instrucciones

- Instanciar un nuevo objeto LogisticRegression(), lr
- Ajuste el modelo utilizando train_X y train_y
- Realice predicciones con test_X y asigne los resultados a las predicciones (predictions)
- Utilice accuracy_score() para comparar test_y y predicciones, asignando el resultado a accuracy
- Imprimir la variable de accuracy

III.1 Cómo Empezar a Usar Kaggle

Soluciones

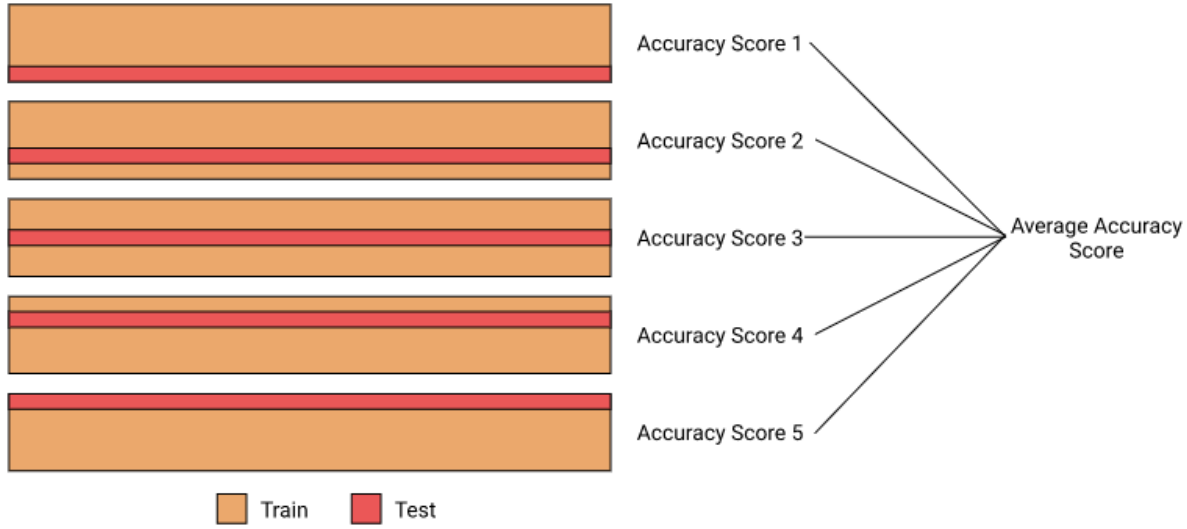
```
from sklearn.metrics import accuracy_score
lr = LogisticRegression()
lr.fit(train_X, train_y)
predictions = lr.predict(test_X)
accuracy = accuracy_score(test_y, predictions)
print(accuracy)
```


III.1 Cómo Empezar a Usar Kaggle

III.1.8 Uso de la validación cruzada para una medición más precisa de los errores

Nuestro modelo tiene una puntuación de precisión del 81,0% cuando se prueba con nuestro conjunto de pruebas del 20%. Dado que este conjunto de datos es bastante pequeño, hay muchas posibilidades de que nuestro modelo esté sobreajustado y no funcione tan bien con datos totalmente desconocidos.

Para comprender mejor el rendimiento real de nuestro modelo, podemos utilizar una técnica llamada **validación cruzada** para entrenar y probar nuestro modelo en diferentes partes de nuestros datos, y luego promediar las puntuaciones de precisión.



III.1 Cómo Empezar a Usar Kaggle

La forma más común de validación cruzada, y la que utilizaremos, se llama validación cruzada de k Fold (k-pliegues). El término "pliegue" se refiere a cada iteración diferente en la que entrenamos nuestro modelo, y "k" sólo se refiere al número de pliegues. En el diagrama anterior, hemos ilustrado la validación de k pliegues donde k es 5.

Utilizaremos la model_selection.cross_val_score() function de scikit-learn para automatizar el proceso. La sintaxis básica de cross_val_score() es:

```
cross_val_score(estimator, X, y, cv=None)
```

- estimator es un objeto estimador de scikit-learn, como los objetos LogisticRegression() que hemos estado creando
- X son todas las características de nuestro conjunto de datos
- y es la variable objetivo
- cv especifica el número de pliegues

III.1 Cómo Empezar a Usar Kaggle

La función devuelve un ndarray numpy de las puntuaciones de precisión de cada pliegue.

Cabe destacar que la función cross_val_score() puede utilizar una variedad de técnicas de validación cruzada y tipos de puntuación, pero por defecto utiliza la validación k-fold y las puntuaciones de precisión para nuestros tipos de entrada.

Instrucciones

- Instanciar un nuevo objeto LogisticRegression(), lr
- Utilice model_selection.cross_val_score() para realizar la validación cruzada de nuestros datos y asignar los resultados a las puntuaciones (scores):
    - Utilizar el lr recién creado como estimador
    - Utilizar all_X y all_y como datos de entrada
    - Especificar 10 pliegues (folds) a utilizar
- Utilizar la función numpy.mean() function para calcular la media de las puntuaciones (scores) y asignar el resultado a la precisión (accuracy)
- Imprime las variables puntuaciones(scores) y precisión (accuracy)

III.1 Cómo Empezar a Usar Kaggle

Soluciones

```
from sklearn.model_selection import cross_val_score
import numpy as np
lr = LogisticRegression()
scores = cross_val_score(lr, all_X, all_y, cv=10)
accuracy = np.mean(scores)
print(scores)
print(accuracy)
```


III.1 Cómo Empezar a Usar Kaggle

III.1.9 Hacer predicciones con datos no vistos

A partir de los resultados de nuestra validación de k pliegues, se puede ver que el número de precisión varía con cada pliegue, oscilando entre el 76,4% y el 87,6%. Esto demuestra la importancia de la validación cruzada.

En realidad, nuestra puntuación media de precisión fue del 80,2%, lo que no está lejos del 81,0% que obtuvimos de nuestra simple división de entrenamiento/prueba, sin embargo, este no siempre será el caso, y siempre debe utilizar la validación cruzada para asegurarse de que las métricas de error que está obteniendo de su modelo son precisas.

Ahora estamos listos para utilizar el modelo que hemos construido para entrenar nuestro modelo final y luego hacer predicciones en nuestros datos no vistos, o lo que Kaggle llama el conjunto de datos de “prueba”.

III.1 Cómo Empezar a Usar Kaggle

Instrucciones

- Instanciar un nuevo objeto LogisticRegression(), lr
- Utilice el método fit() para entrenar el modelo lr utilizando todos los datos de entrenamiento de Kaggle: all_X y all_y
- Haga predicciones utilizando los datos retenidos (holdout) y asigne el resultado a holdout_predictions

Soluciones

```
columns = ['Pclass_1', 'Pclass_2', 'Pclass_3', 'Sex_female', 'Sex_male',
       'Age_categories_Missing','Age_categories_Infant',
       'Age_categories_Child', 'Age_categories_Teenager',
       'Age_categories_Young Adult', 'Age_categories_Adult',
       'Age_categories_Senior']
lr = LogisticRegression()
lr.fit(all_X,all_y)
holdout_predictions = lr.predict(holdout[columns])
```


III.1.10 Fichero de presentación de la creación

Lo último que tenemos que hacer es crear un archivo de presentación. Cada competición de Kaggle puede tener requisitos ligeramente diferentes para el archivo de presentación. Esto es lo que se especifica en la **página de evaluación del concurso Titanic**.

Debes enviar un archivo csv con exactamente 418 entradas más una fila de cabecera. Su presentación mostrará un error si tiene columnas adicionales (más allá de PassengerId y Survived) o filas.

El archivo debe tener exactamente 2 columnas:

- *PassengerId (clasificado en cualquier orden)*
- *Survived (contiene sus predicciones binarias: 1 para sobrevivido, 0 para fallecido)*

III.1 Cómo Empezar a Usar Kaggle

La siguiente tabla muestra esto en un formato un poco más fácil de entender, para que podamos visualizar lo que pretendemos.

PassengerId	Survived
892	0
893	1
894	0

Tendremos que crear un nuevo marco de datos que contenga el holdout_predictions que creamos en la pantalla anterior y la columna PassengerId del marco de datos holdout. No tenemos que preocuparnos de hacer coincidir los datos, ya que ambos permanecen en su orden original.

Para ello, podemos pasar un diccionario a la pandas.DataFrame() function:

```
holdout_ids = holdout["PassengerId"]
submission_df = {"PassengerId": holdout_ids,
                 "Survived": holdout_predictions}
submission = pd.DataFrame(submission_df)
```

III.1 Cómo Empezar a Usar Kaggle

Finalmente, utilizaremos el pandas.DataFrame() function para guardar el dataframe en un archivo CSV. Tenemos que asegurarnos de que el parámetro de index(índice) se establece en False, de lo contrario vamos a añadir una columna extra a nuestro CSV.

Instrucciones

- Cree una submission (presentación) de marco de datos que coincida con la especificación de Kaggle
- Utilice el método to_csv() para guardar el marco de datos de submission (presentación) utilizando el nombre de archivo submission.csv, utilizando **la documentación** para buscar la sintaxis correcta

Soluciones

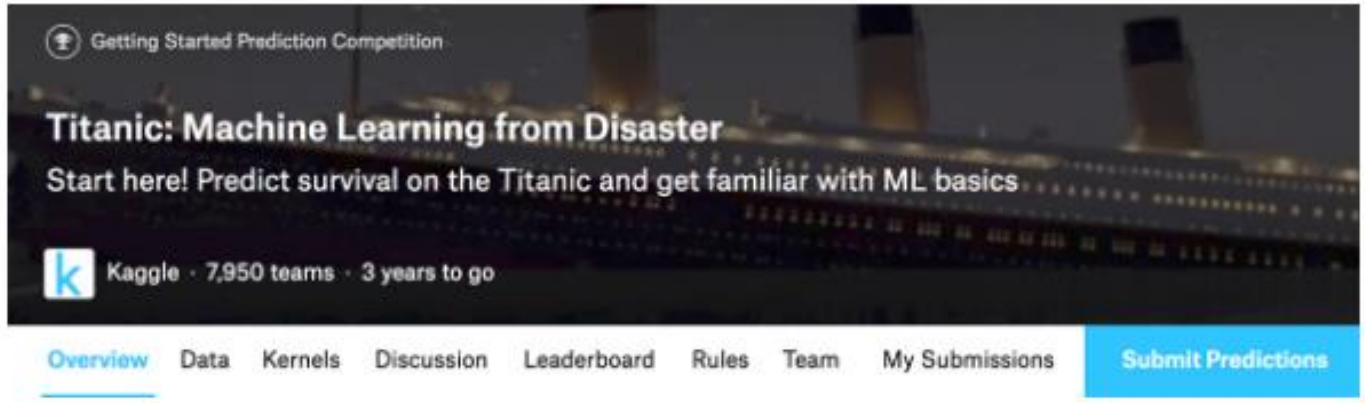
```
holdout_ids = holdout["PassengerId"]
submission_df = {"PassengerId": holdout_ids,
                 "Survived": holdout_predictions}
submission = pd.DataFrame(submission_df)

submission.to_csv("submission.csv",index=False)
```

III.1 Cómo Empezar a Usar Kaggle

III.1.11 Cómo hacer nuestro primer envío a Kaggle

Puedes descargar el archivo de envío que acabas de crear (si trabajas localmente, estará en el mismo directorio que tu cuaderno).

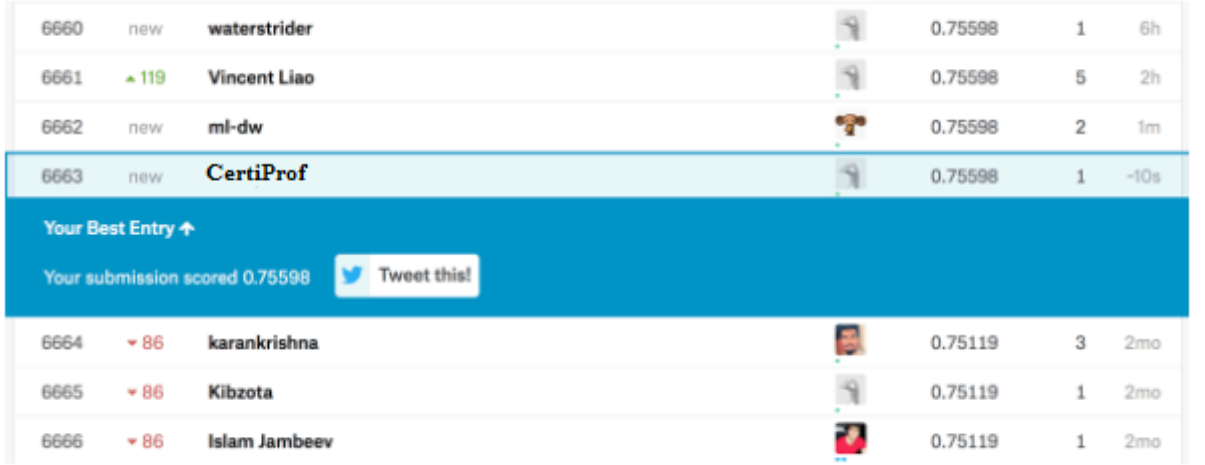
Ahora que tenemos nuestro archivo de envío, podemos empezar nuestro envío a Kaggle haciendo clic en el botón azul "Submit Predictions" en la **página del concurso .**



III.1 Cómo Empezar a Usar Kaggle

A continuación, se le pedirá que cargue su archivo CSV y que añada una breve descripción de su presentación. Cuando hagas tu envío, Kaggle procesará tus predicciones y te dará tu precisión para los datos retenidos y tu clasificación.

Cuando termine de procesar, verás que nuestro primer envío obtiene una puntuación de precisión de 0,75598, es decir, un 75,6%.



III.1 Cómo Empezar a Usar Kaggle

El hecho de que nuestra precisión en los datos retenidos sea del 75,6%, en comparación con la precisión del 80,2% que obtuvimos con la validación cruzada, indica que nuestro modelo se está sobreajustando ligeramente a nuestros datos de entrenamiento.

En el momento de escribir este artículo, la precisión del 75,6% da un rango de 6.663 de 7.954. Es fácil mirar las tablas de clasificación de Kaggle después de su primera presentación y desanimarse, pero tenga en cuenta que esto es sólo un punto de partida.

También es muy común ver un pequeño número de puntuaciones del 100% en la parte superior de la tabla de clasificación del Titanic y pensar que tienes un largo camino por recorrer. En realidad, cualquiera que obtenga una puntuación de alrededor del 90% en esta competición probablemente esté haciendo trampas (es fácil buscar los nombres de los pasajeros en el conjunto de espera en Internet y ver si sobrevivieron).

III.1 Cómo Empezar a Usar Kaggle

Hay un gran análisis en Kaggle, **How am I doing with my score**, que utiliza algunas estrategias diferentes y sugiere que la puntuación mínima para esta competición es del 62,7% (que se consigue suponiendo que todos los pasajeros murieron) y una máxima de alrededor del 82%. Estamos a poco más de la mitad del camino entre el mínimo y el máximo, lo cual es un gran punto de partida.

Hay muchas cosas que podemos hacer para mejorar la precisión de nuestro modelo. Éstas son algunas de las que cubriremos en las dos próximas misiones de este curso:

- Mejorar las características:
  - Ingeniería de rasgos: Crear nuevas características a partir de los datos existentes
  - Selección de características: Seleccionar las características más relevantes para reducir el ruido y el sobreajuste
- Mejora del modelo:
  - Selección de modelos: Probar una variedad de modelos para mejorar el rendimiento
  - Optimización de hiperparámetros: Optimizar los ajustes dentro de cada modelo de aprendizaje automático en particular

III.2 Preparación, Selección e Ingeniería de las Características

En la última misión, hicimos nuestro primer envío a Kaggle, obteniendo una puntuación de precisión del 75,6%. Aunque este es un buen comienzo, sin duda hay margen de mejora. Hay dos áreas principales en las que podemos centrarnos para aumentar la precisión de nuestras predicciones:

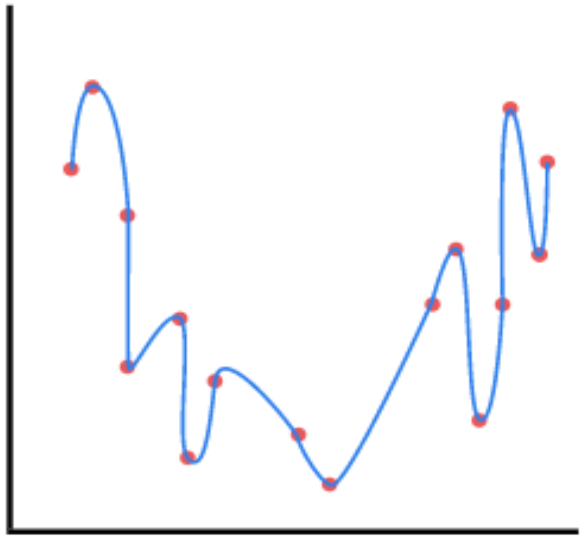
- Mejorar las características con las que entrenamos nuestro modelo
- Mejorar el propio modelo

En esta misión, vamos a centrarnos en trabajar con las características utilizadas en nuestro modelo.

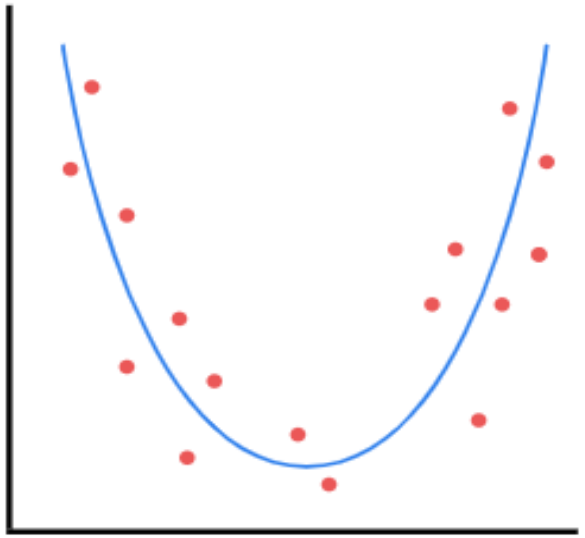
Empezaremos por analizar la **selección de características**. La selección de características es importante porque ayuda a excluir características que no son buenos predictores, o características que están estrechamente relacionadas entre sí. Ambas cosas harán que nuestro modelo sea menos preciso, sobre todo en datos no vistos anteriormente.

El siguiente diagrama lo ilustra. Los puntos rojos representan los datos que intentamos predecir, y cada una de las líneas azules representa un modelo diferente.

III.2 Preparación, Selección e Ingeniería de las Características



Un modelo sobreajustado



Un modelo bien ajustado

III.2 Preparación, Selección e Ingeniería de las Características

El modelo de la izquierda está sobreajustado, lo que significa que el modelo representa los datos de entrenamiento con demasiada exactitud, y es poco probable que prediga bien en datos no vistos, como los datos retenidos para nuestra competición Kaggle.

El modelo de la derecha está bien ajustado. Captura el patrón subyacente en los datos sin el ruido detallado que se encuentra sólo en el conjunto de entrenamiento. Un modelo bien ajustado es probable que haga predicciones precisas sobre datos no vistos anteriormente. La clave para crear un modelo bien ajustado es seleccionar el equilibrio adecuado de características y crear nuevas características para entrenar el modelo.

En la misión anterior, entrenamos nuestro modelo utilizando datos sobre la edad, el sexo y la clase de los pasajeros del Titanic. Empecemos por utilizar las funciones que creamos en esa misión para añadir las columnas que teníamos al final de la primera misión.

Recuerda que cualquier modificación que hagamos en nuestros datos de entrenamiento (train.csv) también tenemos que hacerla en nuestros datos de retención (test.csv).

III.2 Preparación, Selección e Ingeniería de las Características

Instrucciones

- Utilice la función process_age():
  - Para convertir la columna Age en train, asignando el resultado a train
  - Para convertir la columna Age en holdout, asignando el resultado a holdout
- Crear un bucle for que itere sobre los nombres de columna "Age_categories", "Pclass" y "Sex" en cada iteración
  - Utiliza la función create_dummies() para procesar el marco de datos de train para la columna dada, asignando el resultado a train
  - Utilice la función create_dummies() para procesar el marco de datos holdout para la columna dada, asignando el resultado a holdout
- Utilice la función print() para mostrar las columnas de train utilizando train.columns

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
import pandas as pd

train = pd.read_csv('train.csv')
holdout = pd.read_csv('test.csv')

def process_age(df):
    df["Age"] = df["Age"].fillna(-0.5)
    cut_points = [-1,0,5,12,18,35,60,100]
    label_names =
["Missing","Infant","Child","Teenager","Young
Adult","Adult","Senior"]
    df["Age_categories"] =
pd.cut(df["Age"],cut_points,labels=label_names)
    return df

```


III.2 Preparación, Selección e Ingeniería de las Características

III.2.1 Preparar más características

Nuestro modelo en la misión anterior se basaba en tres columnas de los datos originales: Age, Sex y PClass. Como ha visto al imprimir los nombres de las columnas en la pantalla anterior, hay otras columnas que aún no hemos utilizado. Para facilitar la referencia, la salida de la pantalla anterior se copia a continuación:

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name',
       'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
       'Fare', 'Cabin', 'Embarked', 'Age_categories',
       'Age_categories_Missing',
       'Age_categories_Infant',
       'Age_categories_Child',
       'Age_categories_Teenager',
       'Age_categories_Young Adult',
       'Age_categories_Adult',
       'Age_categories_Senior', 'Pclass_1',
       'Pclass_2', 'Pclass_3','Sex_female',
       'Sex_male'], dtype='object')
```

III.2 Preparación, Selección e Ingeniería de las Características

Las últimas nueve filas de la salida son columnas ficticias que hemos creado, pero en las tres primeras filas podemos ver que hay una serie de características que aún no hemos utilizado. Podemos ignorar PassengerId, ya que esto es sólo una columna que Kaggle ha añadido para identificar a cada pasajero y calcular las puntuaciones. También podemos ignorar Survived, ya que esto es lo que estamos prediciendo, así como las tres columnas que ya hemos utilizado.

A continuación se muestra una lista de las columnas restantes (con una breve descripción), seguida de 10 pasajeros seleccionados al azar y sus datos de esas columnas, para que podamos volver a familiarizarnos con los datos.

- SibSp - El número de hermanos o cónyuges que el pasajero tenía a bordo del Titanic
- Parch - El número de padres o hijos que el pasajero tenía a bordo del Titanic
- Ticket - Número de billete del pasajero
- Fare - La tarifa que el pasajero ha pagado
- Cabin - El número de cabina del pasajero
- Embarked - El puerto en el que embarcó el pasajero (C=Cherbourg, Q=Queenstown, S=Southampton)

III.2 Preparación, Selección e Ingeniería de las Características

	Name	SibSp	Parch	Ticket	Fare	Cabin	Embarked
680	Peters, Miss. Katie	0	0	330935	8.1375	NaN	Q
333	Vander Planke, Mr. Leo Edmondus	2	0	345764	18.0000	NaN	S
816	Heininen, Miss. Wendla Maria	0	0	STON/O2. 3101290	7.9250	NaN	S
310	Hays, Miss. Margaret Bechstein	0	0	11767	83.1583	C54	C
291	Bishop, Mrs. Dickinson H (Helen Walton)	1	0	11967	91.0792	B49	C
33	Wheadon, Mr. Edward H	0	0	C.A. 24579	10.5000	NaN	S
761	Nirva, Mr. Iisakki Antino Aijo	0	0	SOTON/O2 3101272	7.1250	NaN	S
305	Allison, Master. Hudson Trevor	1	2	113781	151.5500	C22 C26	S
210	Ali, Mr. Ahmed	0	0	SOTON/O.Q. 3101311	7.0500	NaN	S
272	Mellinger, Mrs. (Elizabeth Anne Maidment)	0	1	250644	19.5000	NaN	S

III.2 Preparación, Selección e Ingeniería de las Características

A primera vista, las columnas Name y Ticket parecen ser únicas para cada pasajero. Volveremos a hablar de estas columnas más adelante, pero por ahora nos centraremos en las demás columnas.

Podemos utilizar el Dataframe.describe() method para obtener más información sobre los valores de las demás columnas.

```
columns = ['SibSp','Parch','Fare','Cabin','Embarked']
print(train[columns].describe(include='all',percentiles=[]))
```

```
SibSp         Parch        Fare Cabin Embarked
count   891.000000  891.000000  891.000000   204      889
unique         NaN         NaN         NaN   147        3
top            NaN         NaN         NaN    G6        S
freq           NaN         NaN         NaN     4      644
mean      0.523008    0.381594   32.204208   NaN      NaN
std       1.102743    0.806057   49.693429   NaN      NaN
min       0.000000    0.000000    0.000000   NaN      NaN
50%       0.000000    0.000000   14.454200   NaN      NaN
max       8.000000    6.000000  512.329200   NaN      NaN
```


III.2 Preparación, Selección e Ingeniería de las Características

De ellas, SibSp, Parch y Fare parecen ser columnas numéricas estándar sin valores perdidos. Cabin tiene valores para sólo 204 de las 891 filas, e incluso entonces la mayoría de los valores son únicos, así que por ahora dejaremos esta columna también. Embarked (Embarcado) parece ser una columna categórica estándar con 3 valores únicos, al igual que PClass, excepto que hay dos valores que faltan. Podemos rellenar fácilmente estos dos valores que faltan con el valor más común, "S", que aparece 644 veces.

Mirando nuestras columnas numéricas, podemos ver una gran diferencia entre el rango de cada una. SibSp tiene valores entre 0-8, Parch entre 0-6, y Fare está en una escala dramáticamente diferente, con valores que van de 0-512. Para asegurarnos de que estos valores tienen la misma ponderación en nuestro modelo, tendremos que **reescalar** los datos.

El reescalado simplemente estira o encoge los datos según sea necesario para que estén en la misma escala, en nuestro caso entre 0 y 1.

III.2 Preparación, Selección e Ingeniería de las Características

A	B	C
1	2	3
2	4	6
3	6	9
4	8	12
5	10	15

Datos antes de reescalar

A	B	C
0	0	0
0.25	0.25	0.25
0.5	0.5	0.5
0.75	0.75	0.75
1	1	1

Datos después de reescalar

En el diagrama anterior, las tres columnas tienen valores mínimos y máximos diferentes antes del reescalado.

Después de reescalar, los valores de cada característica se han comprimido o estirado para que todos estén en la misma escala - tienen el mismo mínimo y máximo, y la relación entre cada punto sigue siendo la misma en relación con otros puntos de esa característica. Ahora puede ver fácilmente que los datos representados en cada columna son idénticos.

Dentro de scikit-learn, la función (**function**) preprocessing.minmax_scale() nos permite reescalar rápida y fácilmente nuestros datos:

```
from sklearn.preprocessing import minmax_scale
columns = ["column one", "column two"]
data[columns] = minmax_scale(data[columns])
```

III.2 Preparación, Selección e Ingeniería de las Características

Procesemos las columnas Embarked, SibSp, Parch y Fare en nuestros marcos de datos de train y holdouts.

Instrucciones

- Para los marcos de datos de train y de holdout (retención):
  - Utilice Series.fillna() method para sustituir cualquier valor que falte en la columna Embarked por "S"
  - Utilice nuestra función create_dummies() para crear columnas ficticias para la columna Embarked
  - Utilice minmax_scale() para reescalar las columnas SibSp, Parch y Fare, asignando los resultados a las nuevas columnas SibSp_scaled, Parch_scaled y Fare_scaled respectivamente

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
from sklearn.preprocessing import minmax_scale
# The holdout set has a missing value in the Fare column which
# we'll fill with the mean.
holdout["Fare"] = holdout["Fare"].fillna(train["Fare"].mean())
columns = ["SibSp","Parch","Fare"]

train["Embarked"] = train["Embarked"].fillna("S")
holdout["Embarked"] = holdout["Embarked"].fillna("S")

train = create_dummies(train,"Embarked")
holdout = create_dummies(holdout,"Embarked")

for col in columns:
    train[col + "_scaled"] = minmax_scale(train[col])
    holdout[col + "_scaled"] = minmax_scale(holdout[col])
```


III.2 Preparación, Selección e Ingeniería de las Características

III.2.2 Determinación de las características más relevantes

Para seleccionar las características con mejor rendimiento, necesitamos una forma de medir cuáles de nuestras características son relevantes para nuestro resultado, en este caso, la supervivencia de cada pasajero. Una forma eficaz es entrenar un modelo de regresión logística utilizando todas nuestras características y, a continuación, observar los coeficientes de cada característica.

LogisticRegression class de scikit-learn tiene un atributo en el que se almacenan los coeficientes después de ajustar el modelo, LogisticRegression.coef_. Primero tenemos que entrenar nuestro modelo, después podemos acceder a este atributo.

```
lr = LogisticRegression()
lr.fit(train_X,train_y)
coefficients = lr.coef_
```


III.2 Preparación, Selección e Ingeniería de las Características

El método coef() devuelve una matriz NumPy de coeficientes, en el mismo orden que las características que se utilizaron para ajustar el modelo. Para facilitar su interpretación, podemos convertir los coeficientes en una serie de pandas, añadiendo los nombres de las columnas como índice:

```
feature_importance = pd.Series(coefficients[0],
                               index=train_X.columns)
```

Ahora ajustaremos un modelo y trazaremos los coeficientes de cada característica.

III.2 Preparación, Selección e Ingeniería de las Características

Instrucciones

- Instanciar un objeto LogisticRegression()
- Ajuste el objeto LogisticRegression utilizando las columnas de la lista de columnas del marco de datos de entrenamiento (train) y la columna objetivo Survived
- Utilice el atributo coef_ para recuperar los coeficientes de las características y asigne los resultados a los coefficients (coeficientes)
- Cree un objeto de serie utilizando los (coefficients)coeficientes, con los nombres de las columnas de las características como índice y asígnelo a feature_importance
- Utilice el método Series.plot.barh() method para trazar la serie feature_importance

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

columns = ['Age_categories_Missing', 'Age_categories_Infant',
       'Age_categories_Child', 'Age_categories_Teenager',
       'Age_categories_Young Adult', 'Age_categories_Adult',
       'Age_categories_Senior', 'Pclass_1', 'Pclass_2', 'Pclass_3',
       'Sex_female', 'Sex_male', 'Embarked_C', 'Embarked_Q', 'Embarked_S',
       'SibSp_scaled', 'Parch_scaled', 'Fare_scaled']
lr = LogisticRegression()
lr.fit(train[columns],train['Survived'])

coefficients = lr.coef_

feature_importance = pd.Series(coefficients[0],
                               index=train[columns].columns)
feature_importance.plot.barh()
plt.show()
```

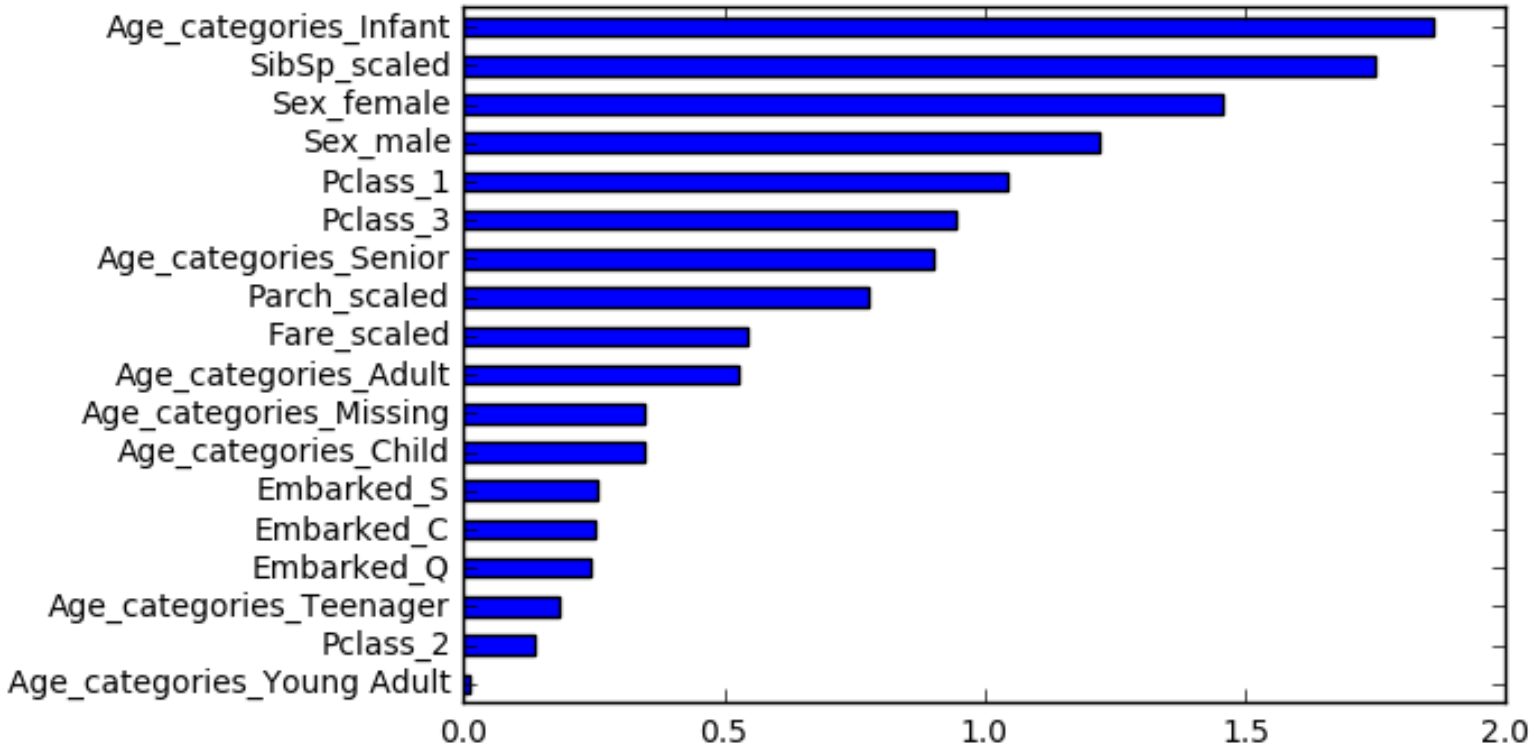

III.2.3 Entrenamiento de un modelo con características relevantes

El gráfico que generamos en la última pantalla mostraba un rango de valores positivos y negativos. Que el valor sea positivo o negativo no es tan importante en este caso, en relación con la magnitud del valor. Si lo piensas, esto tiene sentido. Una característica que indica con fuerza si un pasajero murió es tan útil como una característica que indica con fuerza que un pasajero sobrevivió, dado que son resultados mutuamente excluyentes.

Para facilitar la interpretación, modificaremos el gráfico para que muestre todos los valores positivos, y hemos ordenado las barras por orden de tamaño:

```
ordered_feature_importance = feature_importance.abs().sort_values()
ordered_feature_importance.plot.barh()
plt.show()
```

III.2 Preparación, Selección e Ingeniería de las Características



Entrenaremos un nuevo modelo con las 8 mejores puntuaciones y comprobaremos nuestra precisión utilizando la validación cruzada.

III.2 Preparación, Selección e Ingeniería de las Características

Instruciones

- Instanciar un objeto LogisticRegression()
- Utilice la función model_selection.cross_val_score() function y asigne el objeto devuelto a las puntuaciones, utilizando
- Las columnas especificadas en columns y todas las filas del marco de datos de train
- Un parámetro cv de 10
- Calcule la media de las puntuaciones de validación cruzada y asigne los resultados a accuracy (precisión)
- Utilice la función print() para mostrar la variable accuracy

Soluciones

```
from sklearn.model_selection import cross_val_score

columns = ['Age_categories_Infant', 'SibSp_scaled', 'Sex_female', 'Sex_male',
       'Pclass_1', 'Pclass_3', 'Age_categories_Senior', 'Parch_scaled']
all_X = train[columns]
all_y = train['Survived']

lr = LogisticRegression()
scores = cross_val_score(lr, all_X, all_y, cv=10)
accuracy = scores.mean()
print(accuracy)
```


III.2 Preparación, Selección e Ingeniería de las Características

III.2.4 Envío de nuestro modelo mejorado a Kaggle

La puntuación de validación cruzada del 81,48% es ligeramente superior a la del modelo que creamos en la misión anterior, que tenía una puntuación del 80,2%.

Es de esperar que esta mejora se traduzca en datos no vistos anteriormente. Entrenemos un modelo utilizando las columnas del paso anterior, hagamos algunas predicciones sobre los datos retenidos y enviémoslo a Kaggle para su puntuación.

III.2 Preparación, Selección e Ingeniería de las Características

Instrucciones

- Instanciar un objeto LogisticRegression() y ajustarlo utilizando all_X y all_y
- Utilice el método predict() para realizar predicciones utilizando las mismas columnas del marco de datos holdout, y asigne el resultado a holdout_predictions
- Cree un submission (envío) de marco de datos con dos columnas:
  - PassengerId, con los valores de la columna PassengerId del marco de datos holdout
  - Survived, con los valores de holdout_predictions
- Utilice el método DataFrame.to_csv para guardar el marco de datos de presentación en el archivo submission_1.csv

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
columns = ['Age_categories_Infant', 'SibSp_scaled', 'Sex_female', 'Sex_male',
       'Pclass_1', 'Pclass_3', 'Age_categories_Senior', 'Parch_scaled']

all_X = train[columns]
all_y = train['Survived']
lr = LogisticRegression()
lr.fit(all_X,all_y)
holdout_predictions = lr.predict(holdout[columns])

holdout_ids = holdout["PassengerId"]
submission_df = {"PassengerId": holdout_ids,
                 "Survived": holdout_predictions}
submission = pd.DataFrame(submission_df)

submission.to_csv("submission_1.csv",index=False)
```


III.2.5 Ingeniería de una nueva característica utilizando Binning

Puedes descargar el CSV del paso anterior **aquí**. Cuando lo envíes a Kaggle, verás que la puntuación es del 77,0%, lo que en el momento de escribir este artículo equivale a subir unos 1.500 puestos en la tabla de clasificación (esto variará, ya que la tabla de clasificación siempre cambia). Es sólo una pequeña mejora, pero vamos en la dirección correcta.

Gran parte de las ganancias de precisión en el aprendizaje automático provienen de la **Ingeniería de Características**. La ingeniería de características es la práctica de crear nuevas características a partir de los datos existentes.

Una forma común de diseñar una característica es utilizar una técnica llamada binning. El binning consiste en tomar una característica continua, como la tarifa que un pasajero ha pagado por su billete, y separarla en varios rangos (o "bins"), convirtiéndola en una variable categórica.

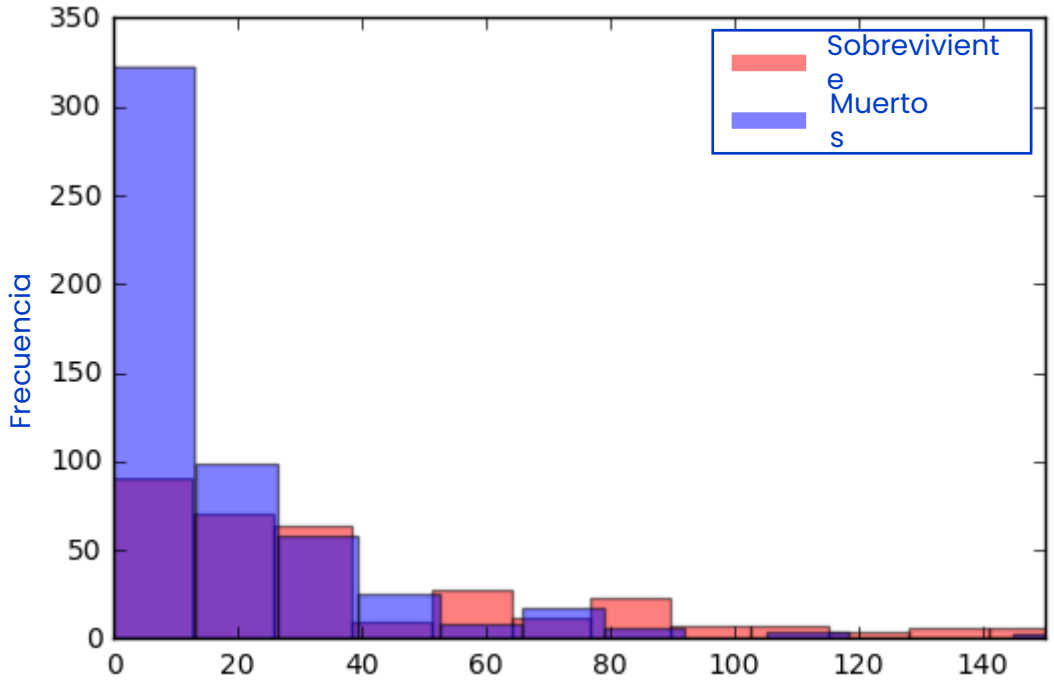
III.2 Preparación, Selección e Ingeniería de las Características

Esto puede ser útil cuando hay patrones en los datos que no son lineales y usted está utilizando un modelo lineal (como la regresión logística). En realidad utilizamos el binning en la misión anterior cuando tratamos la columna Age, aunque no utilizamos el término.

Veamos los histogramas de la columna Fare para los pasajeros que murieron y sobrevivieron, y veamos si hay patrones que podamos utilizar al crear nuestros bins.

Si observamos los valores, parece que podemos separar la característica en cuatro franjas para capturar algunos patrones de los datos:

- 0-12
- 12-50
- 50-100
- 100+



Al igual que en la misión anterior, podemos utilizar la **función pandas.cut()** para crear nuestros bins.

III.2 Preparación, Selección e Ingeniería de las Características

Instrucciones

- Utilizar la función process_age() como modelo, crear una función process_fare() que utilice el método pandas cut() para crear bins para la columna Fare y asignar los resultados a una nueva columna llamada Fare_categories
  - Ya hemos tratado los valores que faltan en la columna Fare, así que no necesitará la línea que utiliza fillna()
- Utilice la función process_fare() en los marcos de datos del train y de la holdout, creando las cuatro "bins" o "franjas":
  - 0-12, para valores entre 0 y 12
  - 12-50, para valores entre 12 y 50
  - 50-100, para valores entre 50 y 100
  - 100+, para valores entre 100 y 1000
- Utilice la función create_dummies() que creamos anteriormente en la misión, tanto en los marcos de datos de train como en los de holdout, para crear columnas ficticias basadas en nuestros nuevos intervalos de tarifas

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
def process_age(df,cut_points,label_names):
    df["Age"] = df["Age"].fillna(-0.5)
    df["Age_categories"] = pd.cut(df["Age"],cut_points,labels=label_names)
    return df
def process_fare(df,cut_points,label_names):
    df["Fare_categories"] = pd.cut(df["Fare"],cut_points,labels=label_names)
    return df

cut_points = [0,12,50,100,1000]
label_names = ["0-12","12-50","50-100","100+"]

train = process_fare(train,cut_points,label_names)
holdout = process_fare(holdout,cut_points,label_names)

train = create_dummies(train,"Fare_categories")
holdout = create_dummies(holdout,"Fare_categories")
```


III.2 Preparación, Selección e Ingeniería de las Características

III.2.6 Características de ingeniería de las columnas de texto

Otra forma de diseñar características es extrayendo datos de las columnas de texto. Anteriormente, decidimos que las columnas Name y Cabin no eran útiles por sí mismas, pero ¿y si hay algún dato que podamos extraer? Veamos una muestra aleatoria de filas de esas dos columnas:

	Name	Cabin
772	Mack, Mrs. (Mary)	E77
148	Navratil, Mr. Michel ("Louis M Hoffman")	F2
707	Calderhead, Mr. Edward Pennington	E24
879	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	C50
21	Beesley, Mr. Lawrence	D56
456	Millet, Mr. Francis Davis	E38
97	Greenfield, Mr. William Bertram	D10 D12
263	Harrison, Mr. William	B94

III.2 Preparación, Selección e Ingeniería de las Características

Aunque aisladamente el número de cabina de cada pasajero será razonablemente único para cada uno, podemos ver que el formato de los números de cabina es una letra seguida de dos números. Parece que la letra es representativa del tipo de cabina, lo que podría ser un dato útil para nosotros. Podemos usar el accesorio **Series.str accessor** de pandas y luego subsumir el primer carácter usando paréntesis:

```
print(train.head()["Cabin"])
```

```
0     NaN
1     C85
2     NaN
3    C123
4     NaN
Name: Cabin, dtype: object
```

```
print(train.head()["Cabin"].str[0])
```

```
0    NaN
1      C
2    NaN
3      C
4    NaN
Name: Cabin, dtype: object
```

III.2 Preparación, Selección e Ingeniería de las Características

Si nos fijamos en la columna de Name, en cada uno de ellos hay un título como "señor" o "señora", así como algunos títulos menos comunes, como el de "condesa" de la última fila de nuestra tabla anterior. Si dedicamos algún tiempo a investigar los diferentes títulos, podemos clasificarlos en seis tipos:

- Mr
- Mrs
- Master
- Miss
- Officer
- Royalty

Podemos utilizar el Series.str.extract method y una **regular expression** para extraer el título de cada nombre y luego utilizar el Series.map() method y un diccionario predefinido para simplificar los títulos.

```
titles = {
    "Mme":          "Mrs",
    "Ms":           "Mrs",
    "Mrs" :         "Mrs",
    "Countess":     "Royalty",
    "Lady" :        "Royalty"
}

extracted_titles = train["Name"].str.extract(' ([A-Za-z]+)\.', expand=False)
train["Title"] = extracted_titles.map(titles)
```

III.2 Preparación, Selección e Ingeniería de las Características

Instrucciones

- Utilice extract(), map() y el diccionario de títles (titulos) para categorizar los títulos del marco de datos holdout y asigne los resultados a una nueva columna Título (Titulo)
- Para los marcos de datos de train y de holdout:
  - Utilice el accesorio str() para extraer la primera letra de la columna Cabin y asigne el resultado a una nueva columna Cabin_type
  - Utilice el método fillna() para rellenar cualquier valor que falte en Cabin_type con "Unknown"
- Para las columnas recién creadas Title y Cabin_type, utilice create_dummies() para crear columnas ficticias para los marcos de datos del train y de holdout

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
titles = {
    "Mr" :          "Mr",
    "Mme":          "Mrs",
    "Ms":           "Mrs",
    "Mrs" :         "Mrs",
    "Master" :      "Master",
    "Mlle":         "Miss",
    "Miss" :        "Miss",
    "Capt":         "Officer",
    "Col":          "Officer",
    "Major":        "Officer",
    "Dr":           "Officer",
    "Rev":          "Officer",
    "Jonkheer":     "Royalty",
    "Don":          "Royalty",
    "Sir" :         "Royalty",
    "Countess":     "Royalty",
    "Dona":         "Royalty",
    "Lady" :        "Royalty"
}

extracted_titles = train["Name"].str.extract(' ([A-Za-z]+)\.',expand=False)
train["Title"] = extracted_titles.map(titles)
extracted_titles = holdout["Name"].str.extract(' ([A-Za-z]+)\.',expand=False)
holdout["Title"] = extracted_titles.map(titles)

train["Cabin_type"] = train["Cabin"].str[0]
train["Cabin_type"] = train["Cabin_type"].fillna("Unknown")
```

III.2.7 Búsqueda de características correlacionadas

Ahora tenemos 34 posibles columnas de características que podemos utilizar para entrenar nuestro modelo. Una cosa que hay que tener en cuenta al empezar a añadir más características es un concepto llamado **colinealidad**. La colinealidad ocurre cuando más de una característica contiene datos que son similares.

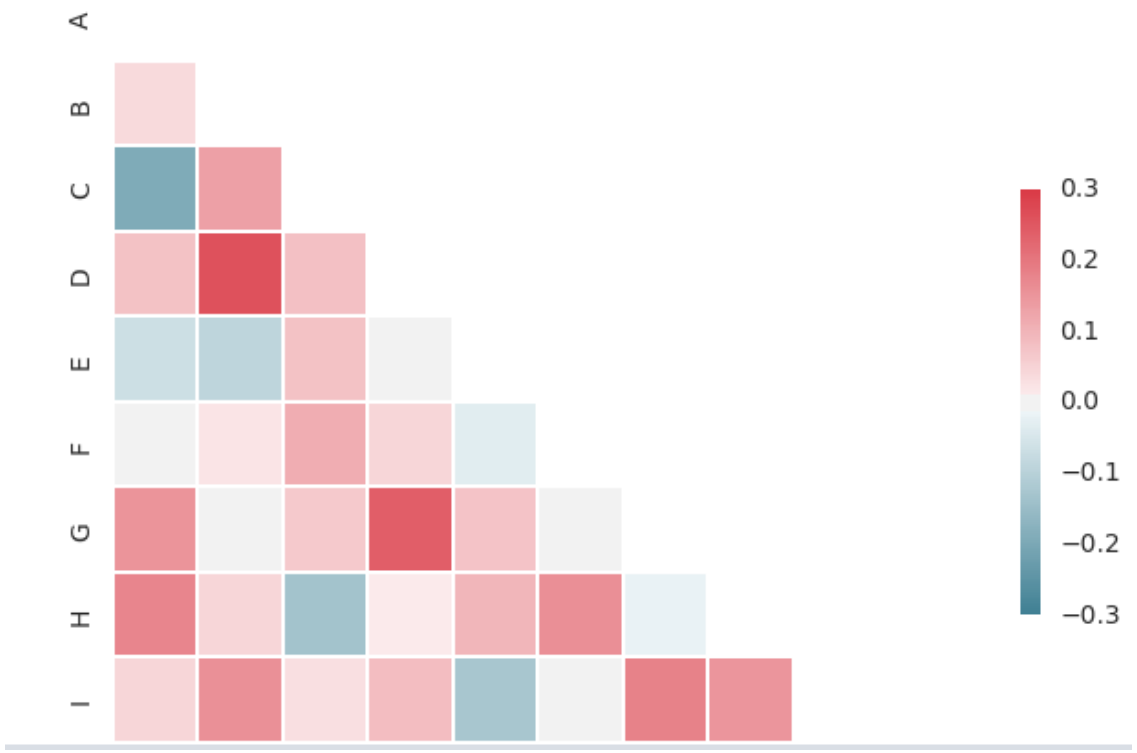
El efecto de la colinealidad es que su modelo se sobreajustará: puede obtener grandes resultados en su conjunto de datos de prueba, pero luego el modelo se desempeña peor en los datos no vistos (como el conjunto de retención).

Una forma fácil de entender la colinealidad es con una simple variable binaria como la columna Sex en nuestro conjunto de datos. Cada pasajero de nuestros datos está clasificado como male o female, por lo que "no male" es exactamente lo mismo que "female".

III.2 Preparación, Selección e Ingeniería de las Características

Como resultado, cuando creamos nuestras dos columnas ficticias a partir de la columna categórica Sex, en realidad hemos creado dos columnas con datos idénticos en ellas. Esto ocurrirá siempre que creemos columnas ficticias, y se llama la trampa de la variable ficticia (**dummy variable trap**). La solución fácil es elegir una columna para eliminarla cada vez que cree columnas ficticias.

La colinealidad también puede ocurrir en otros lugares. Una forma común de detectar la colinealidad es trazar correlaciones entre cada par de variables en un mapa de calor. Un ejemplo de este estilo de gráfico es el siguiente:



III.2 Preparación, Selección e Ingeniería de las Características

Los cuadrados más oscuros, ya sea el rojo más oscuro o el azul más oscuro, indican pares de columnas que tienen una mayor correlación y que pueden dar lugar a colinealidad. La forma más fácil de producir este gráfico es utilizar el DataFrame.corr() method para producir una matriz de correlación, y luego utilizar la seaborn.heatmap() function de la biblioteca Seaborn para trazar los valores:

```
import seaborn as sns
correlations = train.corr()
sns.heatmap(correlations)
plt.show()
```

El ejemplo de gráfico de arriba fue producido usando un ejemplo de **código de la documentación de seaborn** que produce un mapa de calor de correlación que es más fácil de interpretar que la salida por defecto de heatmap(). Hemos creado una función que contiene ese código para facilitarle el trazado de las correlaciones entre las características de nuestros datos.

III.2 Preparación, Selección e Ingeniería de las Características

Instrucciones

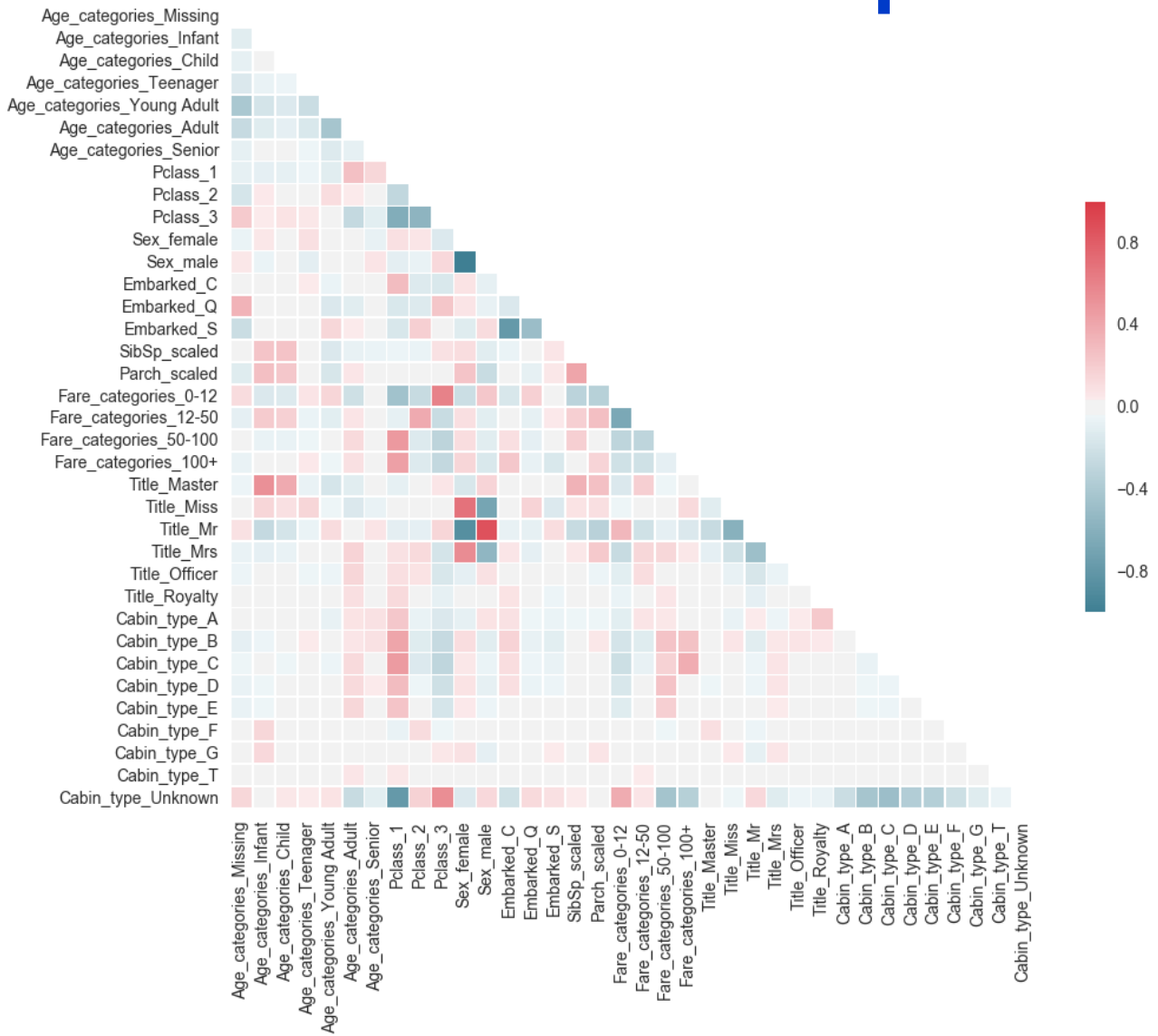
Utilice la función plot_correlation_heatmap() para producir un heatmap (mapa de calor) para el marco de datos de train, utilizando sólo las características de las columnas de la lista.

Soluciones

```
import numpy as np
import seaborn as sns

def plot_correlation_heatmap(df):
    corr = df.corr()

    sns.set(style="white")
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    f, ax = plt.subplots(figsize=(11, 9))
    cmap = sns.diverging_palette(220, 10, as_cmap=True)


    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
    plt.show()
```

```
columns = ['Age_categories_Missing', 'Age_categories_Infant',
       'Age_categories_Child', 'Age_categories_Teenager',
       'Age_categories_Young Adult', 'Age_categories_Adult',
       'Age_categories_Senior', 'Pclass_1', 'Pclass_2', 'Pclass_3',
       'Sex_female', 'Sex_male', 'Embarked_C', 'Embarked_Q', 'Embarked_S',
       'SibSp_scaled', 'Parch_scaled', 'Fare_categories_0-12',
       'Fare_categories_12-50','Fare_categories_50-100', 'Fare_categories_100+',
       'Title_Master', 'Title_Miss', 'Title_Mr','Title_Mrs', 'Title_Officer',
       'Title_Royalty', 'Cabin_type_A','Cabin_type_B', 'Cabin_type_C',
'Cabin_type_D',
       'Cabin_type_E','Cabin_type_F', 'Cabin_type_G', 'Cabin_type_T',
'Cabin_type_Unknown']
plot_correlation_heatmap(train[columns])
```

III.2 Preparación, Selección e Ingeniería de las Características



III.2.8 Selección final de características mediante la RFECV

A continuación reproducimos el gráfico que hemos creado en la pantalla anterior:

III.2 Preparación, Selección e Ingeniería de las Características

Podemos ver que hay una alta correlación entre Sex_female/Sex_male y Title_Mr/Title_Mrs. Eliminaremos las columnas Sex_female y Sex_male ya que los datos de los títulos pueden ser más matizados.

Aparte de eso, deberíamos eliminar una de cada una de nuestras variables ficticias para reducir la colinealidad en cada una. Eliminaremos:

- Pclass_2
- Age_categories_Teenager
- Fare_categories_12-50
- Title_Master
- Cabin_type_A

III.2 Preparación, Selección e Ingeniería de las Características

En un paso anterior, utilizamos manualmente los coeficientes logit para seleccionar las características más relevantes. Un método alternativo es utilizar una de las clases de selección de características incorporadas en scikit-learn. Utilizaremos la feature_selection.RFECV class que realiza la **eliminación recursiva de características** con validación cruzada.

La clase RFECV comienza entrenando un modelo utilizando todas sus características y lo puntúa utilizando la validación cruzada. A continuación, utiliza los coeficientes logit para eliminar la característica menos importante, y entrena y puntúa un nuevo modelo. Al final, la clase mira todas las puntuaciones y selecciona el conjunto de características que han obtenido la mayor puntuación.

Al igual que la clase LogisticRegression, la RFECV debe instanciarse primero y luego ajustarse. El primer parámetro al crear el objeto RFECV debe ser un estimador, y necesitamos utilizar el parámetro cv para especificar el número de pliegues para la validación cruzada.

```
from sklearn.feature_selection import RFECV
lr = LogisticRegression()
selector = RFECV(lr,cv=10)
selector.fit(all_X,all_y)
```


III.2 Preparación, Selección e Ingeniería de las Características

Una vez ajustado el objeto RFECV, podemos utilizar el atributo RFECV.support para acceder a una máscara booleana de valores True y False que podemos utilizar para generar una lista de columnas optimizadas:

```
optimized_columns = all_X.columns[selector.support_]
```

Instrucciones

- Instanciar un objeto LogisticRegression(), lr
- Instanciar un objeto selector RFECV() utilizando el objeto lr recién creado y cv=10 como parámetros
- Utilice el método fit() para ajustar el selector utilizando all_X y all_y
- Utilice el selector de atributos de support (soporte) para subconjuntar all_X.columns, y asigne el resultado a optimized_columns

Debido al cálculo que implica este ejercicio, la ejecución del código puede llevar más tiempo que otras pantallas.

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
all_X = train[columns]
all_y = train["Survived"]
lr = LogisticRegression()
selector = RFECV(lr,cv=10)
selector.fit(all_X,all_y)

optimized_columns = all_X.columns[selector.support_]
```


III.2 Preparación, Selección e Ingeniería de las Características

III.2.9 Entrenamiento de un modelo con nuestras columnas optimizadas

El selector RFECV() sólo devuelve cuatro columnas:

```
['SibSp_scaled', 'Title_Mr', 'Title_Officer', 'Cabin_type_Unknown']
```

Vamos a entrenar un modelo mediante validación cruzada utilizando estas columnas y comprobar la puntuación.

Instrucciones

- Instanciar el objeto LogisticRegression()
  - Utilice la función model_selection.cross_val_score() function y asigne los resultados a las puntuaciones, utilizando
  - all_X y all_y
- Un parámetro cv de 10
- Calcule la media de las puntuaciones de validación cruzada y asigne los resultados a accuracy (precisión)

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
all_X = train[optimized_columns]
all_y = train["Survived"]
lr = LogisticRegression()
scores = cross_val_score(lr, all_X, all_y, cv=10)
accuracy = scores.mean()
```


III.2.10 Envío de nuestro modelo a Kaggle

Este modelo de cuatro características obtiene una puntuación del 82,3%, una modesta mejora en comparación con el 81,5% de nuestro modelo anterior. Vamos a entrenar estas columnas en el conjunto de retención, guardar un archivo de presentación y ver qué puntuación obtenemos de Kaggle.

Instrucciones

- Instanciar un objeto LogisticRegression() y ajustarlo usando all_X y all_y
- Utiliza el método predict() para hacer predicciones utilizando las mismas columnas en el marco de datos holdout, y asigna el resultado a holdout_predictions
- Cree un submission (envío)de marco de datos con dos columnas:
- PassengerId, con los valores de la columna PassengerId del marco de datos holdout
- Survived, con los valores de holdout_predictions
- Utilice el método DataFrame.to_csv para guardar el marco de datos de submission en el archivo submission_2.csv

III.2 Preparación, Selección e Ingeniería de las Características

Soluciones

```
lr = LogisticRegression()
lr.fit(all_X,all_y)
holdout_predictions = lr.predict(holdout[optimized_columns])

holdout_ids = holdout["PassengerId"]
submission_df = {"PassengerId": holdout_ids,
                 "Survived": holdout_predictions}
submission = pd.DataFrame(submission_df)

submission.to_csv("submission_2.csv",index=False)
```

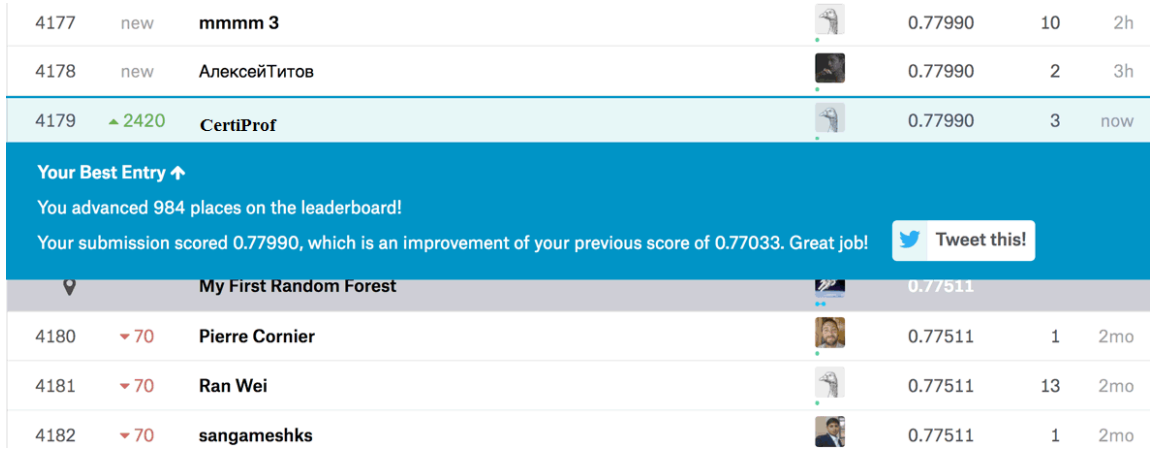
Puedes descargar el archivo de presentación que acabamos de crear y enviarlo a Kaggle. La puntuación que obtiene este envío es del 78,0%, lo que equivale a un salto de aproximadamente 1.000 puestos (de nuevo, esto variará, ya que los envíos se realizan constantemente a la tabla de clasificación).

III.2 Preparación, Selección e Ingeniería de las Características

Mediante la preparación, ingeniería y selección de características, hemos aumentado nuestra precisión en un-2,4%. Cuando se trabaja en las competiciones de Kaggle, hay que dedicar mucho tiempo a experimentar con las características, sobre todo con la ingeniería de características. Aquí hay algunas ideas que puedes usar para trabajar con características para esta competición:

- Utilizar SibSp y Parch para explorar el total de parientes a bordo
- Crear combinaciones de varias columnas, por ejemplo Pclass + Sex
- Vea si puede extraer datos útiles de la columna Ticket
- Prueba diferentes combinaciones de características para ver si puedes identificar las características que se ajustan menos que otras

En la próxima misión de este curso, veremos cómo seleccionar y optimizar diferentes modelos para mejorar nuestra puntuación.



III.3 Selección y Ajuste del Modelo

III.3.1 Selección del modelo

En la misión anterior, trabajamos para optimizar nuestras predicciones creando y seleccionando las características utilizadas para entrenar nuestro modelo. La otra mitad del rompecabezas de la optimización consiste en optimizar el modelo en sí, o más concretamente, el algoritmo utilizado para entrenar nuestro modelo.

Hasta ahora, hemos utilizado el algoritmo de **regresión logística** para entrenar nuestros modelos, pero hay cientos de algoritmos de aprendizaje automático diferentes entre los que podemos elegir. Cada algoritmo tiene diferentes puntos fuertes y débiles, por lo que tenemos que seleccionar el algoritmo que mejor funcione con nuestros datos específicos, en este caso nuestra competición de Kaggle.

III.3 Selección y Ajuste del Modelo

El proceso de selección del algoritmo que ofrece las mejores predicciones para sus datos se denomina **selección del modelo**.

En esta misión, vamos a trabajar con dos nuevos algoritmos: k- vecinos más cercanos y bosques aleatorios.

Antes de empezar, tendremos que importar los datos. Para ahorrar tiempo, hemos guardado las características que creamos en la misión anterior como archivos CSV, train_modified.csv y holdout_modified.csv

Instrucciones

- Importe train_modified.csv en un grupo de datos pandas y asigne el resultado a train
- Importe holdout_modified.csv en un grupo de datos pandas y asigne el resultado a holdout

III.3 Selección y Ajuste del Modelo

Soluciones

```
import pandas as pd
train = pd.read_csv('train_modified.csv')
holdout = pd.read_csv('holdout_modified.csv')
```


III.3 Selección y Ajuste del Modelo

III.3.2 Entrenamiento de un modelo de referencia

Vamos a entrenar nuestros modelos utilizando todas las columnas del marco de datos de entrenamiento. Esto causará una pequeña cantidad de sobreajuste debido a la colinealidad (como discutimos en la misión anterior), pero tener más características nos permitirá comparar más a fondo los algoritmos.

Para tener algo con lo que comparar, vamos a entrenar un modelo de regresión logística como en las dos misiones anteriores. Utilizaremos la validación cruzada para obtener una puntuación de referencia.

III.3 Selección y Ajuste del Modelo

Instrucciones

- Instanciar un linear_model.LogisticRegression class
- Utilice el model_selection.cross_val_score() function para entrenar y probar un modelo asignando los resultados a score, usando:
  - El objeto LogisticRegression que acabas de crear
  - all_X y all_y como parámetros X e y
  - 10 folds (pliegues)
- Calcule la media de los scores (puntuaciones) y asigne el resultado a accuracy_lr

Soluciones

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

all_X = train.drop(['Survived','PassengerId'],axis=1)
all_y = train['Survived']
lr = LogisticRegression()
scores = cross_val_score(lr, all_X, all_y, cv=10)
accuracy_lr = scores.mean()
```

III.3 Selección y Ajuste del Modelo

III.3.3 Entrenamiento de un modelo mediante K-Nearest Neighbors

El modelo de referencia de regresión logística de la pantalla anterior obtuvo el 82,5%.

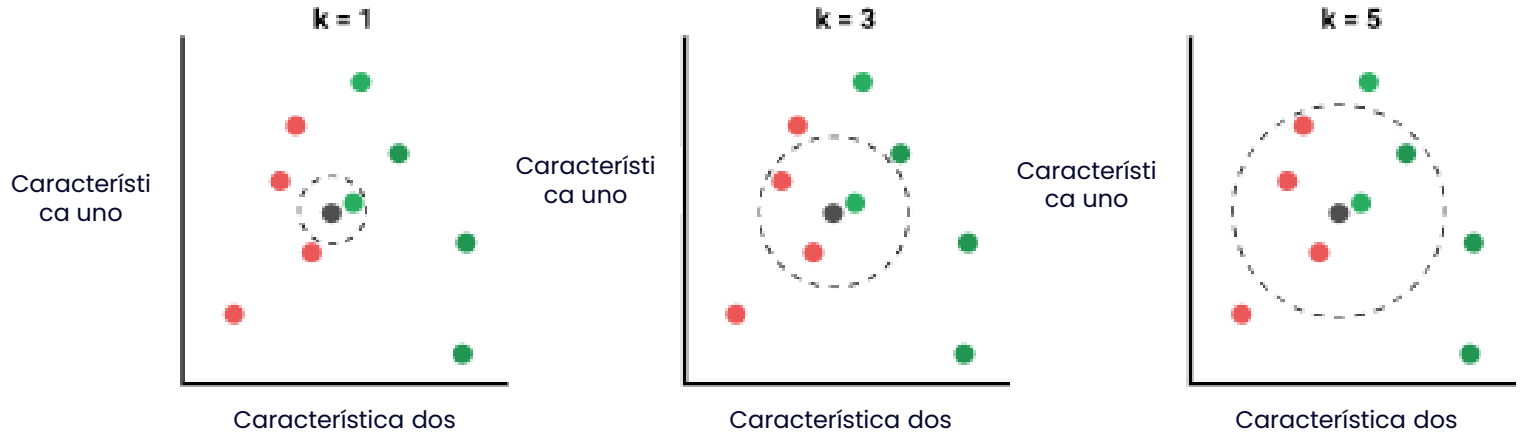
Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	

El algoritmo de regresión logística funciona calculando las relaciones lineales entre las características y la variable objetivo y utilizándolas para hacer predicciones. Veamos un algoritmo que hace predicciones utilizando un método diferente.

El algoritmo de k-próximos encuentra las observaciones de nuestro conjunto de entrenamiento más similares a la observación de nuestro conjunto de prueba, y utiliza el resultado medio de esas observaciones "vecinas" para hacer una predicción. La "k" es el número de observaciones vecinas utilizadas para hacer la predicción.

III.3 Selección y Ajuste del Modelo

Los gráficos siguientes muestran tres modelos simples de vecinos más cercanos donde hay dos características en cada eje y dos resultados, rojo y verde:



- En el primer gráfico, el valor de k es 1. El punto verde es, por tanto, el vecino más cercano al punto gris, lo que hace que la predicción sea **verde**
- En el segundo gráfico, el valor de k es 3. Se utilizan los 3 vecinos más cercanos a nuestro punto gris (2 rojos frente a 1 verde), lo que hace que la predicción sea **roja**
- En el tercer gráfico, el valor de k es 5. Se utilizan los 5 vecinos más cercanos a nuestro punto gris (3 rojos frente a 2 verdes), lo que hace que la predicción sea **roja**

III.3 Selección y Ajuste del Modelo

Si quieres aprender más sobre el algoritmo k-nearest neighbors, puede que te guste nuestra misión gratuita **Introduction to K-Nearest Neighbors.**

Al igual que para la regresión logística, scikit-learn tiene una clase que facilita el uso de los vecinos más cercanos para hacer predicciones, **neighbors.KNeighborsClassifier.**

El uso de Scikit-learn del diseño orientado a objetos hace que sea fácil sustituir un modelo por otro. La sintaxis para instanciar un KNeighborsClassifier es muy similar a la que utilizamos para la regresión logística.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

El argumento opcional n_vecinos establece el valor de k cuando se hacen las predicciones. El valor por defecto de n_vecinos es 5, pero vamos a empezar construyendo un modelo simple que utiliza el vecino más cercano para hacer nuestras predicciones.

III.3 Selección y Ajuste del Modelo

Instrucciones

- Instanciar un objeto neighbors.KNeighborsClassifier, estableciendo el argumento n_neighbors en 1
- Utilice la función model_selection.cross_val_score() para entrenar y probar un modelo asignando el resultado a las scores (puntuaciones), utilizando:
  - El objeto KNeighborsClassifier que acaba de crear
  - all_X y all_y como los parámetros X e y
  - 10 folds(pliegues)
- Calcula la media de las scores(puntuaciones) y asigna el resultado a accuracy_knn

Soluciones

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)

scores = cross_val_score(knn, all_X, all_y, cv=10)
accuracy_knn = scores.mean()
```


III.3 Selección y Ajuste del Modelo

III.3.4 Exploración de diferentes valores K

The k-nearest neighbors model we trained in the previous screen had an accuracy score of 78.3%, worse than our baseline score of 82.5%.

Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	
K-nearest neighbors, `k == 1`	78.3%	

Además de la selección pura del modelo, podemos variar los ajustes de cada modelo, por ejemplo, el valor de k en nuestro modelo de k vecinos más cercanos. Esto se llama **optimización de hiperparámetros**.

Podemos utilizar un bucle y la clase range() de Python (range() class) para iterar a través de diferentes valores de k y **calcular la puntuación de precisión** para cada valor diferente. Sólo querremos probar los valores impares de k para evitar los empates, en los que los resultados de "sobrevivido" y "muerto" tendrían el mismo número de vecinos.

III.3 Selección y Ajuste del Modelo

Esta es la sintaxis que utilizaríamos para obtener los valores impares entre 1-7 de range():

```
for k in range(1,8,2):
    print(k)
```

```
1
3
5
7
```

Obsérvese que utilizamos los argumentos (1,8,2) para obtener valores entre 1 y 7, ya que el objeto range() creado contiene números hasta el 8, pero sin incluirlo.

Utilicemos esta técnica para calcular la precisión de nuestro modelo para valores de k entre 1 y 49, almacenando los resultados en un diccionario.

Para facilitar la comprensión de los resultados, terminaremos trazando las puntuaciones. Hemos proporcionado una función de ayuda, plot_dict() que puede utilizar para trazar fácilmente el diccionario.

III.3 Selección y Ajuste del Modelo

Instrucciones

- Utiliza un bucle for y la clase range para iterar sobre los valores impares de k desde 1-49, y en cada iteración:
  - Instanciar un objeto KNeighborsClassifier con el valor de k para el argumento n_neighbors
  - Utilizar cross_val_score para crear una lista de puntuaciones utilizando el objeto KNeighborsClassifier recién creado, utilizando all_X, all_y, y cv=10 como argumentos
  - Calcular la media de la lista de puntuaciones
  - Añada la media de las puntuaciones al diccionario knn_scores, utilizando k como clave
- Utilice la función de ayuda plot_dict() para trazar el diccionario knn_scores

III.3 Selección y Ajuste del Modelo

```
import matplotlib.pyplot as plt

def plot_dict(dictionary):
    pd.Series(dictionary).plot.bar(figsize=(9,6),
                                   ylim=
(0.78,0.83),rot=0)
    plt.show()

knn_scores = dict()
for k in range(1,50,2):
    knn = KNeighborsClassifier(n_neighbors=k)

    scores = cross_val_score(knn, all_X, all_y, cv=10)
    accuracy_knn = scores.mean()
    knn_scores[k] = accuracy_knn

plot_dict(knn_scores)
```


III.3 Selección y Ajuste del Modelo

III.3.5 Automatización de la optimización de hiperparámetros con la búsqueda en cuadrícula

Mirando nuestro gráfico de la pantalla anterior podemos ver que un valor k de 19 nos dio nuestra mejor puntuación, y comprobando el diccionario knn_scores podemos ver que la puntuación fue del 82,4%, idéntica a nuestra línea de base (si no redondeáramos los números veríamos que en realidad es un 0,01% menos precisa).

Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	
K-nearest neighbors, `k == 1`	78.3%	
K-nearest neighbors, `k == 19`	82.4%	

La técnica que acabamos de utilizar se llama búsqueda en cuadrícula: entrenamos una serie de modelos a través de una "cuadrícula" de valores y luego buscamos el modelo que nos dio la mayor precisión.

Scikit-learn tiene una clase para realizar la búsqueda en cuadrícula, model_selection.GridSearchCV(). El "CV" en el nombre indica que estamos realizando tanto la búsqueda en la cuadrícula como la validación cruzada al mismo tiempo.

III.3 Selección y Ajuste del Modelo

Creando un diccionario de parámetros y posibles valores y pasándolo al objeto GridSearchCV se puede automatizar el proceso. Este es el aspecto del código de la pantalla anterior, cuando se implementa utilizando la clase GridSearchCV.

```
from sklearn.model_selection import GridSearchCV

knn = KNeighborsClassifier()

hyperparameters = {
    "n_neighbors": range(1,50,2)
}
grid = GridSearchCV(knn, param_grid=hyperparameters,
cv=10)
grid.fit(all_X, all_y)

print(grid.best_params_)
print(grid.best_score_)
```

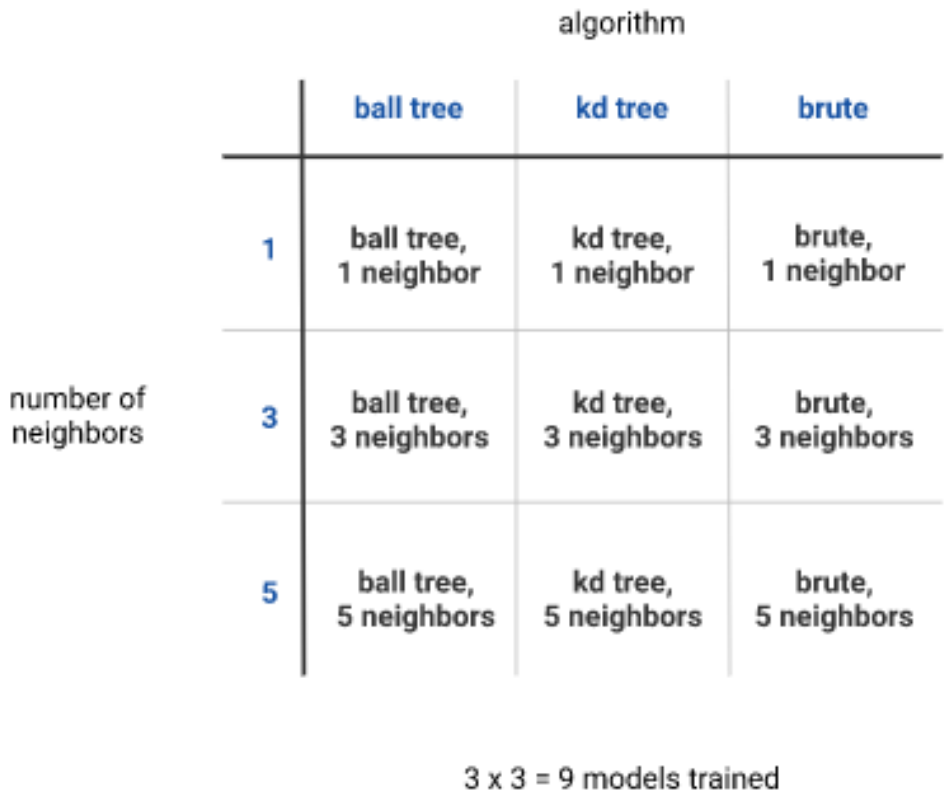
La ejecución de este código producirá el siguiente resultado:

```
{'n_neighbors': 19}
0.82379349046
```

III.3 Selección y Ajuste del Modelo

Nuestro último paso es imprimir los atributos GridSearchCV.best_params_ y GridSearchCV.best_score_ para recuperar los parámetros del modelo de mejor rendimiento y la puntuación que ha obtenido.

También podemos utilizar GridSearchCV para probar combinaciones de diferentes hiperparámetros. Digamos que queremos probar valores de "ball_tree", "kd_tree" y "brute" para el parámetro del algoritmo y valores de 1, 3 y 5 para el parámetro del algoritmo n_neighbors. GridSearchCV entrenaría y probaría 9 modelos (3 para el primer hiperparámetro y 3 para el segundo), como se muestra en el siguiente diagrama.



III.3 Selección y Ajuste del Modelo

Utilicemos GridSearchCV para acelerar la búsqueda de los parámetros más eficaces para nuestro modelo, probando 40 combinaciones de tres hiperparámetros diferentes.

Hemos elegido los hiperparámetros específicos consultando la documentación de la clase KNeighborsClassifier.

Instrucciones

- Instanciar un objeto KNeighborsClassifier
- Instanciar un objeto GridSearchCV, utilizando:
  - El objeto KNeighborsClassifier que acaba de crear como primer argumento (sin nombre)
  - El diccionario de hiperparámetros para el param_grid
  - Un cv de 10
- Ajuste el objeto GridSearchCV utilizando all_X y all_y
- Asignar los parámetros del modelo con mejor rendimiento a best_params
- Asignar la puntuación del modelo más eficaz a best_score

III.3 Selección y Ajuste del Modelo

Soluciones

```
from sklearn.model_selection import GridSearchCV

hyperparameters = {
    "n_neighbors": range(1,20,2),
    "weights": ["distance", "uniform"],
    "algorithm": ['brute'],
    "p": [1,2]
}
knn = KNeighborsClassifier()
grid =
GridSearchCV(knn,param_grid=hyperparameters,cv=10)

grid.fit(all_X, all_y)

best_params = grid.best_params_
best_score = grid.best_score_
```


III.3 Selección y Ajuste del Modelo

III.3.6 Envío de predicciones de K-Nearest Neighbors a Kaggle

La puntuación de la validación cruzada para el modelo de mejor rendimiento fue del 82,9%, mejor que nuestro modelo de referencia.

Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	
K-nearest neighbors, `k == 1`	78.3%	
K-nearest neighbors, `k == 19`	82.4%	
K-nearest neighbors, best model from grid search	82.9%	

III.3 Selección y Ajuste del Modelo

Podemos utilizar el atributo GridSearchCV.best_estimator_ para recuperar un modelo entrenado con los hiperparámetros de mejor rendimiento. Este código:

```
best_knn = grid.best_estimator_
```

Is equivalent to this code Es equivalente a este código donde especificamos manualmente los hiperparámetros y entrenamos el modelo:

```
best_knn =
KNeighborsClassifier(p=1,algorithm='brute',n_neighbors=5,
                     weights='uniform')
best_knn.fit(all_X,all_y)
```

Utilicemos ese modelo para hacer predicciones en el conjunto de los retenidos y enviemos esas predicciones a Kaggle para ver si hemos mejorado en general.

III.3 Selección y Ajuste del Modelo

Instrucciones

- Hacer predicciones sobre los datos de holdout_no_id utilizando el modelo best_knn, y asignar el resultado a holdout_predictions
- Crear un submission (envío) de dataframe con dos columnas:
  - PassengerId, con los valores de la columna PassengerId del marco de datos holdout
  - Survived, con los valores de holdout_predictions
- Utilice el DataFrame.to_csv method para guardar el marco de datos de submission en el archivo submission_1.csv

III.3 Selección y Ajuste del Modelo

Soluciones

```
holdout_no_id = holdout.drop(['PassengerId'],axis=1)
best_knn = grid.best_estimator_
holdout_predictions = best_knn.predict(holdout_no_id)

holdout_ids = holdout["PassengerId"]
submission_df = {"PassengerId": holdout_ids,
                 "Survived": holdout_predictions}
submission = pd.DataFrame(submission_df)

submission.to_csv("submission_1.csv",index=False)
```


III.3 Selección y Ajuste del Modelo

III.3.7 Introducción a los bosques aleatorios

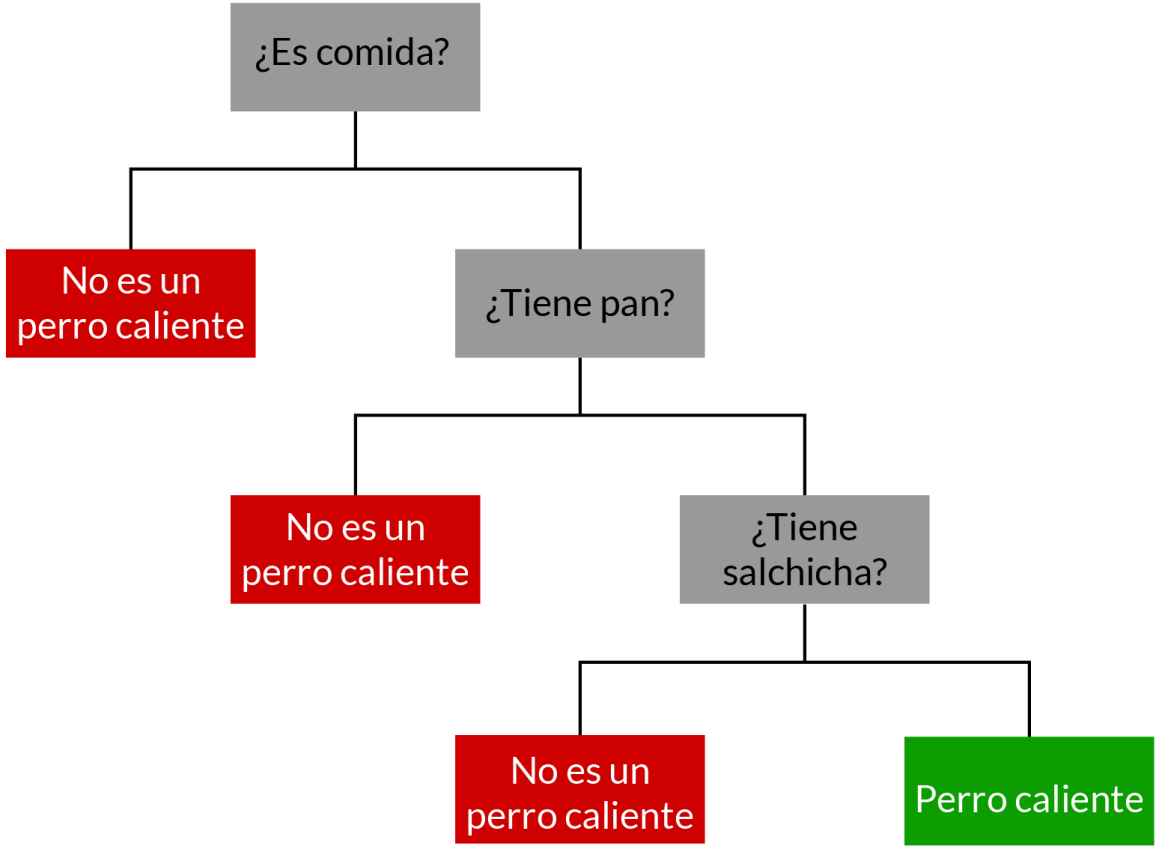
Puedes descargar el archivo de envío de la pantalla anterior **aquí.**

Cuando lo envíes a Kaggle, verás que la puntuación es del 75,6%, menos que nuestra mejor presentación del 78,0%. Si bien nuestro modelo podría estar sobreajustado debido a la inclusión de todas las columnas, también parece que los vecinos más cercanos no son la mejor opción de algoritmo.

Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	
K-nearest neighbors, `k == 1`	78.3%	
K-nearest neighbors, `k == 19`	82.4%	
K-nearest neighbors, best model from grid search	82.8%	75.6%

III.3 Selección y Ajuste del Modelo

Vamos a probar otro algoritmo llamado bosques aleatorios. Los bosques aleatorios son un tipo específico de algoritmo de árbol de decisión. Es probable que hayas visto antes árboles de decisión como parte de diagramas de flujo o infografías. Digamos que queremos construir un árbol de decisión que nos ayude **a categorizar un objeto como 'hotdog' o 'no hotdog'**, podríamos construir un árbol de decisión como el siguiente:



III.3 Selección y Ajuste del Modelo

Los algoritmos del árbol de decisión intentan construir el árbol de decisión más efectivo baso en los datos de entrenamiento, y entonces usa ese árbol para hacer futuras predicciones. Si quiere aprender sobre el árbol de decisión y bosques aleatorios en detalle, debería revisar el **curso de árbol de decisión.**

Scikit-learn contiene una clase para clasificar usando el algoritmo de bosque aleatorio, ensemble.RandomForestClassifier. Asi es como se ajusta el modelo y se hacen predicciones usando la clase RandomForestClassifier:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=1)
clf.fit(train_X,train_y)
predictions = clf.predict(test_X)
```

Porque el algoritmo incluye randomización, temenos que fijar el parametron random_state para asegurarse de que los resultados sean reproducibles.

Usemos un objeto RandomForestClassifier con cross_val_score() como lo hicimos anteriormente para ver como el algoritmo funciona con los hiperparámetros predeterminados.

III.3 Selección y Ajuste del Modelo

Instrucciones

- Instancie un objeto RandomForestClassifier , fijando el parametro random_state a 1
- Use la función cross_val_score() para generar un grupo de puntajes y asignar el resutlado a scores, usando:
  - El objeto RandomForestClassifier que creo como estimador
  - all_X y all_y para los datos de entrenamiento y prueba
  - Un valor cv de10
- Calcule la media de scores y asigne el resutlado a accuracy_rf

Soluciones

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=1)
scores = cross_val_score(clf, all_X, all_y, cv=10)
accuracy_rf = scores.mean()
```


III.3 Selección y Ajuste del Modelo

III.3.8 Ajuste de nuestro modelo de bosques aleatorios con GridSearch

Utilizando la configuración por defecto, nuestro modelo de bosques aleatorios obtuvo una puntuación de validación cruzada del 82,0%.

Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	
K-nearest neighbors, `k == 1`	78.3%	
K-nearest neighbors, `k == 19`	82.4%	
K-nearest neighbors, best model from grid search	82.8%	75.6%
Random forests, default hyperparameters	82.0%	

III.3 Selección y Ajuste del Modelo

Al igual que hicimos con el modelo de k- vecinos más cercanos, podemos utilizar GridSearchCV para probar una variedad de hiperparámetros para encontrar el modelo de mejor rendimiento.

La mejor manera de ver una lista de hiperparámetros disponibles es consultando la documentación del clasificador, en este caso, **the documentation for** RandomForestClassifier. Utilicemos la búsqueda en cuadrícula para probar las combinaciones de los siguientes hiperparámetros:

- criterion: "entropy" o "gini"
- max_depth: 5 o 10
- max_features: "log2" o "sqrt"
- min_samples_leaf: 1 o 5
- min_samples_split: 3 o 5
- n_estimators: 6 o 9

III.3 Selección y Ajuste del Modelo

Instrucciones

- Instanciar un objeto RandomForestClassifier, poniendo el parámetro random_state a 1
- Instanciar un objeto GridSearchCV, utilizando:
  - El objeto RandomForestClassifier que acaba de crear como primer argumento (sin nombre)
  - Un diccionario de hiperparámetros que coincida con la lista anterior para el argumento param_grid
  - Un cv de 10
- Ajuste el objeto GridSearchCV utilizando all_X o all_y
- Asignar los parámetros del modelo con mejor rendimiento a best_params
- Asignar la puntuación del modelo más eficaz a best_score

III.3 Selección y Ajuste del Modelo

Soluciones

```
hyperparameters = {"criterion": ["entropy", "gini"],
                   "max_depth": [5, 10],
                   "max_features": ["log2", "sqrt"],
                   "min_samples_leaf": [1, 5],
                   "min_samples_split": [3, 5],
                   "n_estimators": [6, 9]
}

clf = RandomForestClassifier(random_state=1)
grid =
GridSearchCV(clf,param_grid=hyperparameters,cv=10)

grid.fit(all_X, all_y)

best_params = grid.best_params_
best_score = grid.best_score_
```


III.3 Selección y Ajuste del Modelo

III.3.9 Envío de predicciones del bosque aleatorio a Kaggle

El puntaje de la validación cruzada para el modelo de mejor desempeño fue 83.8%, haciendola la mejor puntuación de validación cruzada que hemos obtenido en esta mission.

Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	
K-nearest neighbors, `k == 1`	78.3%	
K-nearest neighbors, `k == 19`	82.4%	
K-nearest neighbors, best model from grid search	82.8%	75.6%
Random forests, default hyperparameters	82.0%	
Random forests, best model from grid search	83.8%	

III.3 Selección y Ajuste del Modelo

¡Entrenémoslo con los datos de reserva y creemos un archivo de envío para ver cómo se desempeña en la tabla de clasificación de Kaggle!

Instrucciones

- Asignar a best_rf el modelo de mejor rendimiento de la parrilla (grid) de objetos GridSearchCV
- Realizar predicciones sobre los datos de holdout_no_id utilizando el modelo best_rf, y asignar el resultado a holdout_predictions
- Crear un envío (submission) de dataframe con dos columnas:
  - PassengerId, con los valores de la columna PassengerId del dataframe holdout
  - Survived, con los valores de holdout_predictions
- Utilice el DataFrame.to_csv **method** para guardar el marco de datos de presentación (submission) en el archivo submission_2.csv

III.3 Selección y Ajuste del Modelo

Soluciones

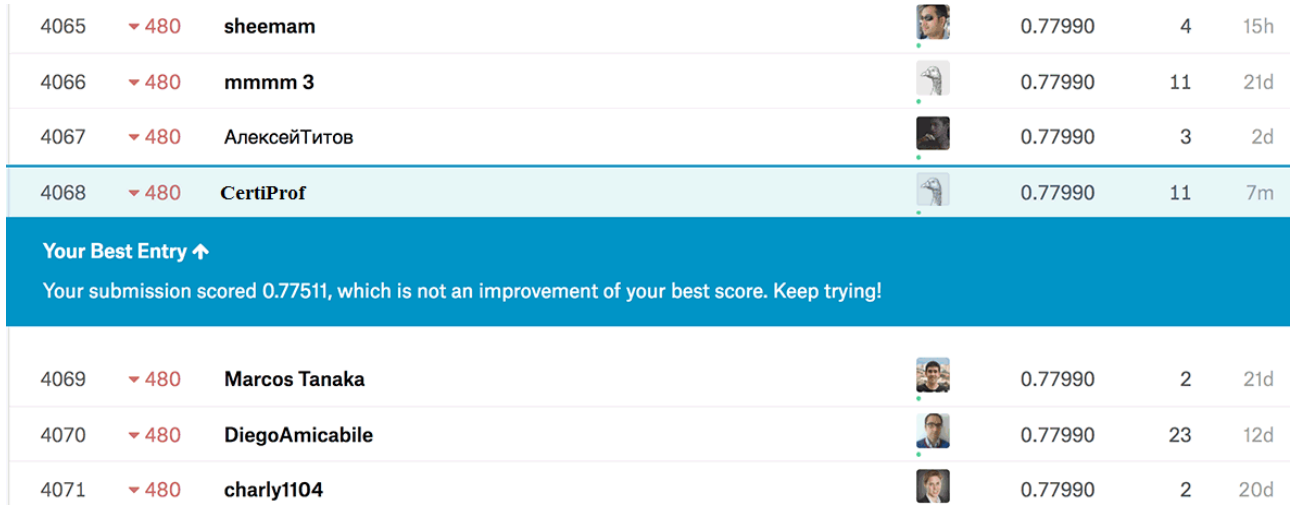
```
# The `GridSearchCV` object is stored in memory from
# the previous screen with the variable name `grid`
best_rf = grid.best_estimator_
holdout_predictions = best_rf.predict(holdout_no_id)

holdout_ids = holdout["PassengerId"]
submission_df = {"PassengerId": holdout_ids,
                 "Survived": holdout_predictions}
submission = pd.DataFrame(submission_df)

submission.to_csv("submission_2.csv",index=False)
```


III.3 Selección y Ajuste del Modelo

III.3.10 Introducción a los bosques aleatorios (Random Forests)

El archivo de presentación que creamos en el paso anterior está disponible.

Si lo envía a Kaggle, consigue una puntuación del 77.1%, considerablemente mejor que nuestra puntuación de k-nearest neighbors del 75.6% y muy cercana (2 predicciones incorrectas) a nuestra mejor puntuación de la misión anterior del 78.0%.



III.3 Selección y Ajuste del Modelo

Model	Cross-validation score	Kaggle score
Previous best Kaggle score	82.3%	78.0%
Logistic regression baseline	82.5%	
K-nearest neighbors, `k == 1`	78.3%	
K-nearest neighbors, `k == 19`	82.4%	
K-nearest neighbors, best model from grid search	82.8%	75.6%
Random forests, default hyperparameters	82.0%	
Random forests, best model from grid search	83.8%	77.1%

Combinando nuestras estrategias de selección de características, ingeniería de características, selección de modelos y ajuste de modelos, podremos seguir mejorando nuestra puntuación.

La siguiente y última misión de este curso es un proyecto guiado, en el que te enseñaremos a combinar todo lo que has aprendido en un flujo de trabajo de Kaggle de la vida real, y a seguir mejorando tu puntuación.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.1 Presentación de los flujos de trabajo de la ciencia de los datos

Hasta ahora en este curso, has estado aprendiendo sobre las competiciones de Kaggle usando misiones de **Dataquest**. Las misiones están muy estructuradas y tu trabajo es revisado en cada paso del camino

Los proyectos guiados, por otro lado, son menos estructurados y se centran más en la exploración. Los proyectos guiados te ayudan a sintetizar los conceptos aprendidos durante las misiones y a practicar lo que has aprendido.

Los proyectos guiados son un puente entre el aprendizaje mediante las misiones de Dataquest y la aplicación de los conocimientos en tu propio ordenador, y tus respuestas no se comprueban como en las misiones normales, aunque puedes acceder a un cuaderno de soluciones utilizando la parte superior de la interfaz.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Trabajar con proyectos guiados es una gran oportunidad para practicar algunas de las habilidades adicionales que necesitarás para hacer ciencia de datos por ti mismo, incluyendo la práctica de la depuración utilizando todas las herramientas a tu disposición, incluyendo la búsqueda de respuestas en Google, visitando **Stack Overflow** y consultando la documentación de los módulos que estás utilizando.

Este proyecto guiado utiliza el **cuaderno Jupyter**, una aplicación web que permite combinar texto y código dentro de un mismo archivo, y que es una de las formas más populares de explorar e iterar cuando se trabaja con datos. **El cuaderno Jupyter** te permite compartir fácilmente tu trabajo, y hace que la exploración de datos sea mucho más fácil.

Si no estás familiarizado con **el cuaderno Jupyter**, te recomendamos que completes nuestro proyecto guiado sobre el **uso del cuaderno Jupyter** para familiarizarte.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

En este proyecto guiado, vamos a reunir todo lo que hemos aprendido en este curso y crear un **flujo de trabajo de ciencia de datos.**

La ciencia de los datos, y en particular el aprendizaje automático, contienen muchas dimensiones de complejidad en comparación con el desarrollo de software estándar. En el desarrollo de software estándar, el código que no funciona como se espera puede ser causado por una serie de factores a lo largo de dos dimensiones:

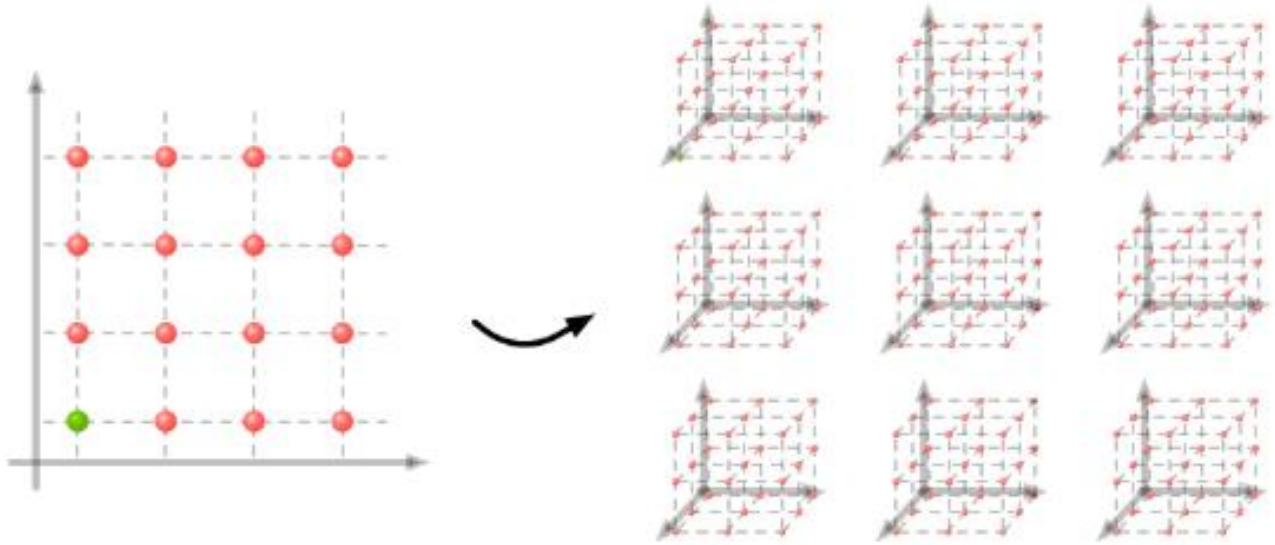
- Errores en la implementación
- Diseño del algoritmo

Los problemas de aprendizaje automático tienen muchas más dimensiones:

- Errores en la implementación
- Diseño del algoritmo
- Problemas con el modelo
- Calidad de los datos

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

El resultado es que hay un número exponencialmente mayor de lugares en los que el aprendizaje automático puede fallar.



Desarrollo de Software

Aprendizaje Automático

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Este concepto se muestra en el diagrama *anterior-tomado del excelente post,* ***¿Por qué es "difícil" el aprendizaje automático?*** El punto verde es una solución "correcta", mientras que los puntos rojos son soluciones incorrectas. En esta ilustración sólo hay un pequeño número de combinaciones incorrectas para la ingeniería de software, pero en el aprendizaje automático esto se vuelve exponencialmente mayor.

Al definir un flujo de trabajo para ti mismo, puedes darte un marco con el que hacer que la iteración de ideas sea más rápida y fácil, permitiéndote trabajar más eficientemente.

En esta misión, vamos a explorar un flujo de trabajo para hacer más fácil competir en el concurso Titanic de Kaggle, utilizando una canalización de funciones para reducir el número de dimensiones en las que hay que centrarse.

Para empezar, leeremos los archivos originales train.csv y test.csv de Kaggle.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Instrucciones

- Importar la biblioteca de pandas
- Utilice pandas para importar el archivo train.csv como train
- Utilice pandas para importar el archivo test.csv como holdout
- Mostrar las primeras líneas del marco de test (datos de prueba)

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.2 Preprocesamiento de datos

Una de las muchas ventajas de utilizar Jupyter es que (por defecto) utiliza el núcleo de Ipython (**IPython kernel**) para ejecutar el código. Esto te da todos los beneficios de IPython, incluyendo la finalización del código y los comandos "mágicos". (Si quieres leer más sobre el funcionamiento interno de Jupyter y cómo puede ayudarte a trabajar de forma más eficiente, puedes consultar nuestra entrada del blog **Jupyter Notebook Tips, Tricks and Shortcuts**.

Podemos utilizar uno de esos comandos mágicos, **the %load command**, para cargar un archivo externo. El comando %load copiará el contenido del archivo en la celda actual del cuaderno. La sintaxis es sencilla:

```
%load [filename]
```

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Para ilustrar, digamos que tenemos un archivo llamado test.py con la siguiente línea de código:

```
print("This is test.py")
```

Para utilizar la carga, simplemente escribimos lo siguiente en una celda de Jupyter:

```
%load test.py
```

Si ejecutamos la celda una vez más, el código se ejecutará, dándonos la salida Esto es test.py.

Hemos creado un archivo, functions.py que contiene versiones de las funciones que creamos en las primeras misiones de este curso, lo que te ahorrará volver a construir esas funciones desde cero.

Vamos a importar ese archivo y a preprocesar nuestros datos de Kaggle.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Instrucciones

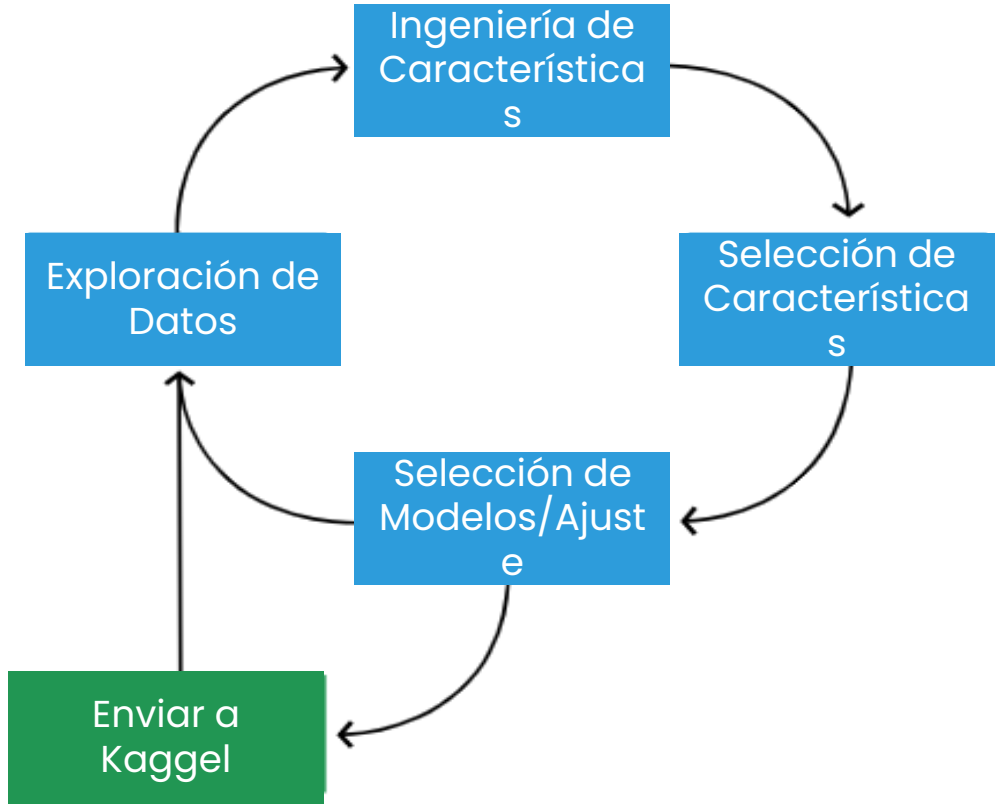
- Utiliza el comando mágico %load para cargar el contenido de functions.py en una celda del bloc de notas y lee las funciones que has importado
- Crea una nueva función que:
  - Acepte un parámetro de marco de datos
  - Aplica las funciones process_missing(), process_age(), process_fare(), process_titles() y process_cabin() al dataframe
  - Aplica la función create_dummies() a las columnas "Age_categories", "Fare_categories", "Title" y "Sex".
  - Devuelve el marco de datos procesado
- Aplica la función recién creada a los marcos de datos de train y de los holdout

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.3 Exploración de los datos

En las tres primeras misiones de este curso, hemos realizado diversas actividades, en su mayoría aisladas: Exploración de los datos, creación de características, selección de características, selección y ajuste de diferentes modelos.

El flujo de **trabajo de Kaggle** que vamos a construir combinará todo esto en un proceso.



III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

- **Exploración de datos,** para encontrar patrones en los datos
- **Ingeniería de características,** para crear nuevas características a partir de esos patrones o a través de la pura experimentación
- **Selección de características,** para seleccionar el mejor subconjunto de nuestro conjunto actual de características
- **Selección/ajuste del modelo,** para entrenar una serie de modelos con diferentes hiperparámetros hasta encontrar el que mejor funcione

Podemos seguir repitiendo este ciclo mientras trabajamos para optimizar nuestras predicciones. Al final de cualquier ciclo que deseemos, también podemos utilizar nuestro modelo para hacer predicciones en el conjunto de espera y luego **enviarlo a Kaggle** para obtener una puntuación en la tabla de clasificación.

Mientras que los dos primeros pasos de nuestro flujo de trabajo son relativamente libres, más adelante en este proyecto crearemos algunas funciones que ayudarán a automatizar la complejidad de los dos últimos pasos para que podamos avanzar más rápido.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Por ahora, vamos a practicar la primera etapa, la exploración de los datos. Vamos a examinar las dos columnas que contienen información sobre los miembros de la familia que cada pasajero llevaba a bordo: SibSp y Parch.

Si necesitas ayuda con las técnicas de exploración y visualización de datos, quizá quieras consultar nuestros cursos de **Análisis de Datos con Pandas** y **Visualización Exploratoria de Datos.**

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Instrucciones

- Revisa el diccionario de datos y las notas de las variables de la competición Titanic en la web de Kaggle para familiarizarte con las columnas SibSp y Parch
- Usa **pandas** y **matplotlib** para explorar esas dos columnas. Puede que te guste probar:
  - Inspeccionar el tipo de las columnas
  - Usar histogramas para ver la distribución de los valores en las columnas
  - Usar tablas dinámicas para ver la tasa de supervivencia para diferentes valores de las columnas
  - Encontrar una forma de combinar las columnas y observar la distribución resultante de los valores y la tasa de supervivencia
- Escribir una celda markdown explicando tus conclusiones

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.4 Ingeniería de nuevas características

Debería haber descubierto en el paso anterior que, combinando los valores de SibSp y Parch en una sola columna, sólo sobrevivieron el 30% de los pasajeros que no tenían familiares a bordo.

Si no ha llegado a esta conclusión, puede utilizar el segmento de código siguiente para comprobarlo por sí mismo:

```
explore_cols = ["SibSp","Parch","Survived"]
explore = train[explore_cols].copy()

explore['familysize'] =
explore[["SibSp","Parch"]].sum(axis=1)
pivot = explore.pivot_table(index=col,values="Survived")
pivot.plot.bar(ylim=(0,1),yticks=np.arange(0,1,.1))
plt.show()
```

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

En base a esto, podemos tener una idea para una nueva característica – estaba el pasajero solo. Esta será una columna binaria que contiene el valor:

- 1 si el pasajero no tiene familiares a bordo
- 0 si el pasajero tiene uno o más miembros de la familia a bordo

Continuemos y creemos esta característica.

Instrucciones

- Cree una función que:
  - Acepte un dataframe como entrada
  - Agregue una nueva columna, isalone, que tenga un valor de 0 si el pasajero tiene 1 o más familiars abordo, y 1 si el pasajero no tiene familiars abordo.
  - Regrese el nuevo dataframe
- Aplique la nueva función creada a los dataframes train y holdout

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.5 Selección de las características más eficaces

El siguiente paso en nuestro flujo de trabajo es la selección de características. En la misión de **Preparación, selección e Ingeniería de las Características**, utilizamos la clase feature_selection.RFECV class de scikit-learn para automatizar la selección de las características de mejor rendimiento mediante la eliminación recursiva de características.

Para acelerar nuestro flujo de trabajo en Kaggle, podemos crear una función que realice este paso por nosotros, lo que significará que podemos realizar la selección de características llamando a una función autónoma y centrar nuestros esfuerzos en la parte más creativa: explorar los datos e ingeniar nuevas características.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Tal vez recuerde que el primer parámetro al instanciar un objeto RFECV() es un estimador. En su momento utilizamos un estimador de Regresión Logística, pero desde entonces hemos descubierto en la misión de **Selección y Ajuste de Modelos** que los Bosques Aleatorios parecen ser un mejor algoritmo para esta competición de Kaggle.

Vamos a escribir una función que:

- Acepte un marco de datos como entrada
- Realiza la preparación de los datos para el aprendizaje automático
- Utiliza la eliminación recursiva de características y el algoritmo de bosques aleatorios para encontrar el conjunto de características de mejor rendimiento

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Instrucciones

- Importar feature_selection.RFECV y ensemble.RandomForestClassifier
- Crear una función, select_features(), que:
  - Acepte un marco de datos como entrada
  - Elimina cualquier columna no numérica o que contenga valores nulos
  - Crea las variables all_X y all_y, asegurándose de que all_X no contiene las columnas PassengerId o Survived
  - Utiliza feature_selection.RFECV y ensemble.RandomForestClassifier para realizar la eliminación recursiva de características utilizando
    - all_X y all_y
    - Un estado aleatorio de 1
    - Validación cruzada de 10 veces
  - Imprime una lista de las mejores columnas de la eliminación recursiva de características
  - Devuelve una lista de las mejores columnas de la eliminación recursiva de características
- Ejecuta la función recién creada utilizando el marco de datos de entrenamiento (train) como entrada y asigna el resultado a una variable

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.6 Selección y ajuste de diferentes algoritmos

Al igual que hicimos con la selección de características, podemos escribir una función que haga el trabajo pesado de la selección y ajuste del modelo. La función que crearemos utilizará tres algoritmos diferentes y utilizará la búsqueda en cuadrícula para entrenar utilizando diferentes combinaciones de hiperparámetros para encontrar el modelo de mejor rendimiento.

Podemos lograr esto mediante la creación de una lista de diccionarios, es decir, una lista donde cada elemento de la lista es un diccionario. Cada diccionario debe contener:

- El nombre del modelo particular
- Un objeto estimador para el modelo
- Un diccionario de hiperparámetros que utilizaremos para la búsqueda en la red

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Este es un ejemplo de cómo será uno de estos diccionarios:

```
{
    "name": "KNeighborsClassifier",
    "estimator": KNeighborsClassifier(),
    "hyperparameters":
        {
            "n_neighbors": range(1,20,2),
            "weights": ["distance", "uniform"],
            "algorithm": ["ball_tree", "kd_tree",
"brute"],
            "p": [1,2]
        }
}
```

A continuación, podemos utilizar un bucle for para iterar sobre la lista de diccionarios, y para cada uno podemos utilizar el model_selection.GridSearchCV class de scikit-learn para encontrar el mejor conjunto de parámetros de rendimiento, y añadir valores para el conjunto de parámetros y la puntuación al diccionario.

Finalmente, podemos devolver la lista de diccionarios, que tendrá nuestros objetos GridSearchCV entrenados, así como los resultados para que podamos ver cuál fue el más preciso.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Instrucciones

- Importe model_selection.GridSearchCV, neighbors import KNeighborsClassifier, y linear_model import LogisticRegression
- Cree una función, select_model(), que:
  - Acepte un dataframe y una lista de caracteristicas como entrada
  - Divide el marco de datos en all_X (que contiene sólo las características del parámetro de entrada) y all_y
  - Contiene una lista de diccionarios, cada uno de los cuales contiene un nombre de modelo, su estimador y un diccionario de hiperparámetros
  - LogisticRegression, utilizando los siguientes hiperparámetros:
    - "solver": ["newton-cg", "lbfgs", "liblinear"]
  - KNeighborsClassifier, utilizando los siguientes hiperparámetros :
    - "n_neighbors": range(1,20,2)
    - "weights": ["distance", "uniform"]
    - "algorithm": ["ball_tree", "kd_tree", "brute"]
    - "p": [1,2]

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

- RandomForestClassifier, utilizando los siguientes hiperparámetros:
  - "n_estimators": [4, 6, 9]
  - "criterion": ["entropy", "gini"]
  - "max_depth": [2, 5, 10]
  - "max_features": ["log2", "sqrt"]
  - "min_samples_leaf": [1, 5, 8]
  - "min_samples_split": [2, 3, 5]
  - Iterar sobre esa lista de diccionarios, y para cada diccionario
    - Imprime el nombre del modelo
    - Instanciar un objeto GridSearchCV() utilizando el modelo, el diccionario de hiperparámetros y especificar la validación cruzada de 10 veces
    - Ajustar el objeto GridSearchCV() utilizando all_X y all_y
    - Asignar los parámetros y la puntuación del mejor modelo al diccionario
    - Asignar al diccionario el mejor estimador del mejor modelo
    - Imprime los parámetros y la puntuación del mejor modelo
  - Devuelve la lista de diccionarios
  - Ejecutar la función recién creada utilizando el marco de datos de train (entrenamiento) y la salida de select_features() como entradas y asignar el resultado a una variable

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.7 Cómo hacer un envío a Kaggle

Después de ejecutar la función, tendrá tres puntuaciones de tres modelos diferentes. En este punto del flujo de trabajo tienes que tomar una decisión: ¿Quieres entrenar a tu mejor modelo en el conjunto de retención y hacer una presentación en Kaggle, o quieres volver a las características de ingeniería.

Es posible que la adición de una característica a su modelo no mejore su precisión. En ese caso, deberías volver a la exploración de datos y repetir el ciclo de nuevo.

Si vas a estar continuamente enviando a Kaggle, una función te ayudará a hacer esto más fácil. Vamos a crear una función para automatizar esto.

Ten en cuenta que en nuestro entorno de Jupyter Notebook, el DataFrame.to_csv() method guardará el CSV en el mismo directorio que tu cuaderno, al igual que lo haría si estás ejecutando Jupyter localmente. Para descargar el CSV de nuestro entorno, puedes hacer clic en el botón 'download' para descargar todos los archivos de tu proyecto como un **tar file**, o hacer clic en el logo de Jupyter en la parte superior de la interfaz, y navegar hasta el propio CSV para descargar sólo ese archivo.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

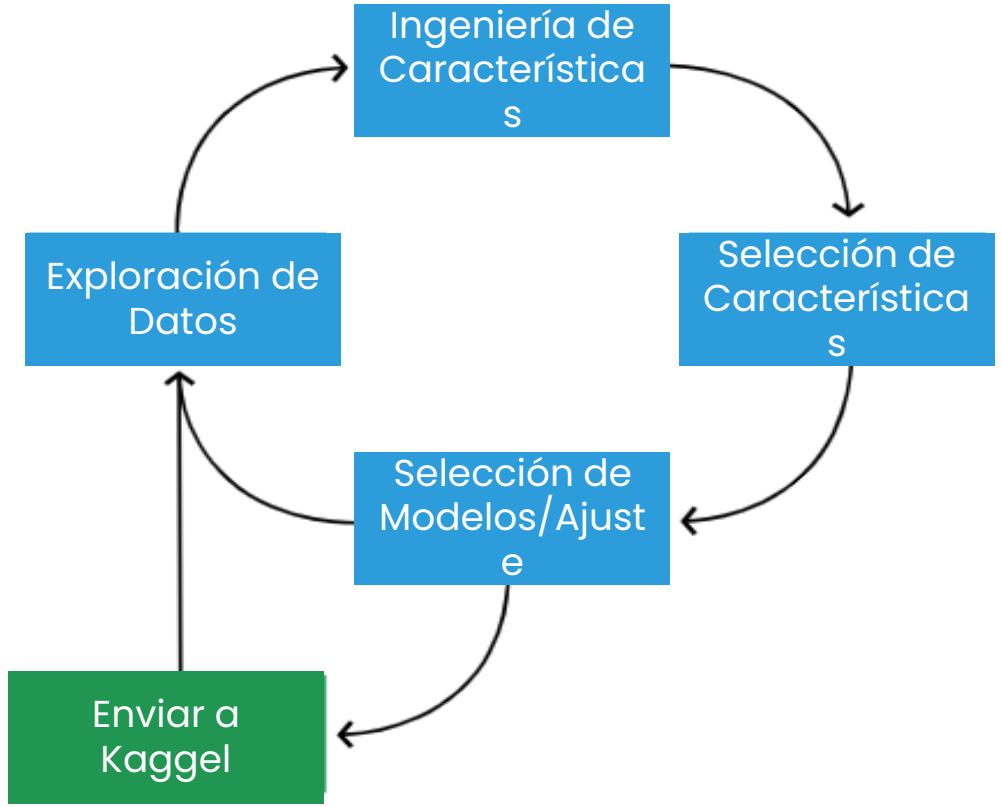
Instrucciones

- Crear una función, save_submission_file(), que:
  - Acepta un modelo entrenado y una lista de columnas como argumentos obligatorios, y un argumento opcional de nombre de archivo
  - Utiliza el modelo para realizar predicciones en el marco de datos de retención (holdout) utilizando las columnas especificadas
  - Transforma las predicciones en un marco de datos de presentación con las columnas PassengerID y Survived **especificadas por Kaggle**
  - Guarda ese marco de datos en un archivo CSV con un nombre de archivo predeterminado o el nombre de archivo especificado por el argumento opcional
- Recupera el modelo de mejor rendimiento de la variable devuelta por select_model()
- Utilice save_submission_file() para guardar un archivo CSV de predicciones
- Descargue ese archivo y envíelo a Kaggle

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

III.4.8 Próxima misión

En este proyecto guiado, hemos creado un flujo de trabajo reproducible para ayudarnos a iterar sobre las ideas y seguir mejorando la precisión de nuestras predicciones. También creamos funciones de ayuda que harán que la selección de características, la selección/ajuste del modelo y la creación de envíos sean mucho más fáciles mientras seguimos explorando los datos y creando nuevas características.



III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Te animamos a que sigas trabajando en esta competición de Kaggle. A continuación, se muestran algunas sugerencias de los próximos pasos:

- Continúe explorando los datos y creando nuevas funciones, siguiendo el flujo de trabajo y utilizando las funciones que creamos
- Lea más sobre Titanic y esta competición de Kaggle para obtener más ideas para nuevas características.
- Use algún algoritmo diferente en la función select_model() , como **maquinas de vectores de apoyo**, **descenso de gradiente estocástico** o modelos lineales **perceptron**
- Experimente con RandomizedSearchCV en vez de GridSearchCV para acelerar su función select_features()

Puede continuar trabajando en esta competición dentro de este ambiente de proyecto guiado y guardar archivos para enviar si lo desea aunque le recomendamos que configure su propio entorno de Python para que pueda trabajar en su propia computadora. Tenemos la **guía de instalación de Python** que lo guiará a través de cómo hacer esto.

III.4 Proyecto Guiado: Creación de un Flujo de Trabajo Kaggle

Por último, aunque la competición Titanic es estupenda para aprender a enfocar tu primera competición de Kaggle, recomendamos que no pases muchas horas centrado en intentar llegar a la cima de la tabla de clasificación. Con un conjunto de datos tan pequeño, hay un límite en cuanto a la calidad de tus predicciones, y tu tiempo estaría mejor invertido en competiciones más complejas.

Una vez que sientas que tienes una buena comprensión del flujo de trabajo de Kaggle, deberías mirar otras competiciones - una gran competición es la de **Precios de la Vivienda**. Tenemos un gran **tutorial para empezar con esta competición en nuestro blog.**

¿Tienes curiosidad por ver lo que otros estudiantes han hecho en este proyecto? **Dirígete a nuestra Comunidad para verlos.** Mientras estás allí, recuerda mostrar algo de amor y dar tus propios comentarios.

Y, por supuesto, te invitamos a que compartas tu propio proyecto y muestres tu duro trabajo. Dirígete a nuestra Comunidad para **compartir tu proyecto guiado terminado.**

IV. Conceptos de TensorFlow

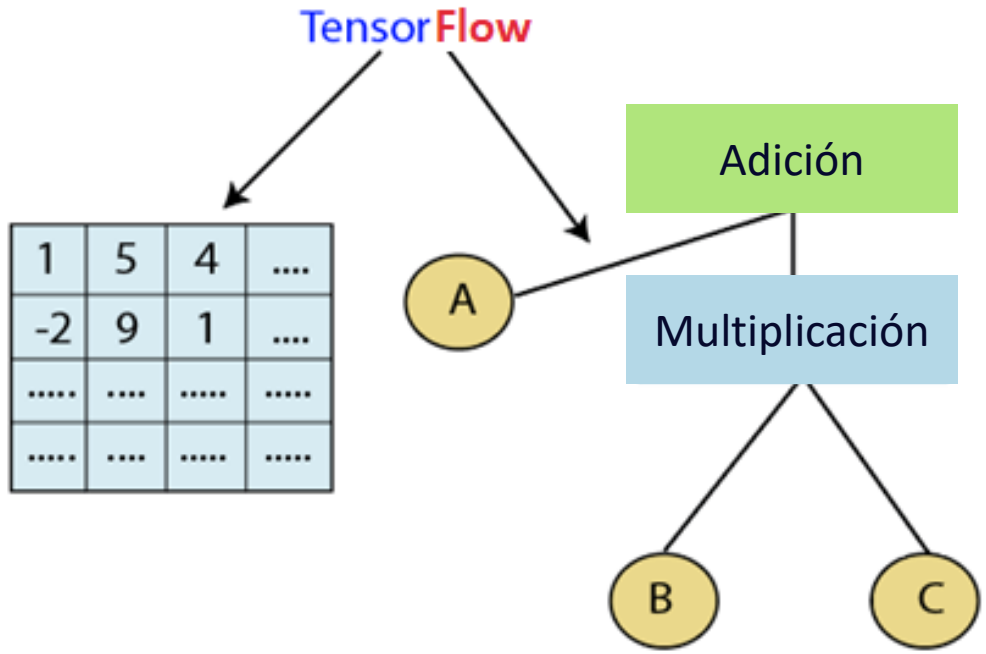
IV.1 Presentación de TensorFlow

TensorFlow es uno de los famosos marcos de aprendizaje profundo, desarrollado por el **equipo de Google**. Es una biblioteca de software libre y de código abierto y diseñado en lenguaje de programación **Python**, este tutorial está diseñado de tal manera que podemos implementar fácilmente el proyecto de aprendizaje profundo en TensorFlow de una manera fácil y eficiente.

La palabra TensorFlow se compone de dos palabras, es decir, Tensor y Flow:

1. **Tensor** es una matriz multidimensional
2. **Flow** se utiliza para definir el flujo de datos en la operación

TensorFlow se utiliza para definir el flujo de datos en la operación en una matriz multidimensional o Tensor.



IV.1 Presentación de TensorFlow

IV.1.1 Historia de TensorFlow

Hace ya muchos años que el aprendizaje profundo empezó a superar a todos los demás algoritmos de aprendizaje automático cuando se le facilitan muchos datos. Google ha visto que podía utilizar estas redes neuronales profundas para mejorar sus servicios:

- El motor de búsqueda de Google
- Gmail
- Fotos

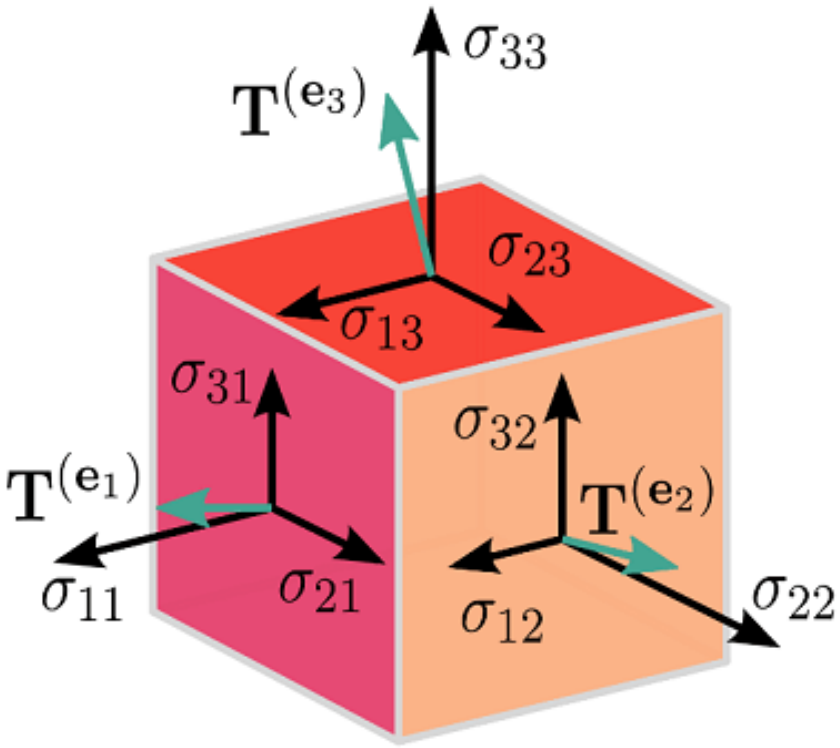
Construyen un marco llamado TensorFlow para permitir a los investigadores y desarrolladores trabajar juntos en un modelo de **IA**. Una vez aprobado y escalado, permite que mucha gente lo utilice. Fue lanzado por primera vez en 2015, mientras que la primera versión estable llegó en 2017. Es una plataforma de código abierto bajo la licencia Apache Open Source. Podemos utilizarla, modificarla y reorganizar la versión revisada de forma gratuita sin pagar nada a Google.

IV.1 Presentación de TensorFlow

IV.1.2 Componentes de TensorFlow

El nombre de TensorFlow se deriva de su núcleo, **"Tensor"**. Un tensor es un vector o una matriz de n dimensiones que representa todo tipo de datos. Todos los valores de un tensor tienen un tipo de datos similar con una forma conocida. La forma de los datos es la dimensión de la matriz o de un array.

Un tensor puede ser generado a partir de los datos de entrada o del resultado de un cálculo. En TensorFlow, todas las operaciones se realizan dentro de un grafo. El grupo es un conjunto de cálculos que se realizan sucesivamente. Cada operación se llama un nodo op están conectados.



TensorFlow hace uso de un marco gráfico. El gráfico reúne y describe todos los cálculos realizados durante el entrenamiento.

IV.1 Presentación de TensorFlow

IV .1.3 Ventajas

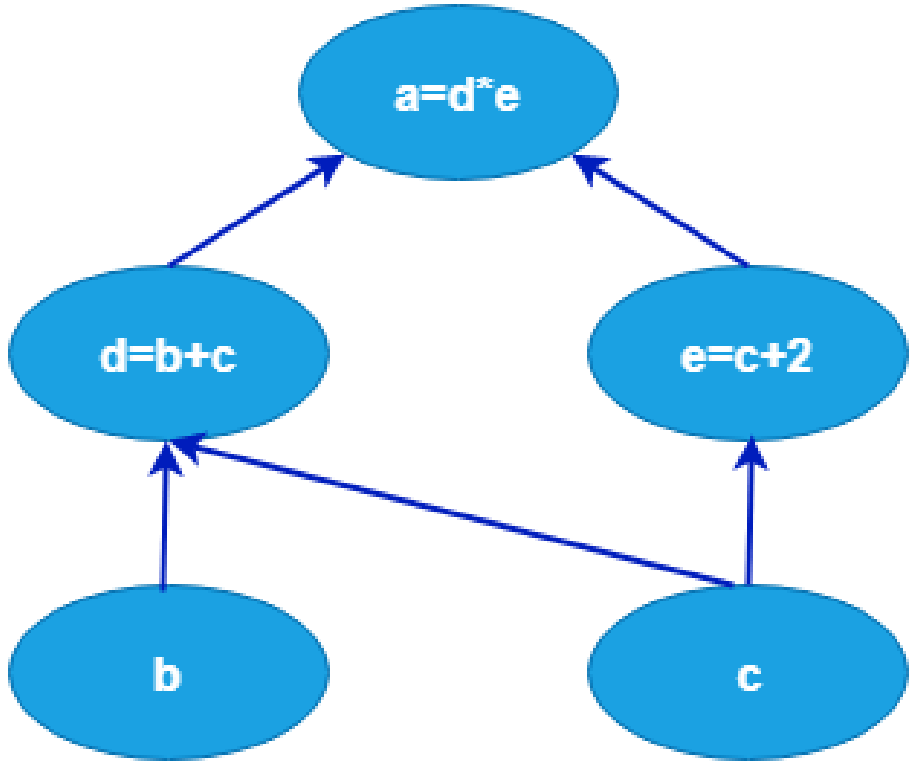
- Se ha fijado para que funcione en múltiples CPUs o GPUs y sistemas operativos móviles
- La portabilidad del gráfico permite conservar los cálculos para su uso actual o posterior. El grafo puede guardarse porque puede ejecutarse en el futuro
- Todo el cómputo en el gráfico se realiza conectando tensores

Considere la siguiente expresión a= (b+c)*(c+2). Podemos dividir las funciones en los siguientes componentes :
d=b+c
e=c+2
a=d*e

Ahora, podemos representar estas operaciones de forma gráfica a continuación:

IV.1 Presentación de TensorFlow

Una sesión puede ejecutar la operación desde el gráfico. Para alimentar el gráfico con el valor de un tensor, necesitamos abrir una sesión. Dentro de una sesión, debemos ejecutar un operador para crear una salida.



IV.1 Presentación de TensorFlow

IV.1.4 ¿Por qué es popular TensorFlow?

TensorFlow es la mejor biblioteca para todos porque es accesible para todos. La biblioteca TensorFlow integra diferentes **API** para crear una arquitectura de aprendizaje profundo a escala como **CNN (Red Neural Convolucional**) o **RNN (Red Neural Recurrente).**

TensorFlow se basa en la computación de grafos; puede permitir al desarrollador crear la construcción de la red neuronal con Tensorboard. Esta herramienta ayuda a depurar nuestro programa. Se ejecuta en la CPU (Unidad Central de Procesamiento) y en la GPU (Unidad de Procesamiento Gráfico).

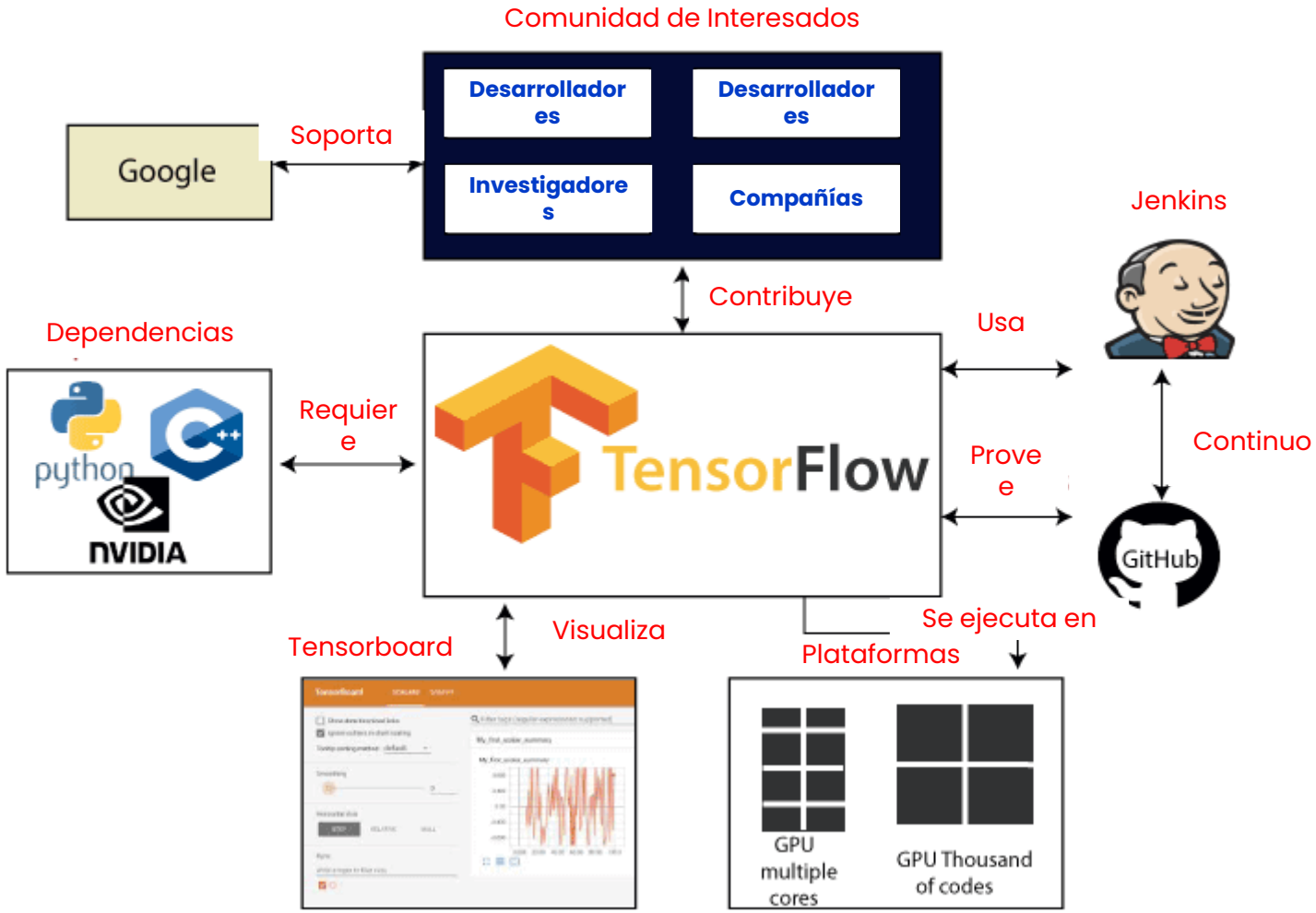
IV.1 Presentación de TensorFlow

IV .1.5 Casos de uso/aplicaciones de TensorFlow

TensorFlow proporciona funcionalidades y servicios sorprendentes en comparación con otros marcos de aprendizaje profundo populares. TensorFlow se utiliza para crear una red neuronal a gran escala con muchas capas.

Casos de uso de TensorFlow

IV.1 Presentación de TensorFlow



IV.1 Presentación de TensorFlow

Se utiliza principalmente para problemas de aprendizaje profundo o aprendizaje automático como la **Clasificación, Percepción, Comprensión, Descubrimiento, Predicción** y **Creación**.

Reconocimiento de voz/sonido

Las aplicaciones de reconocimiento de voz y sonido son los casos de uso más conocidos del aprendizaje profundo. Si las redes neuronales tienen una alimentación de datos de entrada adecuada, las redes neuronales son capaces de entender las señales de audio. Por ejemplo:

- **El reconocimiento de voz** se utiliza en el Internet de las Cosas, la automoción, la seguridad y la **UX/UI**
- **El análisis de sentimientos** se utiliza sobre todo en la gestión de las relaciones con los clientes **(CRM)**
- **La detección de fallos (ruido del motor)** se utiliza sobre todo en la automoción y la aviación
- **La búsqueda por voz** se utiliza sobre todo en la gestión de las relaciones con los clientes **(CRM)**

IV.1 Presentación de TensorFlow

Reconocimiento de imágenes

El reconocimiento de imágenes es la primera aplicación que popularizó el aprendizaje profundo y el aprendizaje automático. Las telecomunicaciones, las redes sociales y los fabricantes de teléfonos móviles utilizan principalmente el reconocimiento de imágenes. También se utiliza para el reconocimiento facial, la búsqueda de imágenes, la detección de movimiento, la visión artificial y la agrupación de fotos.

Por ejemplo, el reconocimiento de imágenes se utiliza para reconocer e identificar personas y objetos en las imágenes. El reconocimiento de imágenes se utiliza para entender el contexto y el contenido de cualquier imagen.

Para el reconocimiento de objetos, TensorFlow ayuda a clasificar e identificar objetos arbitrarios dentro de imágenes más grandes. También se utiliza en aplicaciones de ingeniería para identificar la forma con fines de modelado (reconstrucción **3D** a partir de una imagen **2D**) y en Facebook para el etiquetado de fotos.

Por ejemplo, el aprendizaje profundo utiliza TensorFlow para analizar miles de fotos de gatos. Así, un algoritmo de aprendizaje profundo puede aprender a identificar a un gato porque este algoritmo se utiliza para encontrar características generales de objetos, animales o personas.

IV.1 Presentación de TensorFlow

Series temporales

El aprendizaje profundo utiliza algoritmos de series temporales para examinar los datos de las series temporales y extraer estadísticas significativas. Por ejemplo, ha utilizado las series temporales para predecir el mercado de valores.

La recomendación es el caso de uso más común para las series temporales. **Amazon, Google, Facebook y Netflix** están utilizando el aprendizaje profundo para la sugerencia. Así, el algoritmo de aprendizaje profundo se utiliza para analizar la actividad del cliente y compararla con la de millones de otros usuarios para determinar lo que al cliente le puede gustar comprar o ver.

Por ejemplo, se puede utilizar para recomendarnos programas de televisión o películas que le gustan a la gente basándose en programas de televisión o películas que ya hemos visto.

IV.1 Presentación de TensorFlow

Detección de vídeo
El algoritmo de aprendizaje profundo se utiliza para la detección de vídeo. Se utiliza para la detección de movimiento, la detección de amenazas en tiempo real en los juegos, la seguridad, los aeropuertos y el campo de la interfaz de usuario/UX.

Por ejemplo, la NASA está desarrollando una red de aprendizaje profundo para la agrupación de objetos de asteroides y la clasificación de órbitas. Así, puede clasificar y predecir NEOs (**Near Earth Objects**).

Aplicaciones basadas en texto
Las aplicaciones basadas en texto también son un algoritmo de aprendizaje profundo muy popular. El análisis sentimental, las redes sociales, la detección de amenazas y la detección de fraudes son ejemplos de aplicaciones basadas en texto.

Por ejemplo, Google Translate soporta más de 100 idiomas**.**

Algunas **empresas** que actualmente utilizan TensorFlow son Google, AirBnb, eBay, Intel, DropBox, Deep Mind, Airbus, CEVA, Snapchat, SAP, Uber, Twitter, Coca-Cola e IBM.

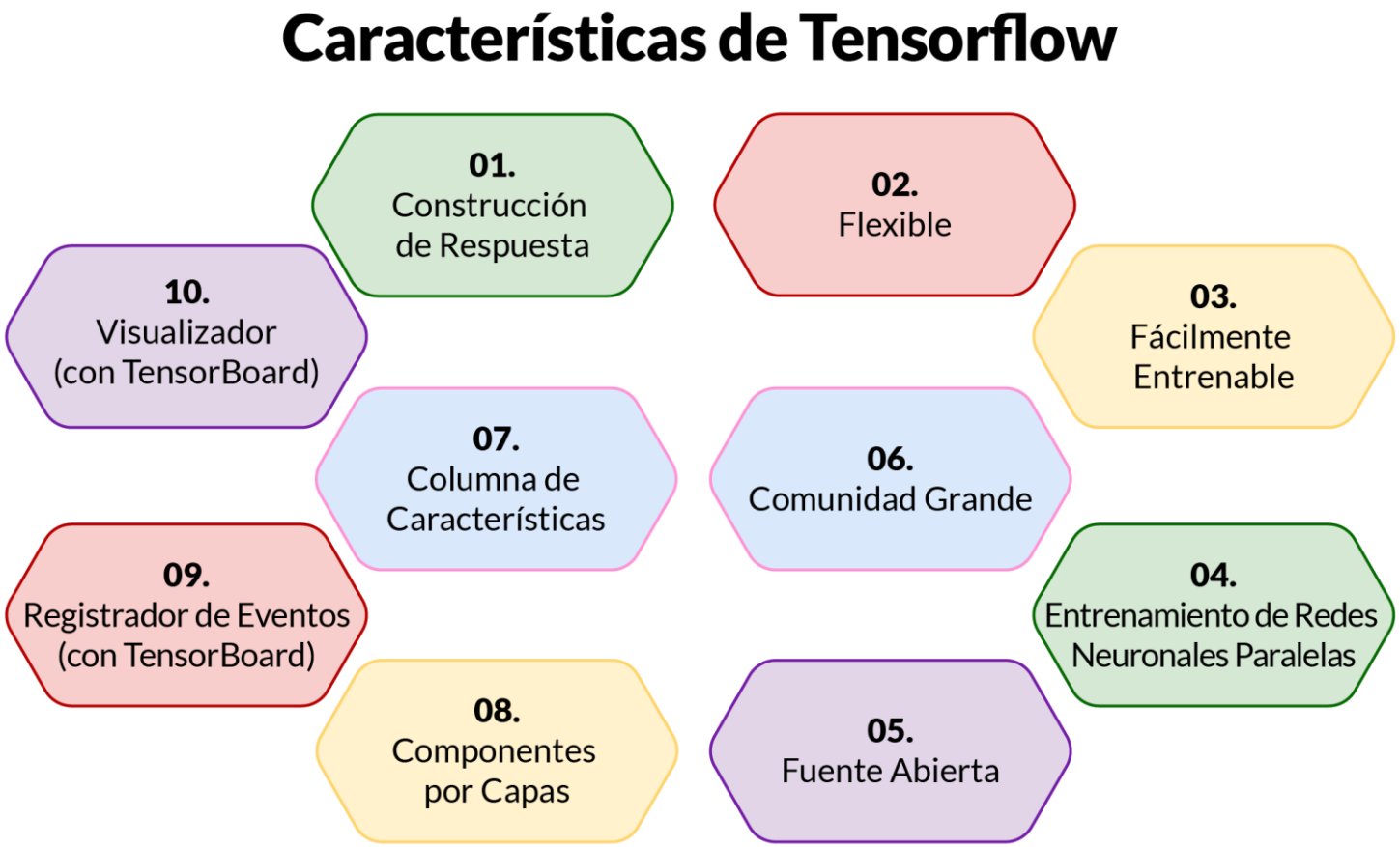
IV.1 Presentación de TensorFlow

IV 1.6 Características de TensorFlow

TensorFlow tiene una interfaz de programación interactiva multiplataforma que es escalable y fiable en comparación con otras bibliotecas de aprendizaje profundo que están disponibles.

Estas características de TensorFlow nos hablan de la popularidad de TensorFlow.

IV.1 Presentación de TensorFlow



IV.1 Presentación de TensorFlow

Construcción de respuesta

Podemos visualizar cada parte del gráfico, lo que no es una opción al utilizar Numpy o **SciKit**. Para desarrollar una aplicación de aprendizaje profundo, en primer lugar, hay dos o tres componentes que se requieren para crear una aplicación de aprendizaje profundo y necesitan un lenguaje de programación.

Flexible

Es una de las características esenciales de TensorFlow según su operatividad. Tiene modularidad y partes que queremos hacer independientes.

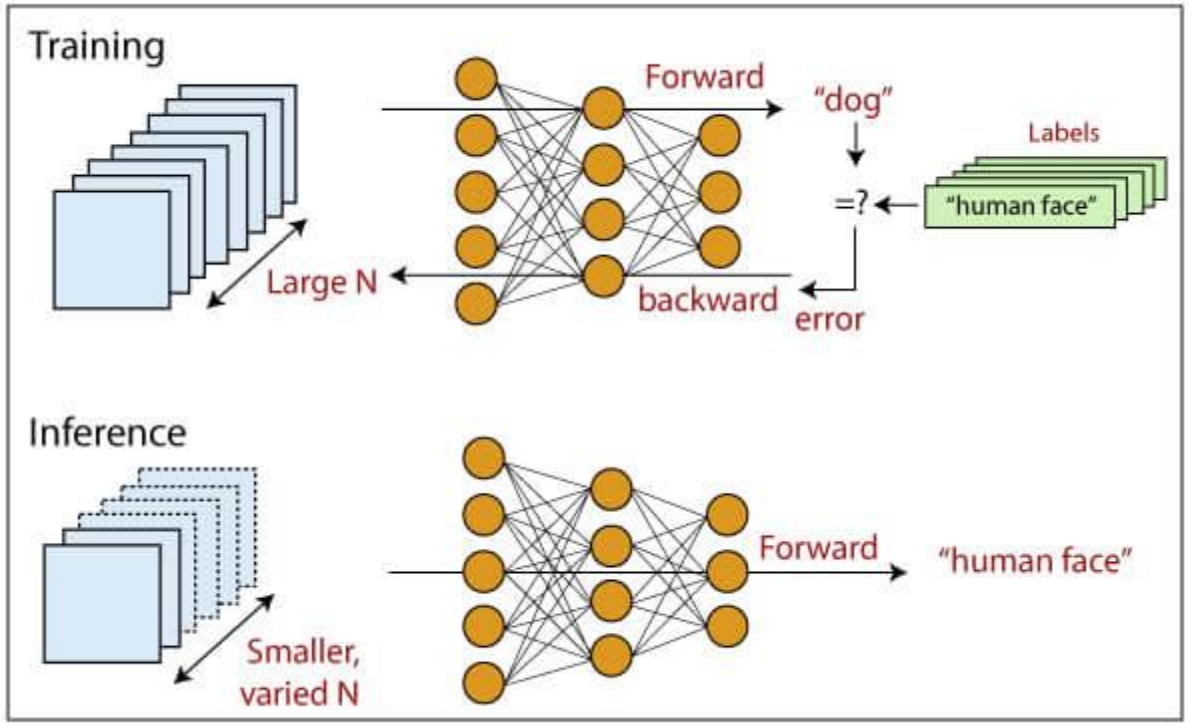
Fácilmente Entrenable

Es fácilmente entrenable en CPU y para GPU en computación distribuida.

IV.1 Presentación de TensorFlow

Entrenamiento de redes neuronales en paralelo

TensorFlow ofrece a la tubería en el sentido de que podemos entrenar múltiples redes neuronales y varias GPU, lo que hace que los modelos sean muy eficientes en sistemas a gran escala.



IV.1 Presentación de TensorFlow

Gran comunidad

Google lo ha desarrollado, y ya existe un gran equipo de ingenieros de software que trabajan continuamente en la mejora de la estabilidad.

Código abierto

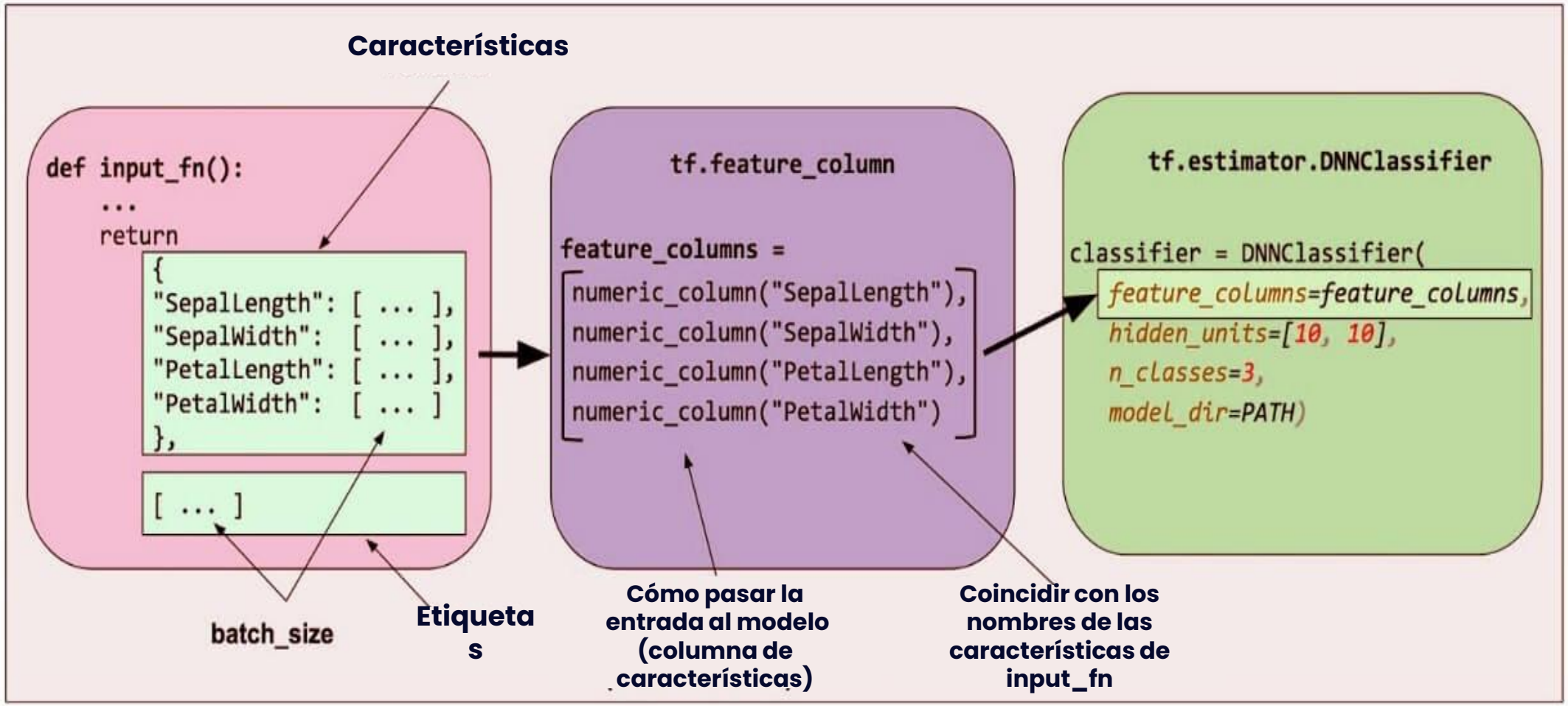
Lo mejor de la biblioteca de aprendizaje automático es que es de código abierto, por lo que cualquiera puede utilizarla siempre que tenga conexión a Internet. Así, la gente puede manipular la biblioteca y crear una fantástica variedad de productos útiles. Además, se ha convertido en otra comunidad de bricolaje que cuenta con un foro masivo para las personas que se inician en ella y las que tienen dificultades para utilizarla.

Columnas de características

TensorFlow tiene columnas de características que podrían ser consideradas como intermediarias entre los datos en bruto y los estimadores; en consecuencia, el puente de los datos de entrada con nuestro modelo.

La característica siguiente describe cómo se implementa la columna de características.

IV.1 Presentación de TensorFlow



IV.1 Presentación de TensorFlow

Disponibilidad de distribuciones estadísticas

Esta librería proporciona funciones de distribución como Bernoulli, Beta, Chi2, Uniforme, Gamma, que son esenciales, especialmente cuando se consideran enfoques probabilísticos como los modelos bayesianos.

Componentes en capas

TensorFlow produce operaciones en capas de peso y sesgo de la función como tf.contrib.layers y también proporciona la normalización de lotes, la capa de convolución, y la capa de abandono. Así que tf.contrib.layers.optimizers tiene optimizadores como Adagrad, SGD, Momentum que se utilizan a menudo para resolver problemas de optimización para el análisis numérico.

IV.1 Presentación de TensorFlow

Visualizador (Con TensorBoard)

Podemos inspeccionar una representación diferente de un modelo y hacer los cambios necesarios mientras lo depuramos con la ayuda de TensorBoard.

Registrador de eventos (con TensorBoard)

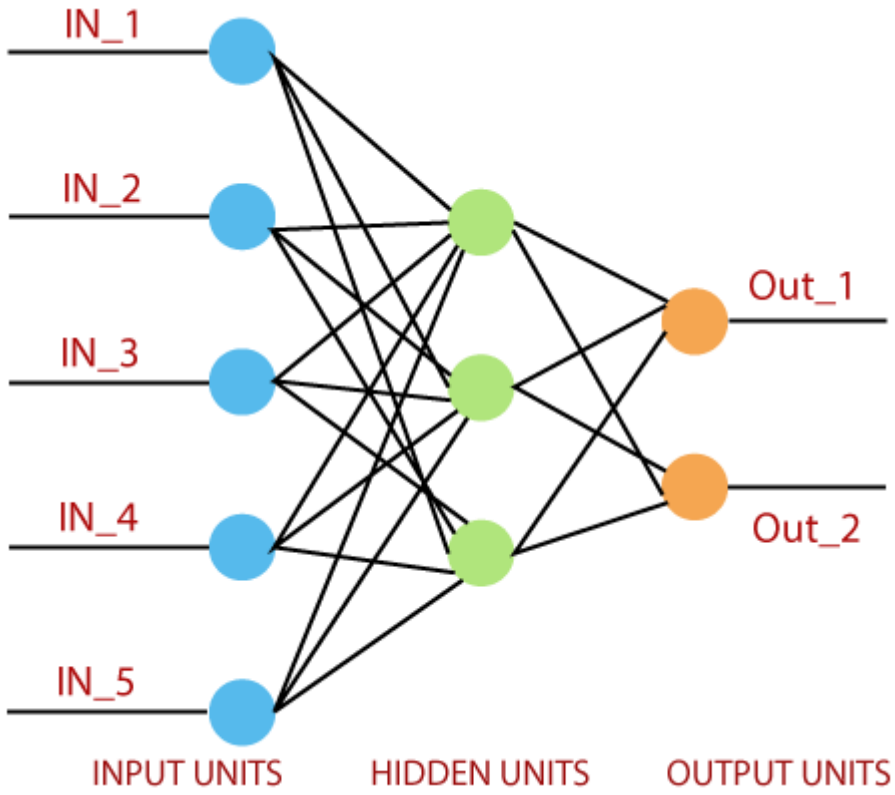
Es como en UNIX, donde usamos tail - f para monitorear la salida de las tareas en el cmd. Se comprueba, el registro de eventos y resúmenes de la gráfica y la producción con el TensorBoard.

IV.2 Fundamentos de TensorFlow

IV.2.1 Perceptrón de una capa en TensorFlow

El perceptrón es una unidad de procesamiento de cualquier red neuronal. Propuesto por Frank Rosenblatt por primera vez en 1958, es una neurona simple que sirve para clasificar su entrada en una o dos categorías. El perceptrón es un clasificador lineal y se utiliza en el aprendizaje supervisado. Ayuda a organizar los datos de entrada dados.

Un perceptrón es una unidad de red neuronal que realiza un cálculo preciso para detectar características en los datos de entrada. El perceptrón se utiliza principalmente para clasificar los datos en dos partes. Por lo tanto, también se conoce como clasificador binario lineal.

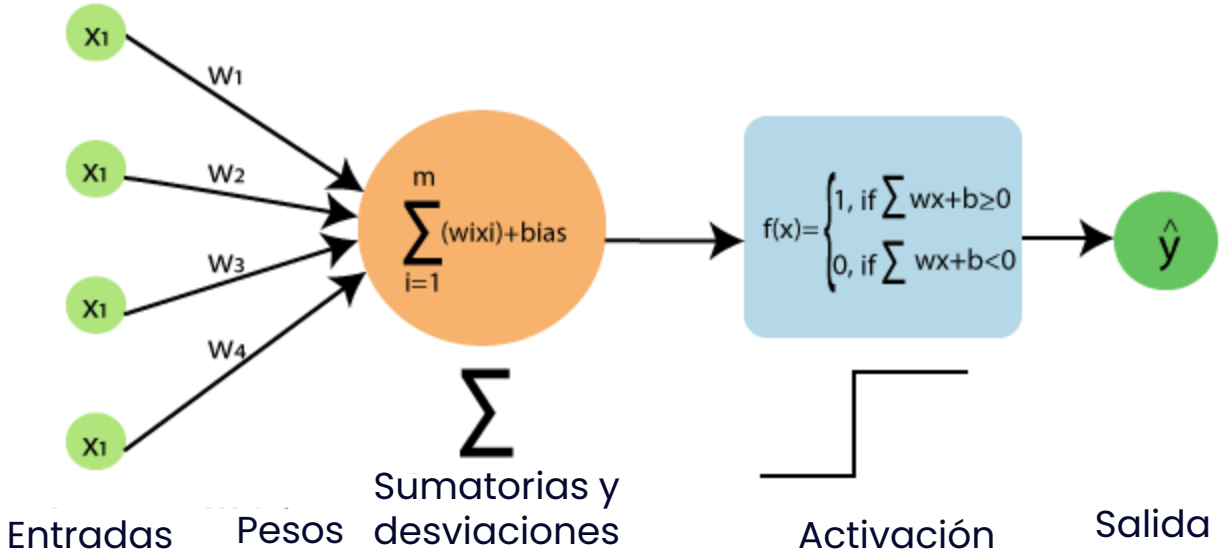


IV.2 Fundamentos de TensorFlow

El perceptrón utiliza la función de paso que devuelve +1 si la suma ponderada de su entrada es 0 y -1.

La función de activación se utiliza para mapear la entrada entre el valor requerido como (0, 1) o (-1, 1).

Una red neuronal normal tiene el siguiente aspecto:

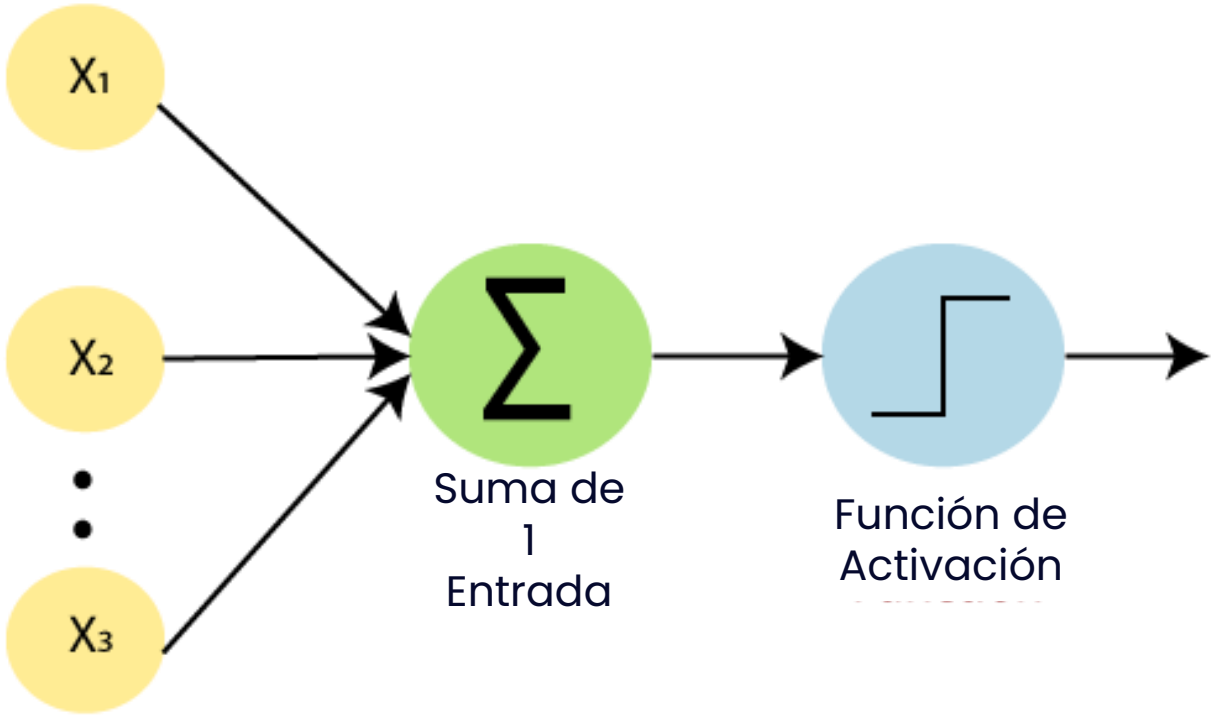


IV.2 Fundamentos de TensorFlow

El perceptrón consta de 4 partes

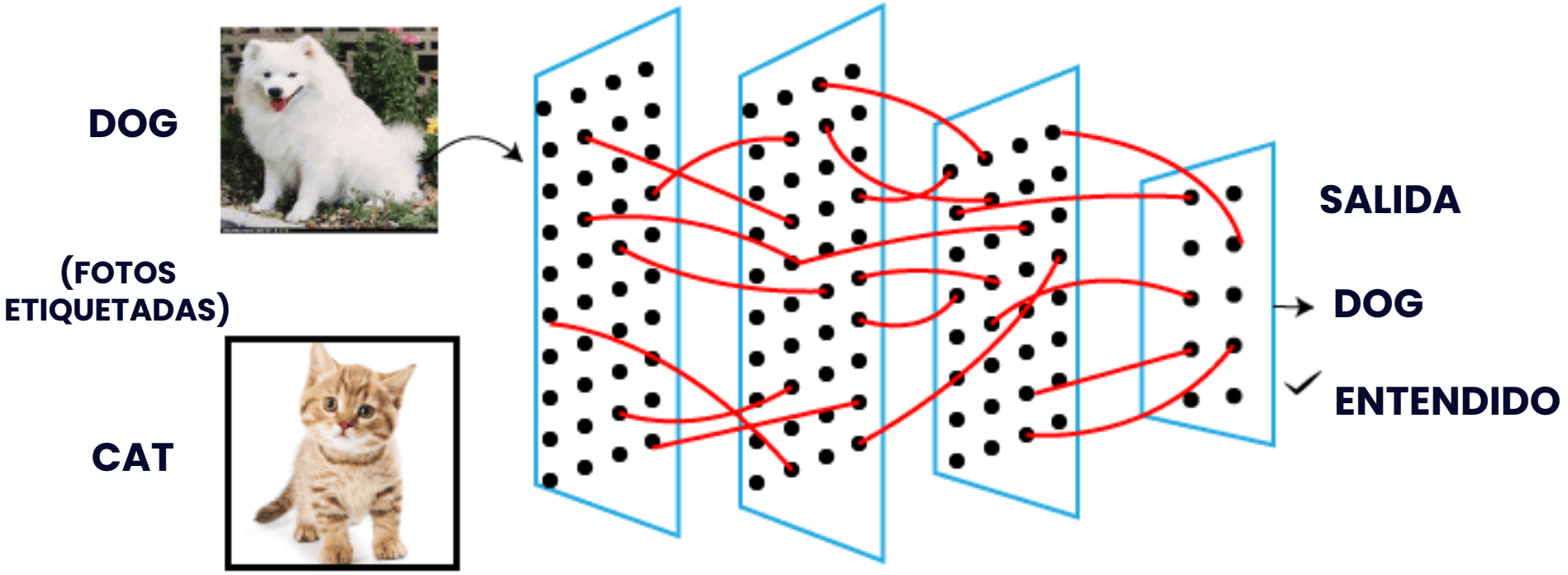
1. **Valor de entrada o una capa de entrada**: La capa de entrada del perceptrón está formada por neuronas artificiales de entrada y toma los datos iniciales en el sistema para su posterior procesamiento
2. **Pesos y sesgo**
   - **Peso:** Representa la dimensión o fuerza de la conexión entre unidades. Si el peso del nodo 1 al nodo 2 tiene una cantidad mayor, entonces la neurona 1 tiene una influencia más considerable sobre la neurona
   - **Sesgo:** Es lo mismo que el intercepto añadido en una ecuación lineal. Es un parámetro adicional cuya tarea es modificar la salida junto con la suma ponderada de la entrada a la otra neurona
3. **Suma neta:** Calcula la suma total
4. **Función de activación:** Una neurona puede ser activada o no, está determinada por una función de activación La función de activación calcula una suma ponderada y además añade un sesgo con ella para dar el resultado

IV.2 Fundamentos de TensorFlow



IV.2 Fundamentos de TensorFlow

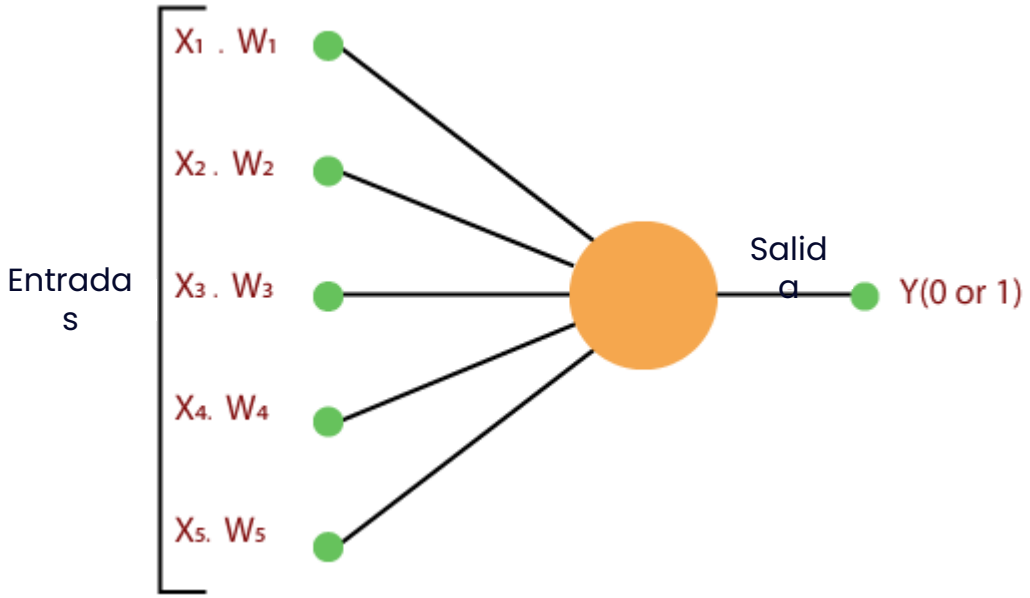
Una red neuronal estándar tiene el aspecto del siguiente diagrama.



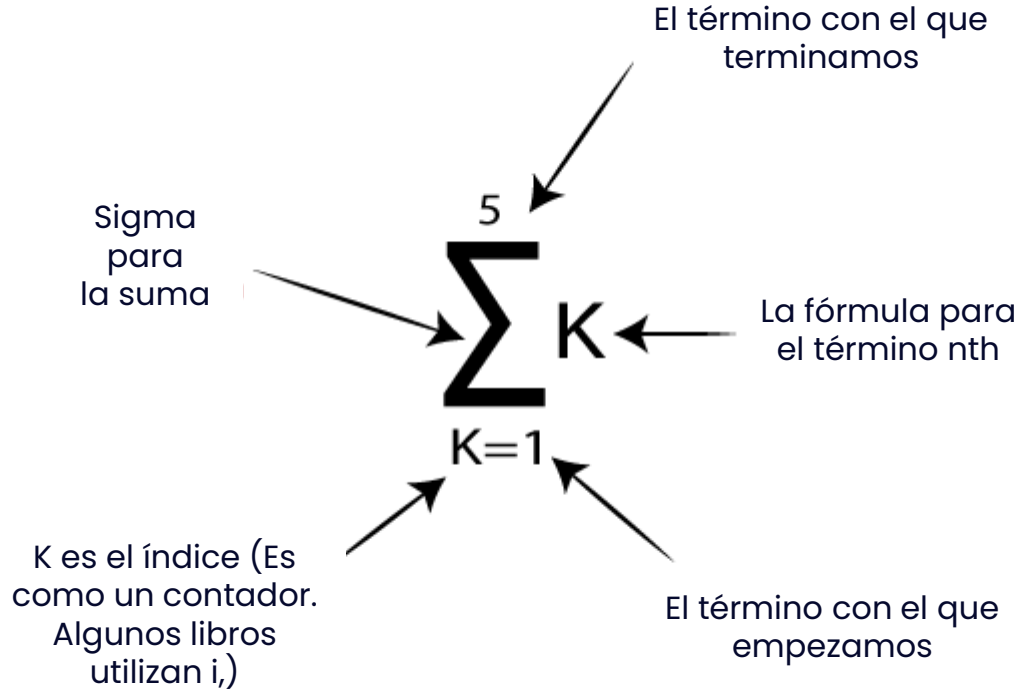
IV.2 Fundamentos de TensorFlow

¿Cómo funciona?
El perceptrón funciona según estos sencillos pasos que se indican a continuación:

A. En el primer paso, todas las entradas x se multiplican con sus pesos w

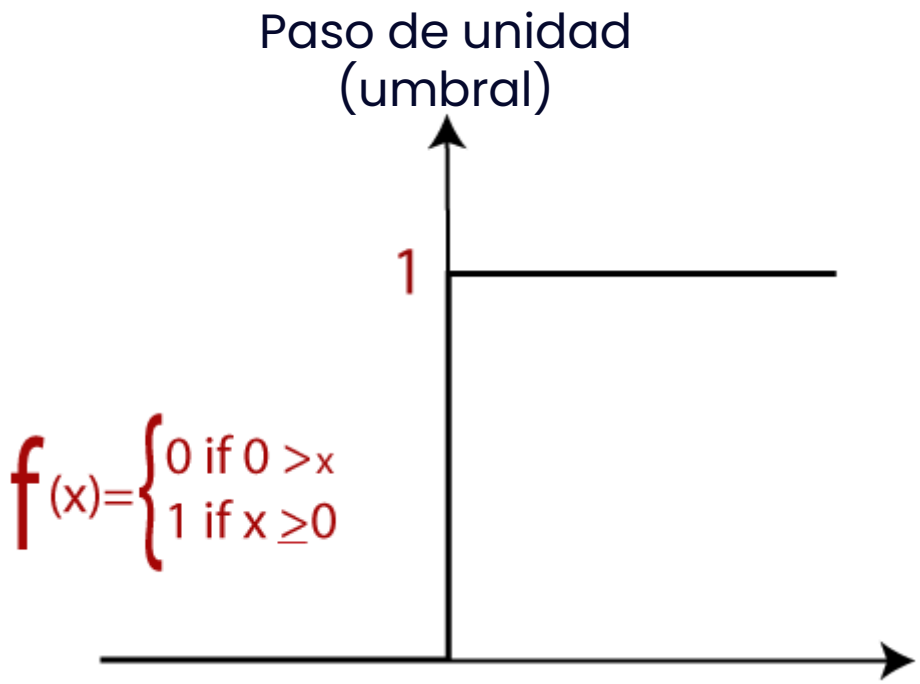


B. En este paso, se suman todos los valores incrementados y se les llama **Suma Ponderada**



IV.2 Fundamentos de TensorFlow

C. En nuestro último paso, aplicar la suma ponderada a una Función de Activación correcta. Por ejemplo: Una función de activación de pasos unitarios.



IV.2 Fundamentos de TensorFlow

Existen dos tipos de arquitectura. Estos tipos se centran en la funcionalidad de las redes neuronales artificiales como sigue:

- Perceptrón de una capa
- Perceptrón multicapa

El perceptrón de una capa fue el primer modelo de red neuronal, propuesto en 1958 por Frank Rosenbluth. Es uno de los primeros modelos de aprendizaje. Nuestro objetivo es encontrar una función de decisión lineal medida por el vector de pesos w y el parámetro de sesgo b.

Para entender la capa del perceptrón, es necesario comprender las redes neuronales artificiales (RNA). La red neuronal artificial (RNA) es un sistema de procesamiento de información cuyo mecanismo se inspira en la funcionalidad de los circuitos neuronales biológicos. Una red neuronal artificial se compone de varias unidades de procesamiento que están interconectadas.

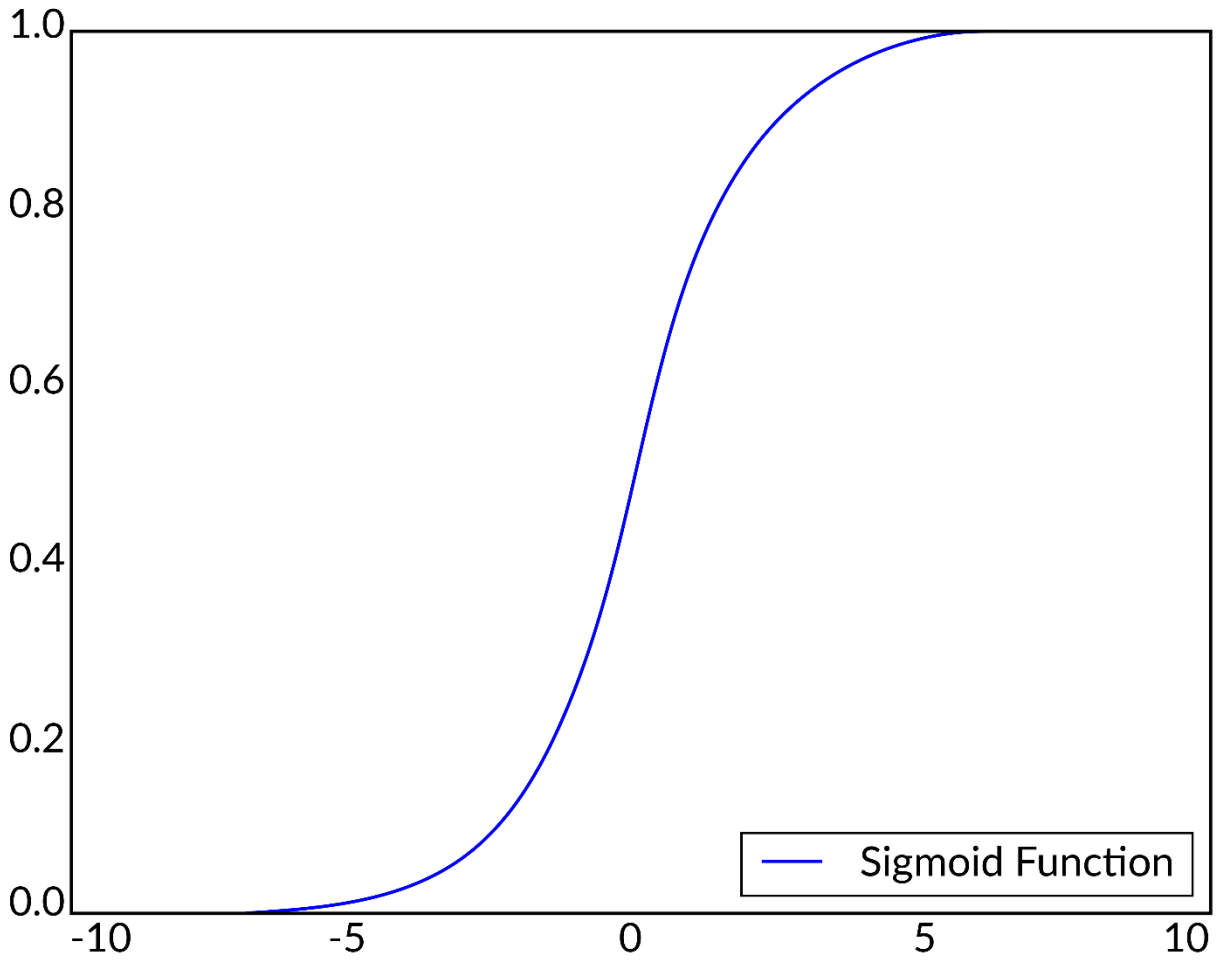
IV.2 Fundamentos de TensorFlow

Esta es la primera propuesta cuando se construye el modelo neuronal. El contenido de la memoria local de la neurona contiene un vector de pesos.

El perceptrón monovectorial se calcula calculando la suma del vector de entrada multiplicada por el elemento correspondiente del vector, aumentando cada vez la cantidad del componente correspondiente del vector por el peso. El valor que aparece en la salida es la entrada de una función de activación.

Centrémonos en la implementación de un perceptrón de una sola capa para un problema de clasificación de imágenes utilizando TensorFlow. El mejor ejemplo de dibujar un perceptrón de una capa es a través de la representación de la **"regresión logística".**

IV.2 Fundamentos de TensorFlow



IV.2 Fundamentos de TensorFlow

Ahora, tenemos que hacer los siguientes pasos necesarios de entrenamiento de regresión logística:

- Los pesos se inicializan con los valores aleatorios al inicio de cada entrenamiento
- Para cada elemento del conjunto de entrenamiento, se calcula el error con la diferencia entre la salida deseada y la salida real. El error calculado se utiliza para ajustar el peso
- El proceso se repite hasta que el error cometido en todo el conjunto de entrenamiento sea inferior al límite especificado hasta que se haya alcanzado el número máximo de iteraciones

IV.2 Fundamentos de TensorFlow

Código completo del perceptrón de una capa

```
import tensorflow as tf
import matplotlib.pyplot as plt
# Parameters
learning_rate = 0.01
training_epochs = 25
batch_size = 100
display_step = 1

# tf Graph Input
x = tf.placeholder("float", [none, 784]) # MNIST data image of shape 28*28 = 784
y = tf.placeholder("float", [none, 10]) # 0-9 digits recognition => 10 classes
# Create model
# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
# Constructing the model
activation=tf.nn.softmaxx(tf.matmul (x, W)+b) # Softmax
 of function
# Minimizing error using cross entropy
cross_entropy = y*tf.log(activation)
```

```
cost = tf.reduce_mean\ (-tf.reduce_sum\ (cross_entropy, reduction_indice = 1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
#Plot settings
avg_set = []
epoch_set = []
# Initializing the variables where init = tf.initialize_all_variables()
# Launching the graph
with tf.Session() as sess:
  sess.run(init)

# Training of the cycle in  the dataset
  for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_example/batch_size)
```


IV.2 Fundamentos de TensorFlow

```
# Creating loops at all the batches in the code
    for i in range(total_batch):
batch_xs, batch_ys = mnist.train.next_batch(batch_size)
      # Fitting the training by the batch data sess.run(optimizr,  feed_dict = {
x: batch_xs, y: batch_ys})
 # Compute all the average of loss avg_cost += sess.run(cost, \ feed_dict = {
x: batch_xs, \ y: batch_ys}) //total batch
    # Display the logs at each epoch steps
    if epoch % display_step==0:
    print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format (avg_cost))
        avg_set.append(avg_cost) epoch_set.append(epoch+1)
  print ("Training phase finished")

  plt.plot(epoch_set,avg_set, 'o', label = 'Logistics Regression Training')
  plt.ylabel('cost')
  plt.xlabel('epoch')
  plt.legend()
  plt.show()
```

```
# Test the model
  correct_prediction = tf.equal (tf.argmax (activation, 1),tf.argmax(y,1))

# Calculating the accuracy of dataset
accuracy = tf.reduce_mean(tf.cast (correct_prediction, "float")) print
("Model accuracy:", accuracy.eval({x:mnist.test.images, y: mnist.test.labels}))
```

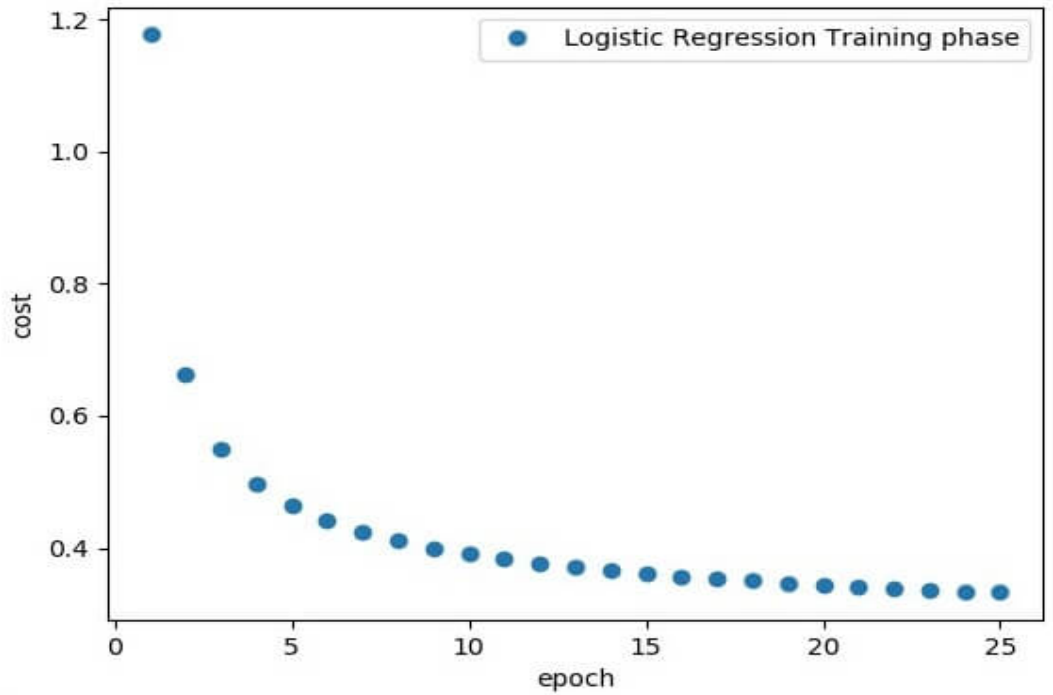

IV.2 Fundamentos de TensorFlow

El Resultado del Código:

```
Use `tf.global_variables_initializer` instead.
2018-07-09 11:41:19.926820: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was
 not compiled to use: AVX2
Epoch: 0001 cost= 1.176560601
Epoch: 0002 cost= 0.662504855
Epoch: 0003 cost= 0.550647454
Epoch: 0004 cost= 0.496803975
Epoch: 0005 cost= 0.463813413
Epoch: 0006 cost= 0.440974006
Epoch: 0007 cost= 0.421977672
Epoch: 0008 cost= 0.410615707
Epoch: 0009 cost= 0.399920001
Epoch: 0010 cost= 0.390970191
Epoch: 0011 cost= 0.383367628
Epoch: 0012 cost= 0.376823489
Epoch: 0013 cost= 0.371032368
Epoch: 0014 cost= 0.365939891
Epoch: 0015 cost= 0.361388407
Epoch: 0016 cost= 0.357227336
Epoch: 0017 cost= 0.353601005
Epoch: 0018 cost= 0.350117204
Epoch: 0019 cost= 0.347027825
Epoch: 0020 cost= 0.344160418
Epoch: 0021 cost= 0.341485278
Epoch: 0022 cost= 0.339003812
Epoch: 0023 cost= 0.336675840
Epoch: 0024 cost= 0.334456453
Epoch: 0025 cost= 0.332455397
Training phase finished
Model accuracy: 0.9135
```

IV.2 Fundamentos de TensorFlow

La regresión logística se considera un análisis predictivo. La regresión logística se utiliza principalmente para describir datos y para explicar la relación entre la variable binaria dependiente y una o varias variables nominales o independientes.



IV.2.2 Perceptrón de capa oculta en TensorFlow

Una capa oculta es una red neuronal artificial que se encuentra entre las **capas de entrada** y las **de salida**. Las neuronas artificiales reciben un conjunto de entradas ponderadas y producen una salida mediante una función de activación. Es una parte de casi y neural en la que los ingenieros simulan los tipos de actividad que se dan en el cerebro humano.

La red neuronal oculta se configura en algunas técnicas. En muchos casos, las entradas ponderadas se asignan al azar. En otros, se ajustan y calibran mediante un proceso llamado **retro propagación.**

La neurona artificial de la capa oculta del perceptrón funciona como una neurona biológica en el cerebro: toma sus señales de entrada probabilísticas y trabaja con ellas. Y las convierte en una salida correspondiente al axón de la neurona biológica.

IV.2 Fundamentos de TensorFlow

Las capas posteriores a la de entrada se denominan ocultas porque resuelven directamente la entrada. La estructura de red más sencilla es tener una sola neurona en la capa oculta que emite directamente el valor.

El aprendizaje profundo puede referirse a tener muchas capas ocultas en nuestra red neuronal. Son profundas porque históricamente han sido inimaginablemente lentas de entrenar, pero pueden tardar segundos o minutos en prepararse utilizando técnicas y hardware modernos.

Una sola capa oculta construirá una red sencilla.

El código de las capas ocultas del perceptrón se muestra a continuación:

IV.2 Fundamentos de TensorFlow

```
#Importing the essential modules in the hidden layer
import tensorflow as tf
import numpy as np
import mat
plotlib.pyplot as plt
import math, random



np.random.seed(1000)
function_to_learn = lambda x: np.cos(x) + 0.1*np.random.randn(*x.shape)
layer_1_neurons = 10
NUM_points = 1000

#Train the parameters of hidden layer
batch_size = 100
NUM_EPOCHS = 1500

all_x = np.float32(np.random.uniform(-2*math.pi, 2*math.pi, (1, NUM_points))).T
```

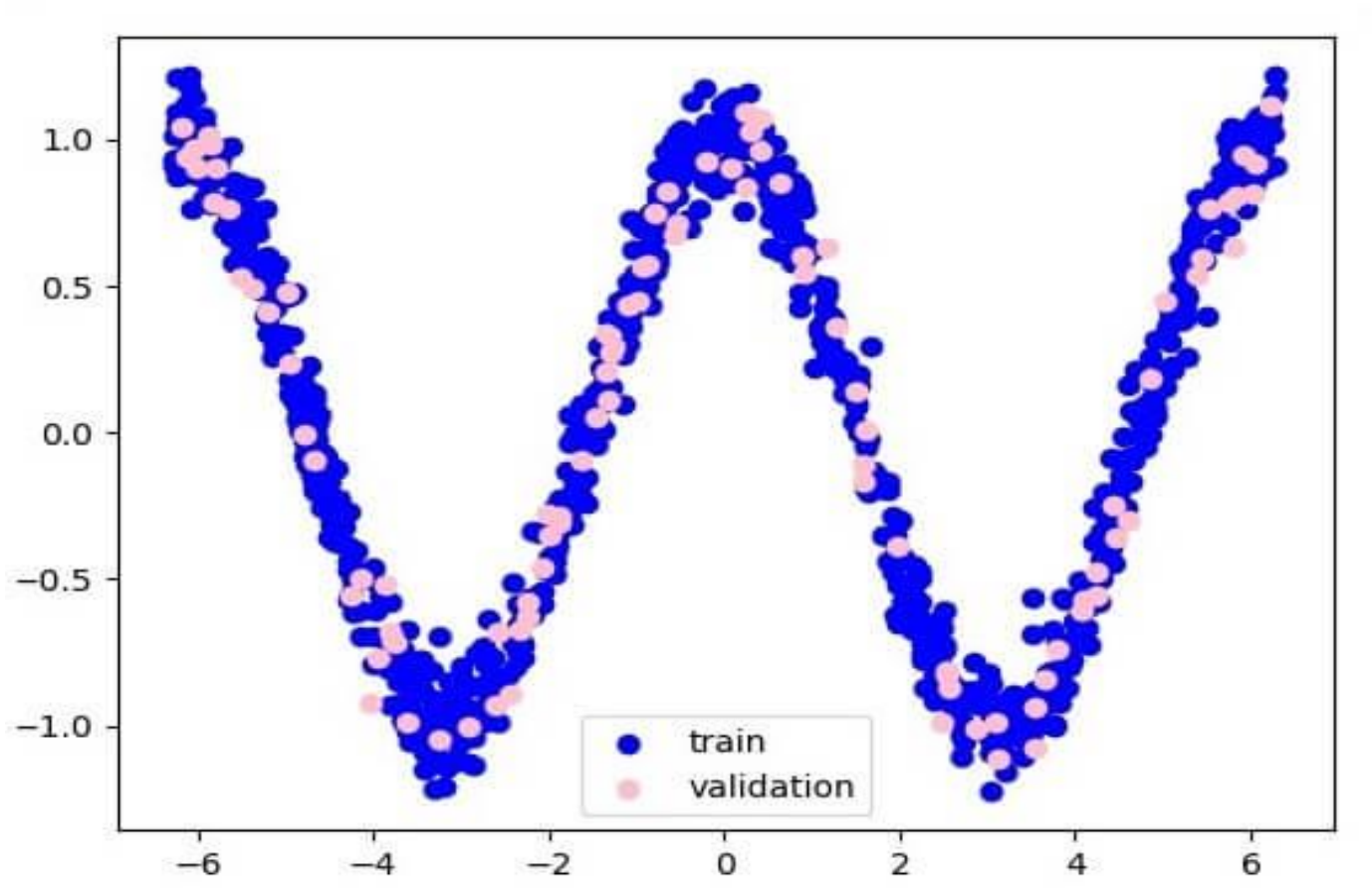
```
  np.random.shuffle(all_x)

train_size = int(900)
#Train the first 700 points in the set x_training = all_x[:train_size]
y_training = function_to_learn(x_training)

#Training the last 300 points in the given set x_validation = all_x[train_size:]
y_validation = function_to_learn(x_validation)

plt.figure(1)
plt.scatter(x_training, y_training, c = 'blue', label = 'train')
plt.scatter(x_validation, y_validation, c = 'pink', label = 'validation')
plt.legend()
plt.show()

X = tf.placeholder(tf.float32, [None, 1], name = "X")
Y = tf.placeholder(tf.float32, [None, 1], name = "Y")

#first layer
#Number of neurons = 10
```

IV.2 Fundamentos de TensorFlow

```
w_h = tf.Variable(
   tf.random_uniform([1, layer_1_neurons],\ minval = -1, maxval = 1, dtype = tf.float32))
b_h = tf.Variable(tf.zeros([1, layer_1_neurons], dtype = tf.float32))
h = tf.nn.sigmoid(tf.matmul(X, w_h) + b_h)

#output layer
#Number of neurons = 10
w_o = tf.Variable(
   tf.random_uniform([layer_1_neurons, 1],\ minval = -1, maxval = 1, dtype = tf.float32))
b_o = tf.Variable(tf.zeros([1, 1], dtype = tf.float32))

#building the model
model = tf.matmul(h, w_o) + b_o

#minimize the cost function (model - Y)
train_op = tf.train.AdamOptimizer().minimize(tf.nn.l2_loss(model - Y))
```

```
#Starting the Learning phase
sess = tf.Session() sess.run(tf.initialize_all_variables())

errors = []
for i in range(NUM_EPOCHS):
   for start, end in zip(range(0, len(x_training), batch_size),\
      range(batch_size, len(x_training), batch_size)):
      sess.run(train_op, feed_dict = {X: x_training[start:end],\ Y: y_training[start:end]})
   cost = sess.run(tf.nn.l2_loss(model - y_validation),\ feed_dict = {X:x_validation})
   errors.append(cost)

   if i%100 == 0:
      print("epoch %d, cost = %g" % (i, cost))

plt.plot(errors,label='MLP Function Approximation') plt.xlabel('epochs')
plt.ylabel('cost')
plt.legend()
plt.show()
```


IV.2 Fundamentos de TensorFlow



IV.2.3 Redes neuronales artificiales en TensorFlow

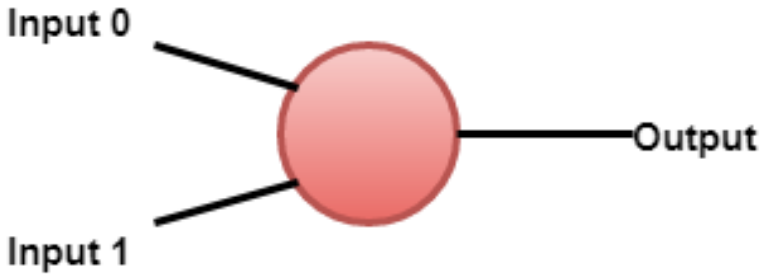
Las redes neuronales o **redes neuronales artificiales (RNA)** se modelan igual que el cerebro humano. El cerebro humano tiene una mente para pensar y analizar cualquier tarea en una situación concreta.

Pero, ¿cómo puede una máquina pensar así? Para ello se diseñó un cerebro artificial que se conoce como red neuronal. La red neuronal está formada por muchos **perceptrones**.

El perceptrón es una red neuronal de una sola capa. Es un clasificador binario y forma parte del aprendizaje supervisado. Un modelo simple de la neurona biológica en una red neuronal artificial se conoce como perceptrón.

La neurona artificial tiene entrada y salida.

IV.2 Fundamentos de TensorFlow



Representación matemática del modelo de perceptrón.

$$\sum_{i=0}^{n} w_i x_i + b$$

El cerebro humano tiene neuronas para transmitir información, y la red neuronal tiene nodos para realizar la misma tarea. Los nodos son las funciones matemáticas. Una red neuronal se basa en la estructura y función de las redes neuronales biológicas.

Una red neuronal cambia o aprende por sí misma en función de la entrada y la salida. Los flujos de información a través del sistema afectan a la estructura de la red neuronal artificial debido a su aprendizaje y mejora de la propiedad.

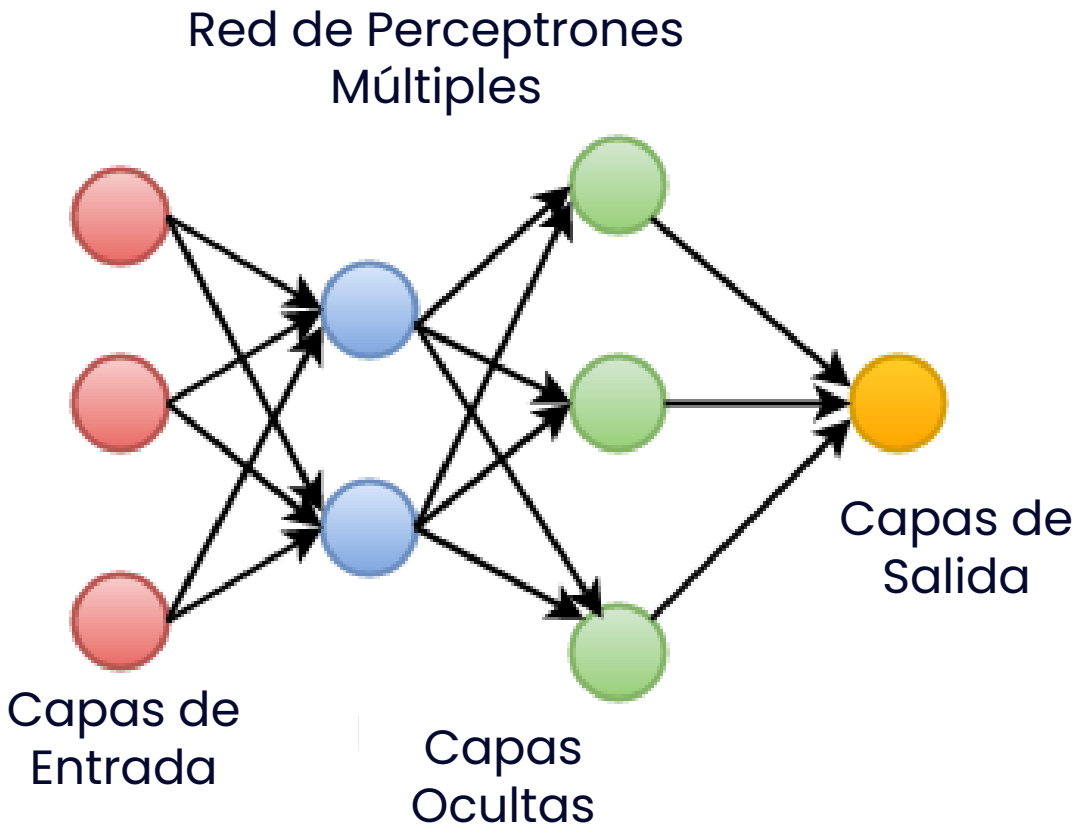
IV.2 Fundamentos de TensorFlow

Una **Red Neuronal** también se define como: *Un sistema informático formado por varios elementos de procesamiento simples y altamente interconectados, que procesan la información mediante su respuesta de estado dinámica a las entradas externas.*

Una red neuronal puede estar formada por múltiples perceptrones. En ella hay tres capas:

- **Capa de entrada:** Las capas de entrada son el valor real de los datos
- **Capa oculta:** Las capas ocultas están entre las capas de entrada y de salida, donde tres o más capas son una red profunda
- **Capa de salida:** Es la estimación final de la salida

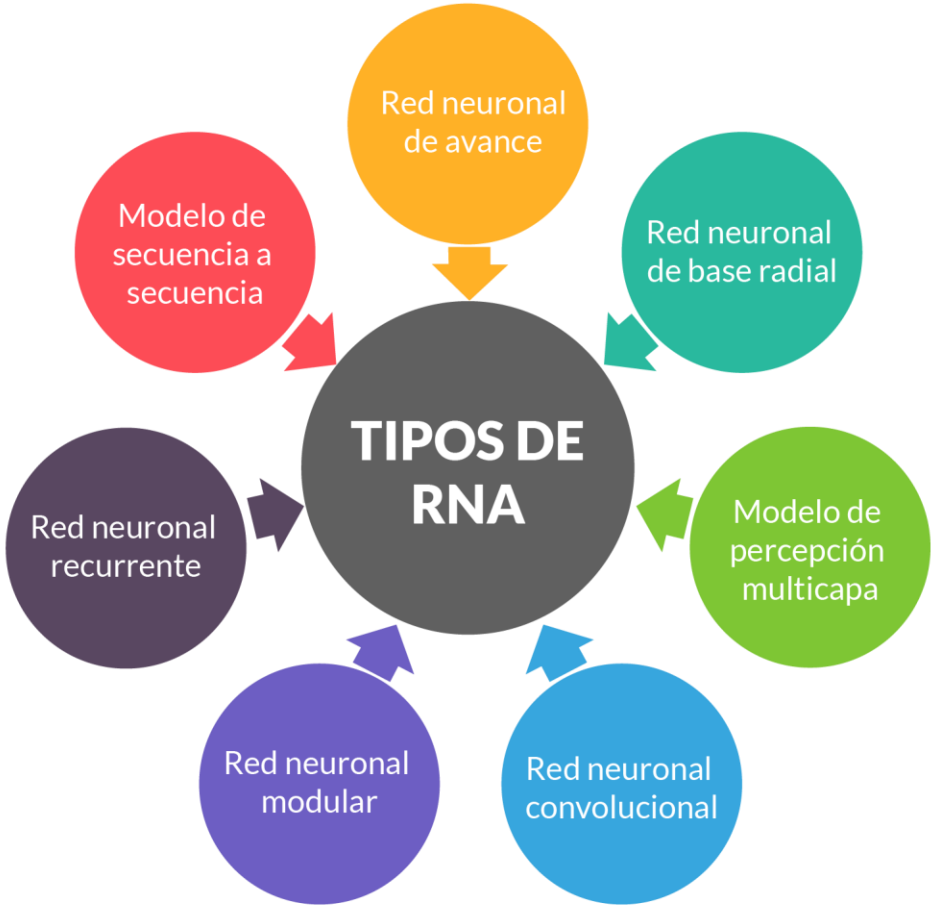
IV.2 Fundamentos de TensorFlow



IV.2 Fundamentos de TensorFlow

IV.2.4 Tipos de redes neuronales artificiales

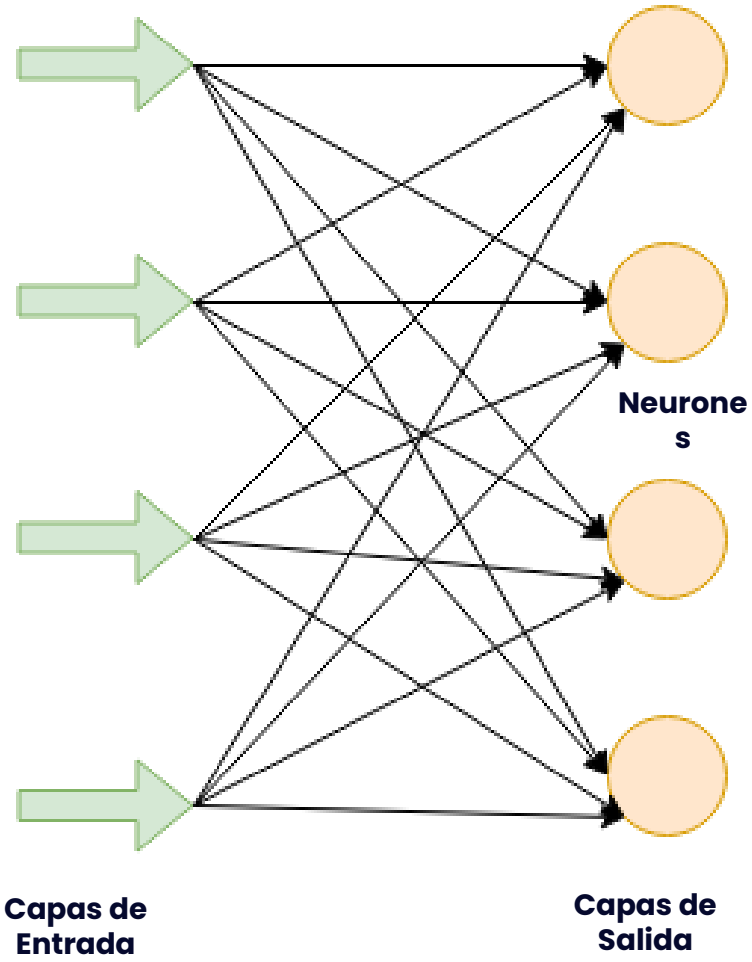
Las redes neuronales funcionan igual que el sistema nervioso humano. Existen varios tipos de redes neuronales. La implementación de estas redes se basa en el conjunto de parámetros y operaciones matemáticas que se requieren para determinar la salida.



IV.2 Fundamentos de TensorFlow

IV.2.5 Red Neuronal de Avance (Neurona Artificial)

La **FNN** es la forma más pura de RNA en la que la entrada y los datos viajan en una sola dirección. Los datos fluyen en una sola dirección hacia adelante; por eso se conoce como **Red Neuronal de Avance**. Los datos pasan por los nodos de entrada y salen por los de salida. Los nodos no están conectados cíclicamente. No necesita tener una capa oculta. En la FNN, no es necesario que haya varias capas. También puede tener una sola capa.

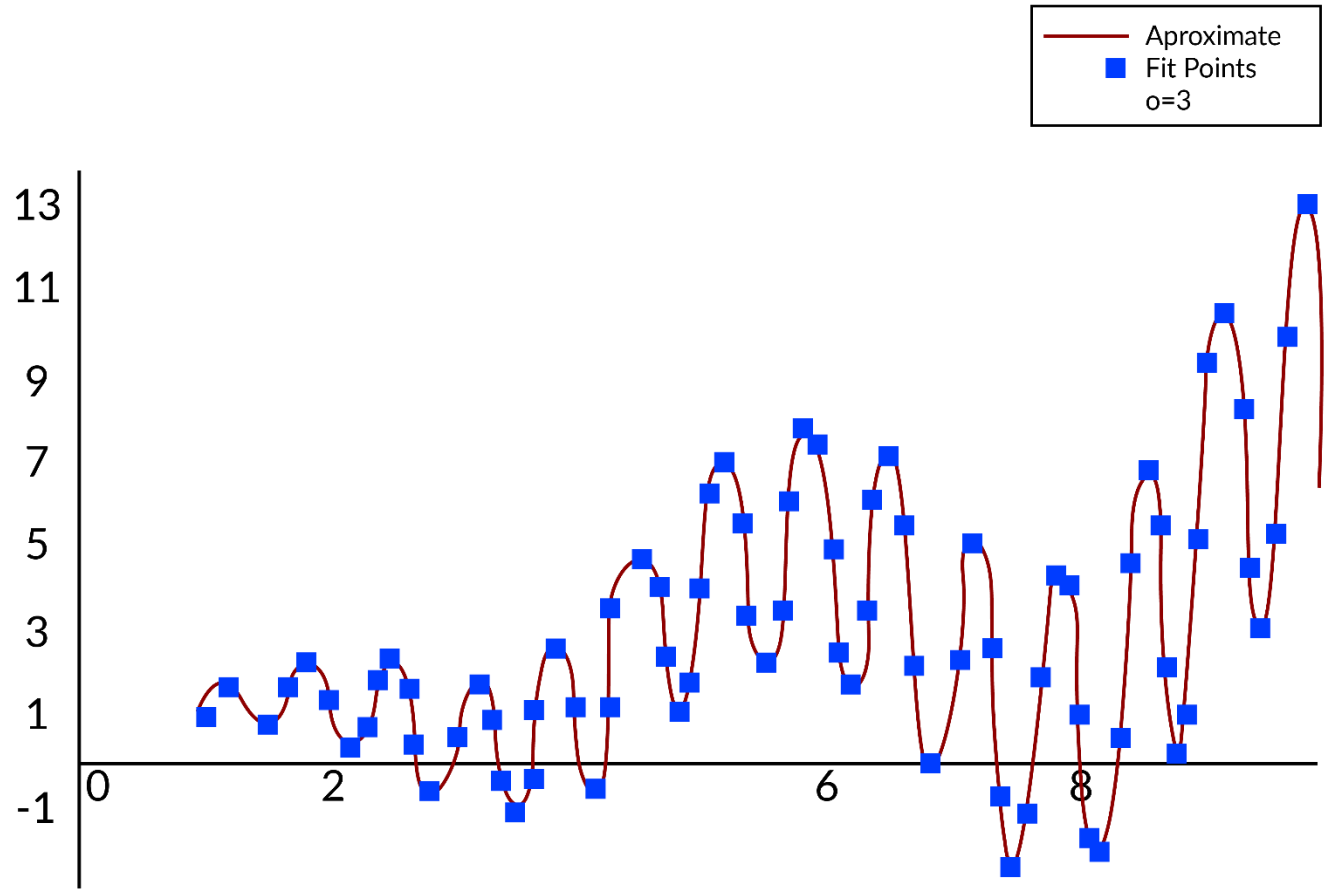


Tiene una onda de propagación frontal que se consigue utilizando una función de activación clasificatoria. Todos los demás tipos de redes neuronales utilizan la retropropagación, pero la FNN no. En la FNN, se calcula la suma de la entrada y el peso del producto, y luego se alimenta a la salida. Tecnologías como el reconocimiento de caras y la visión por ordenador utilizan FNN.

IV.2.6 Red Neuronal de Función de Base Radial

La RBFNN encuentra la distancia de un punto al centro y se considera que funciona sin problemas. Hay dos capas en la red neuronal RBF. En la capa interna, las características se combinan con la función de base radial. Las características proporcionan una salida que se utiliza en la consideración. También se pueden utilizar otras medidas en lugar de la euclidiana.

IV.2 Fundamentos de TensorFlow



IV.2 Fundamentos de TensorFlow

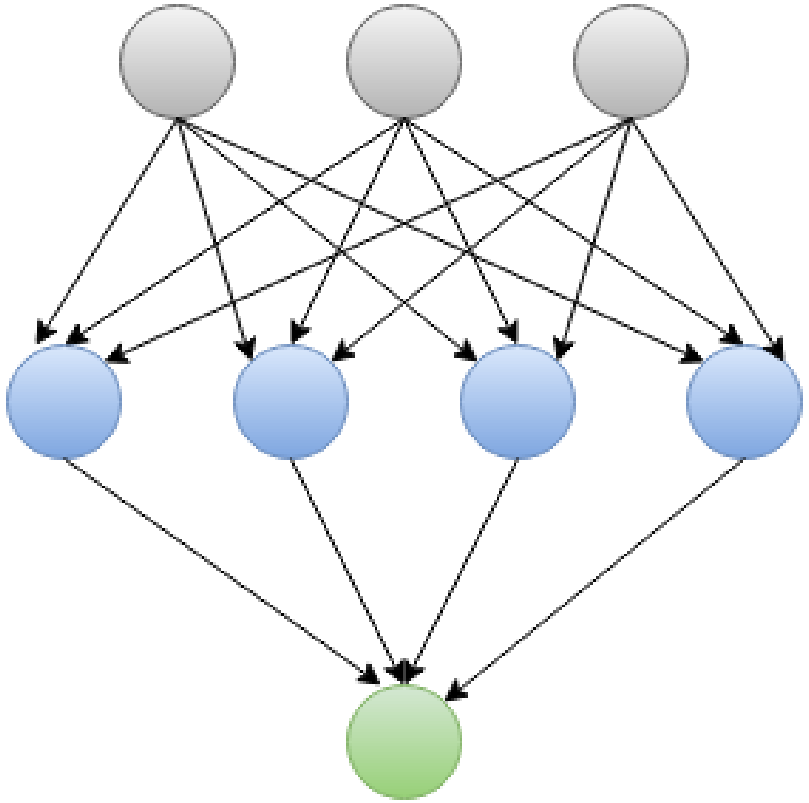
Función de Base Radial

- Definimos un receptor t
- Se dibujan mapas confrontados alrededor del receptor
- Para el RBF se utilizan generalmente funciones gaussianas. Así podemos definir la distancia radial **r=||X-t||**

Función Radial = $\Phi(r) = \exp(-r2/2\sigma2)$, donde $\sigma > 0$

Esta red neuronal se utiliza en el sistema de restauración de energía. En la era actual, el sistema de energía ha aumentado en tamaño y complejidad. Ambos factores aumentan el riesgo de que se produzcan grandes apagones. Es necesario restablecer el suministro eléctrico de la forma más rápida y fiable posible tras un apagón.

IV.2 Fundamentos de TensorFlow



Entrada X

Pesos

Función de Base Radial

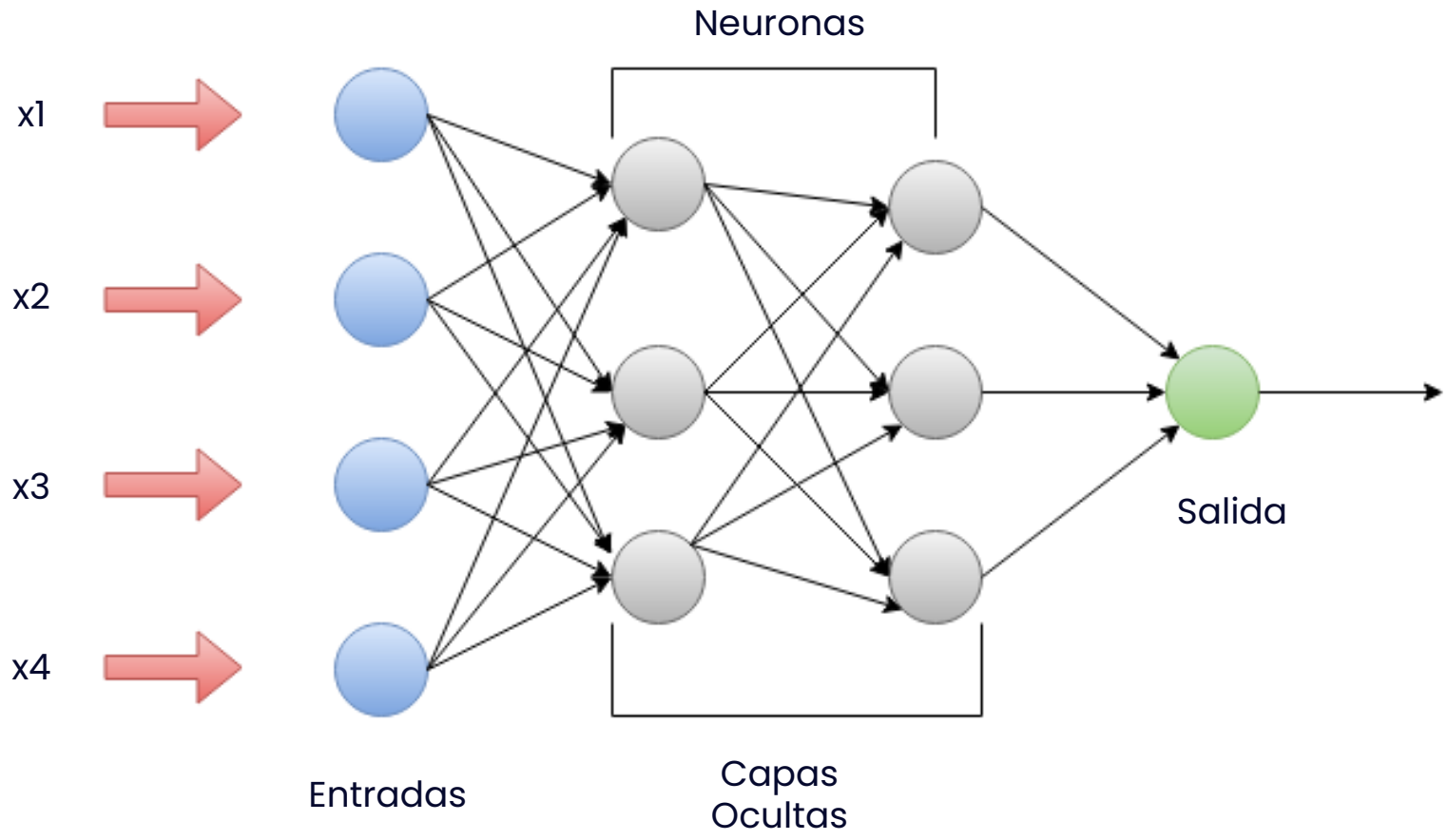
Peso Lineal

Salida Y

IV.2.7 Perceptrón Multicapa

Un **Perceptrón Multicapa** tiene tres o más capas. Los datos que no pueden separarse linealmente se clasifican con la ayuda de esta red. Esta red es una red totalmente conectada, lo que significa que cada nodo está conectado con todos los demás nodos que se encuentran en la siguiente capa. En el **perceptrón multicapa** se utiliza una **función de activación no lineal**. Los nodos de la capa de entrada y salida están conectados como un grafo dirigido. Es un método de aprendizaje profundo, por lo que para entrenar la red se utiliza la **retropropagación**. Se aplica ampliamente en las tecnologías de reconocimiento del habla y de traducción automática.

IV.2 Fundamentos de TensorFlow



IV.2 Fundamentos de TensorFlow

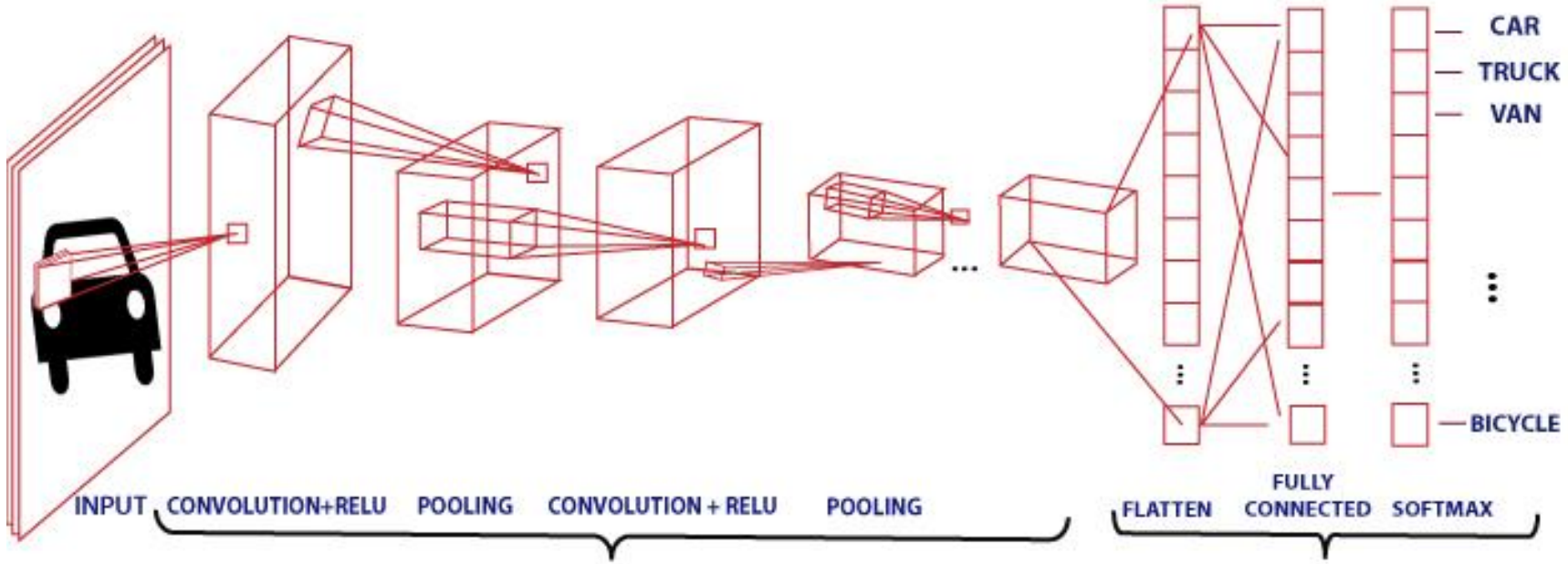
IV.2.8 Redes Neuronales Convolucionales

En la clasificación y el reconocimiento de imágenes, las **Redes Neuronales Convolucionales** desempeñan un papel fundamental, o podemos decir que son la categoría principal para ellas. El reconocimiento de caras, la detección de objetos, etc., son algunas de las áreas en las que se utilizan ampliamente las CNN. Es similar a la FNN, las neuronas tienen pesos y sesgos que se pueden aprender.

La CNN toma una imagen como entrada que se clasifica y procesa bajo una determinada categoría como perro, gato, león, tigre, etc. Como sabemos, el ordenador ve una imagen como píxeles y depende de la resolución de la imagen. Según la resolución de la imagen, verá h * w * d, donde h= altura w= anchura y d= dimensión. Por ejemplo, una imagen RGB es 6 * 6 * 3 matriz de la matriz, y la imagen en escala de grises es 4 * 4 * 3 matriz del patrón.

En la CNN, cada imagen de entrada pasará por una secuencia de capas de convolución junto con la agrupación, las capas totalmente conectadas y los filtros (también conocidos como kernels). Y aplicar la función Soft-max para clasificar un objeto con valores probabilísticos 0 y 1.

IV.2 Fundamentos de TensorFlow



Aprendizaje de Características **Clasificación**

IV.2 Fundamentos de TensorFlow

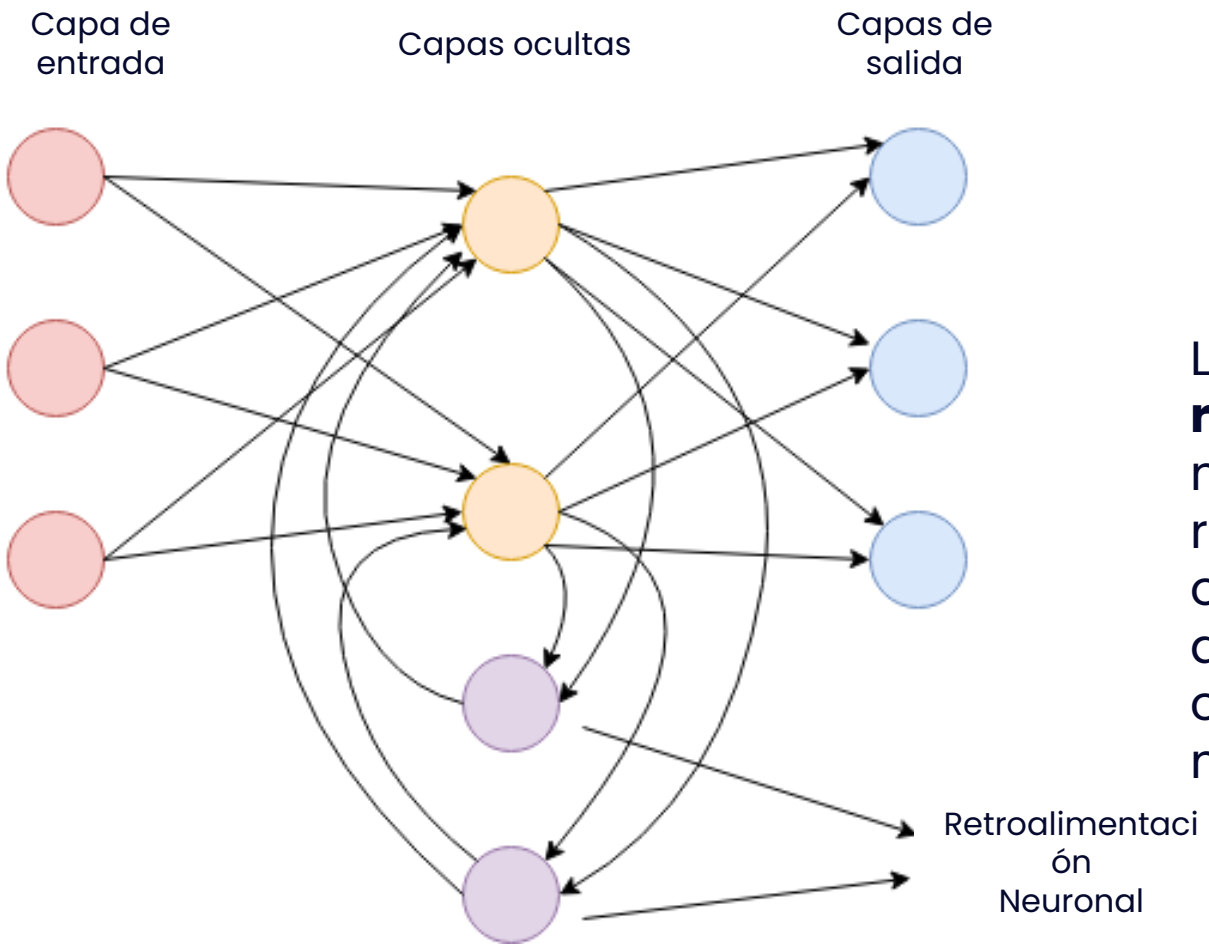
IV.2.9 Red Neuronal Recurrente

La **Red Neuronal Recurrente** se basa en la predicción. En esta red neuronal, la salida de una capa concreta se guarda y se devuelve a la entrada. Esto ayudará a predecir el resultado de la capa. En la red neuronal recurrente, la primera capa se forma de la misma manera que la capa de la FNN, y en la capa siguiente comienza el proceso de la red neuronal recurrente.

Tanto las entradas como las salidas son independientes entre sí, pero en algunos casos, se requiere predecir la siguiente palabra de la frase.

Entonces dependerá de la palabra anterior de la frase. La RNN es famosa por su característica principal y más importante, es decir, el **Estado Oculto**. El estado oculto recuerda la información sobre una secuencia.

IV.2 Fundamentos de TensorFlow

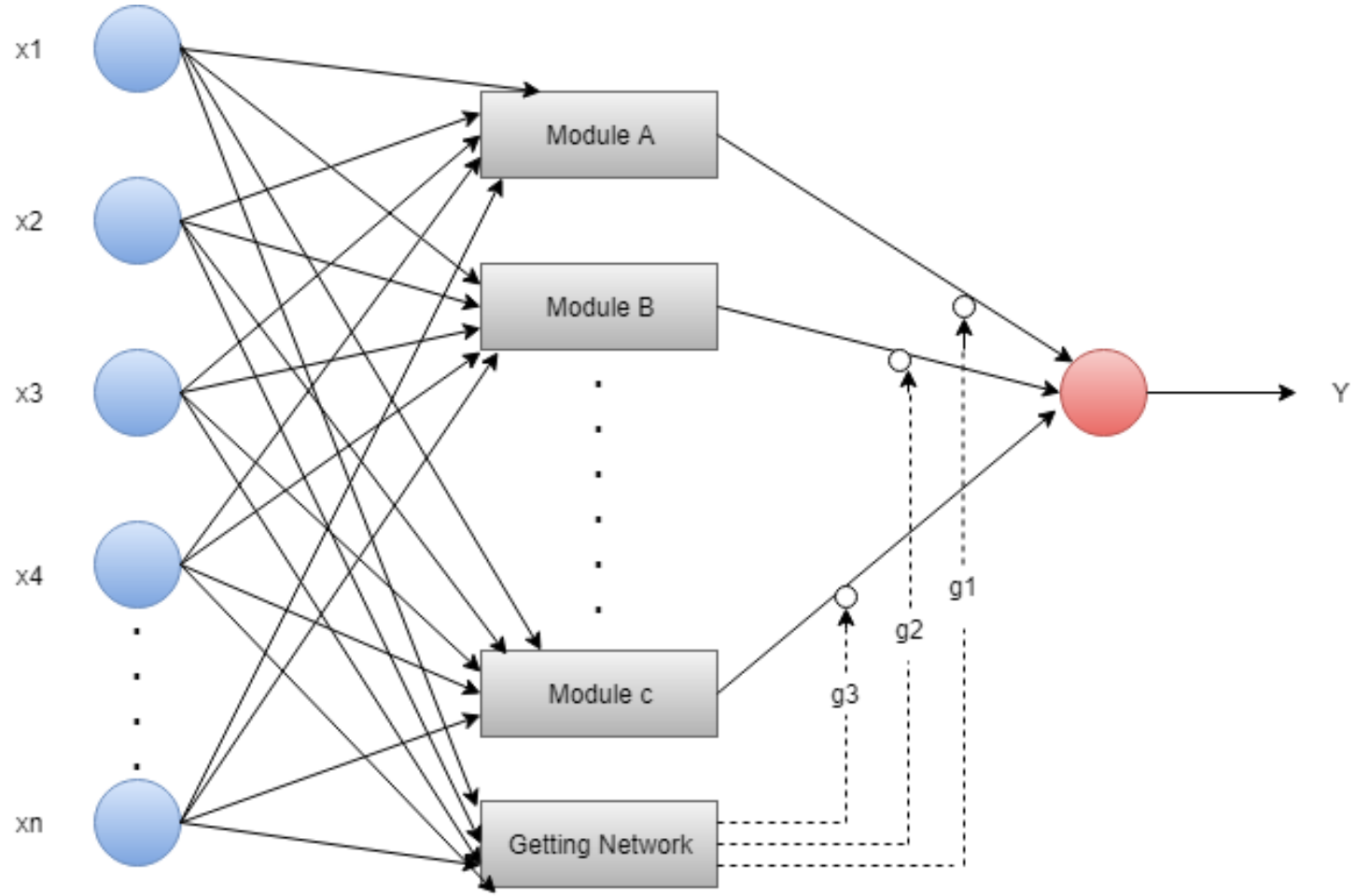


La RNN tiene una memoria para **almacenar el resultado después del cálculo**. La **RNN** utiliza los mismos parámetros en cada entrada para realizar la misma tarea en todas las capas ocultas o datos para producir la salida. A diferencia de otras redes neuronales, la complejidad de los parámetros de la RNN es menor.

IV.2.10 Redes Neuronales Modulares

En las **Redes Neuronales Modulares**, varias redes diferentes son funcionalmente independientes. En las redes neuronales modulares, la tarea se divide en subtareas y las realizan varios sistemas. Durante el proceso de cálculo, las redes no se comunican directamente entre sí. Todas las interfaces trabajan de forma independiente para conseguir el resultado. Las redes combinadas son más potentes que las planas y sin restricciones. Los intermediarios toman la producción de cada sistema y la procesan para producir el resultado final.

IV.2 Fundamentos de TensorFlow



IV.2 Fundamentos de TensorFlow

IV.2.11 Red de Secuencia a Secuencia

Consiste en dos redes neuronales recurrentes. Aquí, el codificador procesa la entrada y el decodificador procesa la salida. El codificador y el decodificador pueden utilizar el mismo parámetro o diferentes.

Los modelos secuencia a secuencia se aplican en los chatbots, la traducción automática y los sistemas de respuesta a preguntas.

IV.2.12 Componentes de una Red Neuronal Artificial

Neuronas

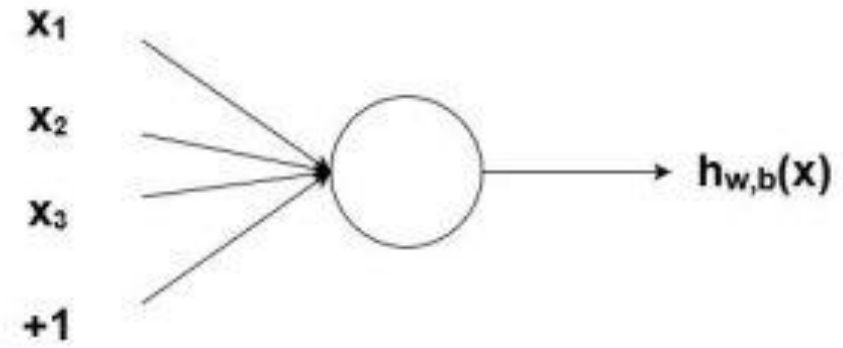
Las neuronas son similares a las neuronas biológicas. Las neuronas no son más que la función de activación. Las neuronas artificiales o la función de activación tiene una característica de **"encendido"** cuando realiza la tarea de clasificación. Podemos decir que cuando la entrada es mayor que un valor específico, la salida debe cambiar de estado, es decir, de 0 a 1, de -1 a 1, etc. La función sigmoidea es la función de activación más utilizada en las **Redes Neuronales Artificiales**.

F (Z) = 1/1+EXP (-Z)

IV.2 Fundamentos de TensorFlow

Nodos

La neurona biológica se conecta en redes jerárquicas, en las que la salida de unas neuronas es la entrada de otras. Estas redes se representan como una capa conectada de nodos. Cada nodo toma múltiples entradas ponderadas y aplica a la neurona la suma de estas entradas y genera una salida.



IV.2 Fundamentos de TensorFlow

Sesgo

En la red neuronal, predecimos la salida (y) basándonos en la entrada dada (x). Creamos un modelo, es decir, (mx + c), que nos ayuda a predecir la salida. Cuando entrenamos el modelo, éste encuentra el valor adecuado de las constantes m y c en sí mismo.

La constante c es el sesgo. El sesgo ayuda a un modelo de tal manera que puede ajustarse mejor a los datos dados. Podemos decir que el sesgo da libertad para que el modelo funcione mejor.

Algoritmo

Los **algoritmos** son necesarios en la red neuronal. Las neuronas biológicas tienen capacidad de autocomprensión y de trabajo, pero ¿cómo va a funcionar una neurona artificial de la misma manera? Para ello, es necesario entrenar nuestra red neuronal artificial. Para ello, se utilizan muchos algoritmos. Cada algoritmo tiene una forma diferente de trabajar.

IV.3 Clasificación de la Red Neuronal en TensorFlow

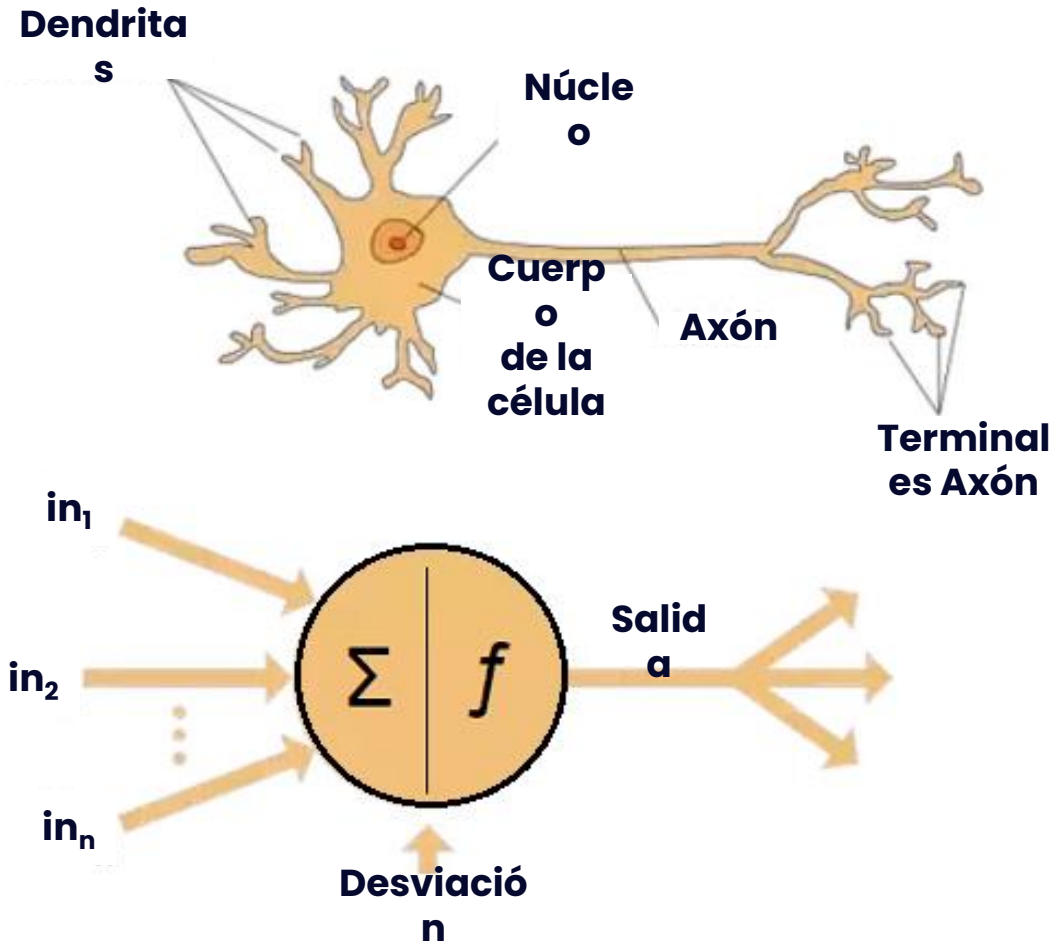
Las redes neuronales artificiales son modelos computacionales que se inspiran en las redes neuronales biológicas y están compuestas por un gran número de elementos de procesamiento altamente interconectados llamados neuronas.

Una **RNA** (Red Neuronal Artificial) se configura para una aplicación específica, como el reconocimiento de patrones o la clasificación de datos.

Puede extraer el significado de datos complicados o imprecisos.

Extrae patrones y detecta tendencias que son demasiado complejas para ser percibidas por los humanos o por otras técnicas informáticas.

IV.3 Clasificación de la Red Neuronal en TensorFlow



IV.3 Clasificación de la Red Neuronal en TensorFlow

Función de Transferencia

El comportamiento de la RNA (Red Neural Artificial) depende tanto de los pesos como de la función de entrada-salida, que se especifica para la unidad. Esta función se clasifica en una de estas tres categorías:

- Lineal (o rampa)
- Umbral
- Sigmoide

Unidades lineales: La actividad de salida es proporcional a la salida total ponderada en unidades lineales.

Umbral: La salida se establece en uno de dos niveles, dependiendo de si la entrada total es mayor o menor que algún valor de umbral.

Unidades sigmoides: La salida varía de forma continua pero no lineal a medida que cambia la entrada. Las unidades sigmoides se parecen más a las neuronas reales que las lineales o las de umbral, pero las tres deben considerarse aproximaciones.

IV.3 Clasificación de la Red Neuronal en TensorFlow

A continuación se muestra el código por el que clasificamos la red neuronal.

En primer lugar, hicimos una función de activación por lo que tenemos que trazar como POPC y para crear la función sigmoide, que es una función de activación sin esfuerzo toma en Z para hacer la sigmoide.

IV.3 Clasificación de la Red Neuronal en TensorFlow

jupyter Untitled3

File Edit View Insert Cell Kernel Widgets Help Trusted Python 2

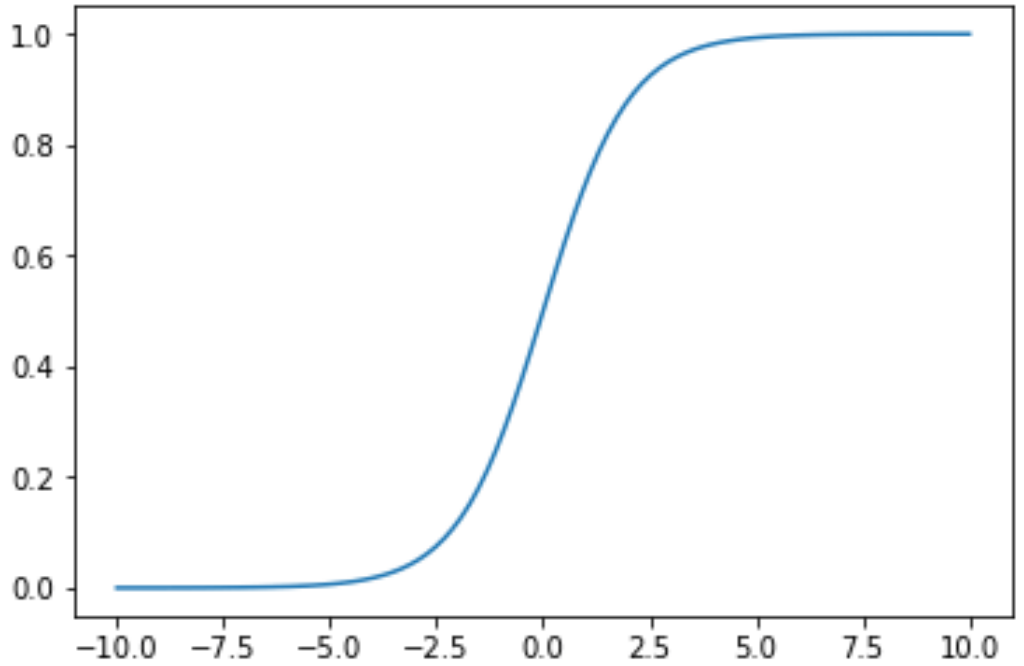
Run Code

Classification

Activation Function

```
In [6]: import matplotlib.pyplot as plt
        import numpy as np
        %matplotlib inline

In [7]: def sigmoid(z):
            return 1/(1+np.exp(-z))

In [8]: sample_z= np.linspace(-10,10,100)
        sample_a=sigmoid (sample_z)

In [9]: plt.plot(sample_z,sample_a)
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x800a6d8>]
```



IV.3 Clasificación de la Red Neuronal en TensorFlow

Entonces, hacemos la operación que hereda sigmoide. Así que vamos a ver un ejemplo de clasificación y sikat aprender tiene una función útil y capacidades para crear conjunto de datos para nosotros. Y entonces vamos a decir que mis datos es igual a hacer blobs. Sólo crea un par de blobs allí que podemos clasificar. Por lo tanto, tenemos que crear 50 muestras y el número de características a un estado que va a hacer dos blobs, así que esto es sólo un problema de clasificación binaria.

IV.3 Clasificación de la Red Neuronal en TensorFlow

```
In [21]: class Operation():
             def __init__(self,z):
                 super().__init__([z])
                 def compute(self,z_val):
                     return 1/ (1+np.exp(-z))

In [22]: from sklearn.datasets import make_blobs

In [24]: ata=make_blobs(n_samples=50,n_features=2,centers=2,random_state=75)

In [25]: data
```

```
Out[25]: (array([[  7.3402781 ,   9.36149154],
                 [  9.13332743,   8.74906102],
                 [  1.99243535,  -8.85885722],
                 [  7.38443759,   7.72520389],
                 [  7.97613887,   8.80878209],
                 [  7.76974352,   9.50899462],
                 [  8.3186688 ,  10.1026025 ],
                 [  8.79588546,   7.28046702],
                 [  9.81270381,   9.46968531],
                 [  1.57961049,  -8.17089971],
                 [  0.06441546,  -9.04982817],
                 [  7.2075117 ,   7.04533624],
                 [  9.10704928,   9.0272212 ],
                 [  1.82921897,  -9.86956281],
                 [  7.85036314,   7.986659  ],
                 [  3.04605603,  -7.50486114],
                 [  1.85582689,  -6.74473432],
                 [  2.88603902,  -8.85261704],
                 [ -1.20046211,  -9.55928542],
                 [  2.00890845,  -9.78471782],
                 [  7.68945113,   9.01706723],
                 [  6.42356167,   8.33356412],
                 [  8.15467319,   7.87489634],
                 [  1.92000795,  -7.50953708],
                 [  1.90073973,  -7.24386675],
                 [  7.7605855 ,   7.05124418],
                 [  6.90561582,   9.23493842],
                 [  0.65582768,  -9.5920878 ],
                 [  1.41804346,  -8.10517372],
                 [  9.65371965,   9.35409538]
```

IV.3 Clasificación de la Red Neuronal en TensorFlow

```
        [  1.23053506,  -7.98873571],
        [  1.96322881,  -9.50169117],
        [  6.11644251,   9.26709393],
        [  7.70630321,  10.78862346],
        [  0.79580385,  -9.00301023],
        [  3.13114921,  -8.6849493 ],
        [  1.3970852 ,  -7.25918415],
        [  7.27808709,   7.15201886],
        [  1.06965742,  -8.1648251 ],
        [  6.37298915,   9.77705761],
        [  7.24898455,   8.85834104],
        [  2.09335725,  -7.66278316],
        [  1.05865542,  -8.43841416],
        [  6.43807502,   7.85483418],
        [  6.94948313,   8.75248232],
        [ -0.07326715, -11.69999644],
        [  0.61463602,  -9.51908883],
        [  1.31977821,  -7.2710667 ],
        [  2.72532584,  -7.51956557],
        [  8.20949206,  11.90419283]]),
 array([1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
0, 0, 1,
       1, 0, 0, 0, 0, 1]))
```

IV.3 Clasificación de la Red Neuronal en TensorFlow

Ahora, tenemos que crear el diagrama de dispersión de las características de todas las filas en la columna 0 y así si hacemos diagrama de dispersión de dos manchas distintivas y capaz de clasificar estas dos clases altamente separables.

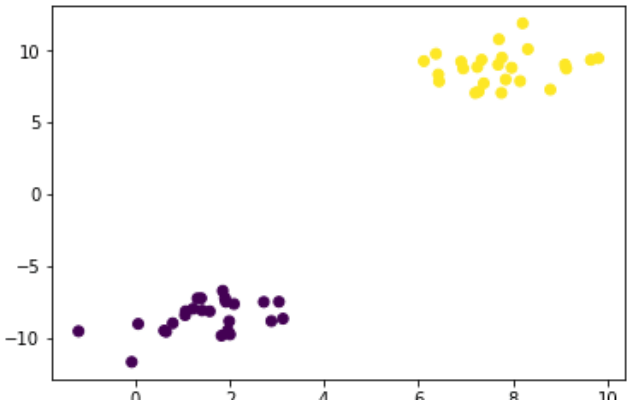
```
In [26]: type (data)

Out[26]: tuple

In [27]: data [0]

Out[27]: array([[  7.3402781 ,   9.36149154],
                [  9.13332743,   8.74906102],
                [  1.99243535,  -8.85885722],
                [  7.38443759,   7.72520389],
                [  7.97613887,   8.80878209],
                [  7.76974352,   9.50899462],
                [  8.3186688 ,  10.1026025 ],
                [  8.79588546,   7.28046702],
                [  9.81270381,   9.46968531],
                [  1.57961049,  -8.17089971],
                [  0.06441546,  -9.04982817],
                [  7.2075117 ,   7.04533624],
                [  9.10704928,   9.0272212 ],
                [  1.82921897,  -9.86956281],
                [  7.85036314,   7.986659  ],
                [  3.04605603,  -7.50486114],
                [  1.85582689,  -6.74473432],
                [  2.88603902,  -8.85261704],
                [ -1.20046211,  -9.55928542],
                [  2.00890845,  -9.78471782],
                [  7.68945113,   9.01706723],
                [  6.42356167,   8.33356412],
```

IV.3 Clasificación de la Red Neuronal en TensorFlow

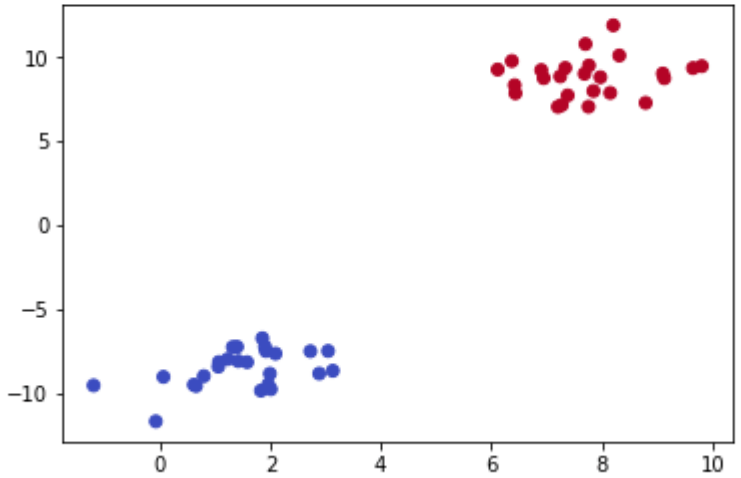
File Edit View Insert Cell Kernel Widgets Help Trusted | Python 2

Run Code

```
In [28]: data [1]

Out[28]: array([1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
                0, 1, 1,
                1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
                0, 0, 1,
                1, 0, 0, 0, 0, 1])

In [33]: features=data[0]
         labels=data[1]

In [35]: plt.scatter (features[:,0],features[:,1],c=labels)

Out[35]: <matplotlib.collections.PathCollection at 0xb2dd7b8>
```

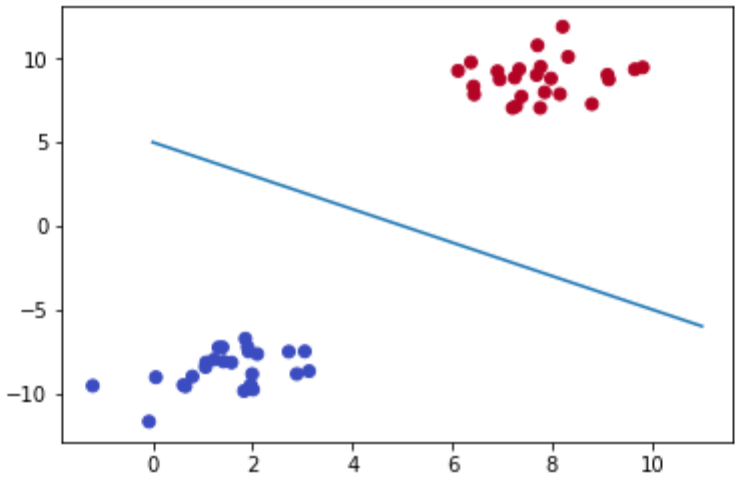



```
In [37]: plt.scatter(features[:,0],features[:,1],c=labels,cmap='coolwarm')

Out[37]: <matplotlib.collections.PathCollection at 0xb3c7c88>
```



IV.3 Clasificación de la Red Neuronal en TensorFlow

```
In [40]: x=np.linspace(0,11,10)
         y=-x +5
         plt.scatter(features[:,0],features[:,1],c=labels,cmap='coolwarm')
         plt.plot(x,y)
Out[40]: [<matplotlib.lines.Line2D at 0xb761b38>]
```



jupyter Untitled3

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 2

Run Code

(1,1)*f-5=0

```
In [41]: np.array([1,1]).dot(np.array([[8],[10]])) -5
Out[41]: array([13])

In [42]: np.array([1,1]).dot(np.array([[2],[-10]])) -5
Out[42]: array([-13])
```

IV.3 Clasificación de la Red Neuronal en TensorFlow

Aquí, vamos a construir una matriz de uno que es una matriz de uno por dos. Y luego, pasamos que en nuestra función sigmoide decir sigmoide Z porque eso es necesariamente va a la salida es 0 o 1 para nosotros como estamos clasificando sobre la base de si es positivo o negativo.

Cuanto más positiva sea la entrada, más seguro estará nuestro modelo de que pertenece a una clase.

IV.3 Clasificación de la Red Neuronal en TensorFlow

```
g = Graph()

g.set_as_default()

x = Placeholder()

w = Variable ([1,1])

b = Variable ( -5 )

z = add (matmul (w,x), b)

a = Sigmoid (z)

: a = Sigmoid (z)

: sess = Session ()

: sess.run (operation=a, feed_dict= {x:[8,10]})

: 0.99999773967570205

: sess.run (operation=a, feed_dict= {x:[2,-10]})

: 2.2603242979035746e-06
```

Así que ahora hemos sido capaces de utilizar con éxito nuestros objetos grafos variables funciones de activación a la recesión y capaz de realizar una clasificación muy simple. Y con suerte, pronto sabemos cómo hacer esto manualmente va a hacer el aprendizaje de flujo tensorial mucho y más fácil en la realización de todas las funciones esenciales con el TensorFlow.

IV.4 Regresión Lineal en TensorFlow

La regresión lineal es un algoritmo de aprendizaje automático que se basa en el aprendizaje supervisado. Realiza una función de regresión. La regresión modela un valor predictivo objetivo basado en la variable independiente. Se utiliza sobre todo para detectar la relación entre las variables y las previsiones.

La regresión lineal es un modelo lineal; por ejemplo, un modelo que asume una relación lineal entre una variable de entrada (x) y una única variable de salida (y). En concreto, y puede calcularse mediante una combinación lineal de las variables de entrada (x).

La regresión lineal es un método estadístico frecuente que nos permite aprender una función o relación a partir de un conjunto de datos continuos. Por ejemplo, se nos da un punto de datos de x y el correspondiente, y necesitamos conocer la relación entre ellos, lo que se llama la hipótesis.

IV.4 Regresión Lineal en TensorFlow

En el caso de la regresión lineal, la hipótesis es una línea recta, es decir: **h (x) = wx + b.**

Donde **w** es un vector llamado **peso**, y **b** es un escalar llamado **sesgo**. El peso y el sesgo se denominan parámetros del modelo.

Necesitamos estimar el valor de w y b a partir del conjunto de datos de forma que la hipótesis resultante produzca al menos el coste 'j', que ha sido definido por la siguiente **función de coste.**

$$J(w, b) = \frac{1}{2m}\sum_{i=1}^{m}(yi - h(xi))^2$$

Donde m son los puntos de datos en el conjunto de datos.

Esta función de coste se denomina **Error Cuadrático Medio**.

IV.4 Regresión Lineal en TensorFlow

Para la optimización de los parámetros para los que el valor de j es mínimo, utilizaremos un algoritmo optimizador comúnmente utilizado, llamado descenso de gradiente. El siguiente es el pseudocódigo para el descenso de gradiente:

```
Repeat until Convergence {
   w = w - ? * ?J/?w
   b = b - ? * ?J/?b
}
```

Implementación de la regresión lineal

Empezaremos a importar las librerías necesarias en Tensorflow. Utilizaremos **Numpy** con Tensorflow para el cálculo y **Matplotlib** para el trazado.

En primer lugar, tenemos que importar los paquetes:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
```

IV.4 Regresión Lineal en TensorFlow

Para hacer la predicción de los números aleatorios, tenemos que definir semillas fijas tanto para Tensorflow como para Numpy.

```
tf.set_random_seed(101)
np.random.seed(101)
```

Ahora, tenemos que generar algunos datos aleatorios para entrenar el Modelo de Regresión Lineal.

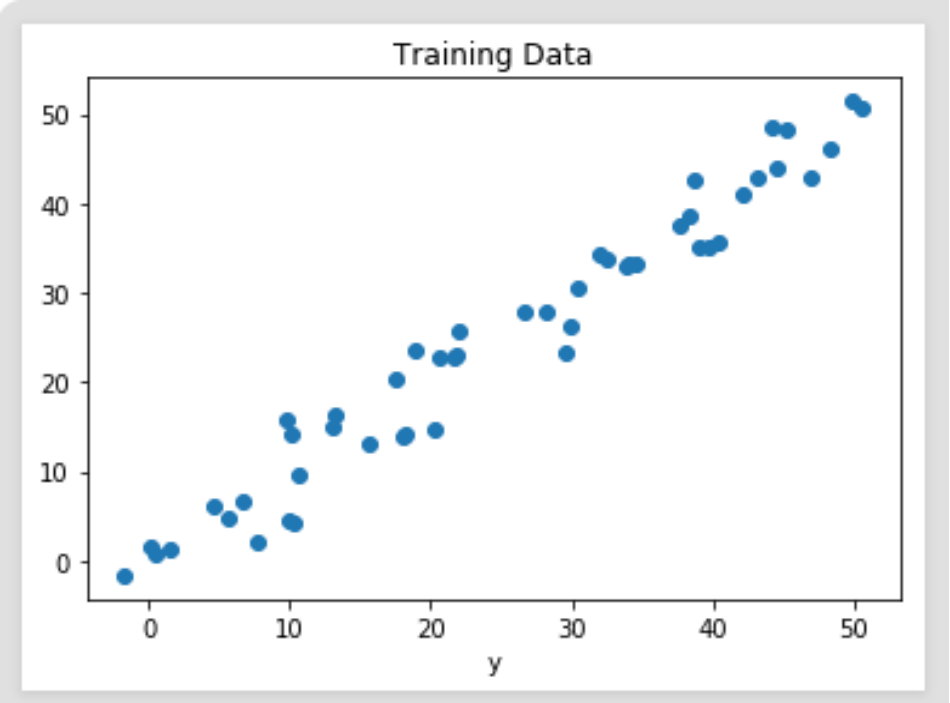
```
# Generating random linear data
# There will be 50 data points which are ranging from 0 to 50.
x = np.linspace(0, 50, 50)
y = np.linspace(0, 50, 50)

# Adding noise to the random linear data
x += np.random.uniform(-4, 4, 50)
y += np.random.uniform(-4, 4, 50)
n= len(x) #Number of data points
```

IV.4 Regresión Lineal en TensorFlow

Visualicemos los datos de entrenamiento.

```
plt.scatter(x, y)
plt.xlabel('x')
plt.xlabel('y')
plt.title("Training Data")
plt.show()
```

Salida



IV.4 Regresión Lineal en TensorFlow

Ahora, comenzaremos a construir nuestro modelo definiendo los marcadores de posición x e y, de modo que alimentemos los ejemplos de entrenamiento x e y en el optimizador durante el proceso de entrenamiento.

```
X= tf.placeholder("float")
Y= tf.placeholder("float")
```

Ahora, podemos declarar dos variables TensorFlow entrenables para el sesgo y los pesos inicializándolos aleatoriamente usando el método:

```
np.random.randn().
W= tf.Variable(np.random.randn(), name="W")
B= tf.Variable(np.random,randn(), name="b")
```

Ahora definimos el hiperparámetro del modelo, la tasa de aprendizaje y el número de épocas.

```
learning_rate= 0 .01
training_epochs= 1000
```


IV.4 Regresión Lineal en TensorFlow

Ahora, construiremos la Hipótesis, la Función de Coste y el Optimizador. No implementaremos manualmente el Optimizador Gradiente Decente porque está construido dentro de TensorFlow. Después de eso, vamos a inicializar las variables en el método.

```
# Hypothesis of the function
y_pred = tf.add(tf.multiply(X, W), b)
# Mean Square Error function
    cost = tf.reduce_sum(tf.pow(y_pred-Y, 2)) / (2 * n)
# Gradient Descent Optimizer function
optimizer = tf.train.GradientDescentOptimizer (learning_rate).minimize(cost)
# Global Variables Initializer
init = tf.global_variables_initializer( )
```

IV.4 Regresión Lineal en TensorFlow

Ahora comenzamos el proceso de entrenamiento dentro de la Sesión TensorFlow.

```
# Starting the Tensorflow Session
with tf.Session() as sess:

  # Initializing the Variables
   sess.run(init)

  # Iterating through all the epochs
   for epoch in range(training_epochs):

   # Feeding each data point into the optimizer according to the Feed Dictionary.
  for (_x, _y) in zip(x, y):
  sess.run(optimizer, feed_dict = {X : _x, Y : _y})

# Here, we are displaying the result after every 50 epoch
       if (epoch + 1) % 50 ==0:
 # Calculating the cost at every epoch.
 c = sess.run(cost, feed_dict = {X : x, Y : y})
print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(W), "b=", sess.run(b))
 # Store the necessary value which has used outside the Session
   training_cost = sess.run (cost, feed_dict ={X: x, Y: y})
```

```
print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(W), "b=", sess.run(b))
 # Store the necessary value which has used outside the Session
   training_cost = sess.run (cost, feed_dict ={X: x, Y: y})
   weight = sess.run(W)
   bias = sess.run(b)
```


IV.4 Regresión Lineal en TensorFlow

El resultado es el siguiente:

```
Epoch: 50  cost = 5.8868037 W = 0.9951241 b = 1.2381057
Epoch: 100 cost = 5.7912708 W = 0.9981236 b = 1.0914398
Epoch: 150 cost = 5.7119676 W = 1.0008028 b = 0.96044315
Epoch: 200 cost = 5.6459414 W = 1.0031956 b = 0.8434396
Epoch: 250 cost = 5.590798 W = 1.0053328 b = 0.7389358
Epoch: 300 cost = 5.544609 W = 1.007242 b = 0.6455922
Epoch: 350 cost = 5.5057884 W = 1.008947 b = 0.56223
Epoch: 400 cost = 5.473068 W = 1.01047 b = 0.46775345
Epoch: 450 cost = 5.453845 W = 1.0118302 b = 0.42124168
Epoch: 500 cost = 5.421907 W = 1.0130452 b = 0.36183489
Epoch: 550 cost = 5.4019218 W = 1.0141305 b = 0.30877414
Epoch: 600 cost = 5.3848578 W = 1.0150996  b = 0.26138115
Epoch: 650 cost = 5.370247 W = 1.0159653  b = 0.21905092
Epoch: 700 cost = 5.3576995 W = 1.0167387  b = 0.18124212
Epoch: 750 cost = 5.3468934 W = 1.0174294  b = 0.14747245
Epoch: 800 cost = 5.3375574 W = 1.0180461  b = 0.11730932
Epoch: 850 cost = 5.3294765 W = 1.0185971  b = 0.090368526
Epoch: 900 cost = 5.322459 W = 1.0190894  b = 0.0663058
Epoch: 950 cost = 5.3163588 W = 1.0195289  b = 0.044813324
Epoch: 1000 cost = 5.3110332 W = 1.0199218  b = 0.02561669
```


IV.4 Regresión Lineal en TensorFlow

Ahora, observe el resultado.

```
# Calculate the predictions
predictions = weight * x + bias
print ("Training cost =", training_cost, "Weight =", weight, "bias =", bias, '\n')
```

Resultado

```
Training cost= 5.3110332 Weight= 1.0199214 bias=0.02561663
```

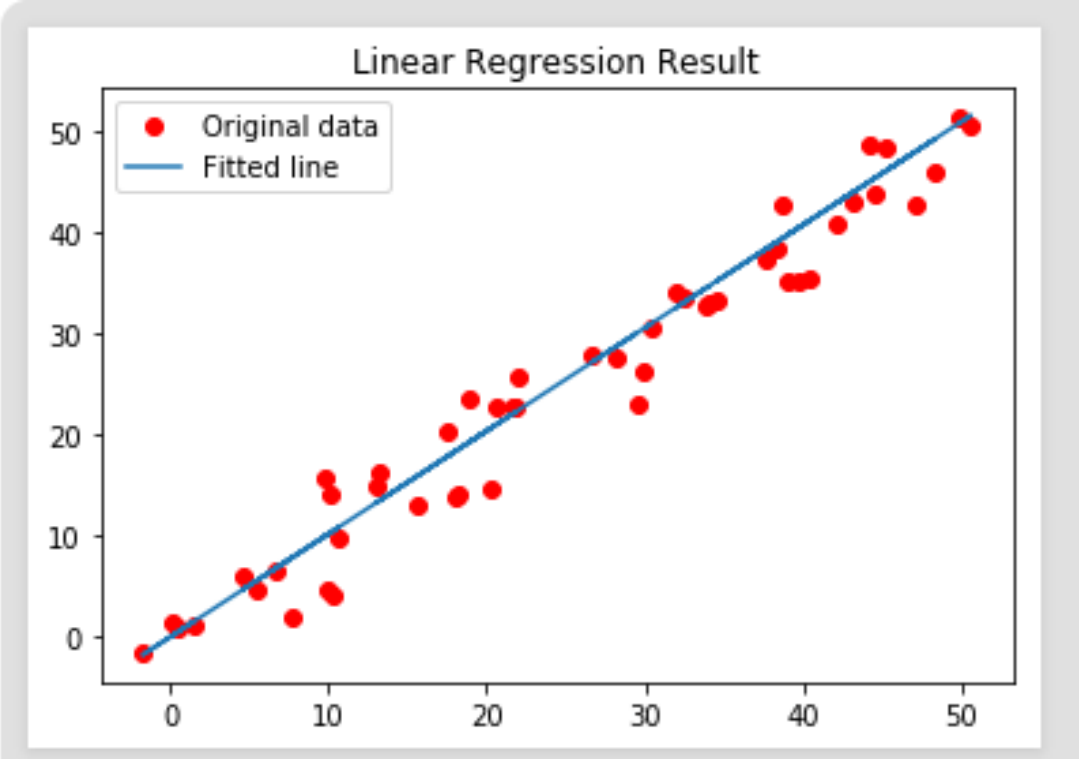
Observe que en este caso, tanto el peso como el sesgo son escalares en orden. Esto se debe a que sólo hemos examinado una variable dependiente en nuestros datos de entrenamiento. Si hay m variables dependientes en nuestro conjunto de datos de entrenamiento, el peso será un vector unidimensional mientras que el sesgo será un escalar.

IV.4 Regresión Lineal en TensorFlow

Por último, trazaremos nuestro resultado:

```
# Plotting the Results below
plt.plot(x, y, 'ro', label ='original data')
plt.plot(x, predictions, label ='Fited line')
plt.title('Linear Regression Result')
plt.legend()
plt.show()
```

Resultado



V. Bases de Keras

V. Bases de Keras

Keras es una librería de redes neuronales de alto nivel de código abierto, que está escrita en Python y es lo suficientemente capaz de ejecutarse en Theano, TensorFlow o CNTK. Fue desarrollada por uno de los ingenieros de Google, Francois Chollet. Es fácil de usar, extensible y modular para facilitar una experimentación más rápida con redes neuronales profundas. No sólo admite redes convolucionales y redes recurrentes de forma individual, sino también su combinación.

No puede manejar cálculos de bajo nivel, por lo que hace uso de la biblioteca Backend para resolverlo. La librería backend actúa como una envoltura de la API de alto nivel para la API de bajo nivel, lo que permite que se ejecute en TensorFlow, CNTK o Theano.

V.1 Capas de Keras

El enfoque en la experiencia del usuario siempre ha sido una parte importante de Keras.

- Gran adopción en la industria
- Es un multi backend y soporta multiplataforma, lo que ayuda a que todos los codificadores se unan para codificar
- La comunidad de investigación presente para Keras trabaja de forma increíble con la comunidad de producción
- Fácil de entender todos los conceptos
- Soporta la creación rápida de prototipos
- Se ejecuta sin problemas tanto en la CPU como en la GPU
- Proporciona la libertad de diseñar cualquier arquitectura, que luego se utiliza como una API para el proyecto
- Es realmente muy sencillo empezar a utilizarlo
- La facilidad de producción de modelos hace que Keras sea especial

V.1 Capas de Keras

Keras es una biblioteca a nivel de modelo que ayuda a desarrollar modelos de aprendizaje profundo ofreciendo bloques de construcción de alto nivel. Todos los cálculos de bajo nivel, como los productos de tensor, las convoluciones, etc., no son manejados por Keras en sí mismo, sino que dependen de una biblioteca de manipulación de tensor especializada que está bien optimizada para servir como motor de backend. Keras lo ha manejado tan perfectamente que en lugar de incorporar una sola librería de tensor y realizar operaciones relacionadas con esa librería en particular, ofrece la conexión de diferentes motores backend en Keras.

Keras consta de tres motores backend, que son los siguientes:

V.1 Capas de Keras

TensorFlow

TensorFlow es un producto de Google, que es una de las herramientas de aprendizaje profundo más famosas y ampliamente utilizadas en el área de investigación de aprendizaje automático y redes neuronales profundas. Salió al mercado el 9 de noviembre de 2015 bajo la licencia Apache 2.0. Está construido de tal manera que puede ejecutarse fácilmente en múltiples CPUs y GPUs, así como en sistemas operativos móviles. Consta de varias envolturas en distintos lenguajes como Java, C++ o Python.

V.1 Capas de Keras

Theano

Theano fue desarrollado en la Universidad de Montreal, Quebec, Canadá, por el grupo MILA. Es una biblioteca de código abierto en python que se utiliza ampliamente para realizar operaciones matemáticas en matrices multidimensionales mediante la incorporación de scipy y numpy. Utiliza las GPUs para un cálculo más rápido y calcula eficientemente los gradientes mediante la construcción de grafos simbólicos de forma automática. Ha resultado ser muy adecuado para expresiones inestables, ya que primero las observa numéricamente y luego las computa con algoritmos más estables.

V.1 Capas de Keras

CNTK

Microsoft Cognitive Toolkit es el marco de código abierto de aprendizaje profundo. Consta de todos los componentes básicos necesarios para formar una red neuronal. Los modelos se entrenan usando C ++ o Python, pero incorpora C # o Java para cargar el modelo y hacer predicciones.

V.1 Capas de Keras

V.1.2 Capas y Funcionamiento de la Red Neuronal de Convolución Keras

Las redes neuronales de convolución se utilizan ampliamente para tareas de visión por ordenador y clasificación de imágenes. La arquitectura de las redes neuronales de convolución suele constar de dos partes. La primera parte es el extractor de características que formamos a partir de una serie de capas de convolución y agrupación. La segunda parte incluye capas totalmente conectadas que actúan como clasificadores.

En esta sección, estudiaremos cómo utilizar las redes neuronales de convolución para tareas de clasificación de imágenes. Recorreremos algunos ejemplos para mostrar el código para la implementación de las Redes Neuronales de Convolución en Keras.

01 Implementación del RNC en el conjunto de datos CIFAR 10

02 Implementación de la CNN en el conjunto de datos Fashion MNIST

V.1 Capas de Keras

El algoritmo de la red neuronal de convolución es el resultado de los continuos avances en la visión por ordenador con el aprendizaje profundo.CNN es un algoritmo de aprendizaje profundo que es capaz de asignar importancia a varios objetos en la imagen y capaz de diferenciarlos.

La CNN tiene la capacidad de aprender las características y realizar la clasificación. Una imagen de entrada tiene muchas dependencias espaciales y temporales, la CNN captura estas características usando filtros/kernels relevantes.Un Kernel o filtro es un elemento en la CNN que realiza la convolución alrededor de la imagen en la primera parte. El kernel se mueve hacia la derecha y se desplaza según el valor de la zancada. Cada vez que se realiza la convolución se lleva a cabo una operación de multiplicación matricial.

Después de la convolución, obtenemos otra imagen con una altura, anchura y profundidad diferentes. Obtenemos más canales que el RGB pero menos anchura y altura. Desplazamos cada filtro a través de la imagen paso a paso, este paso en el forward pass se llama stride.

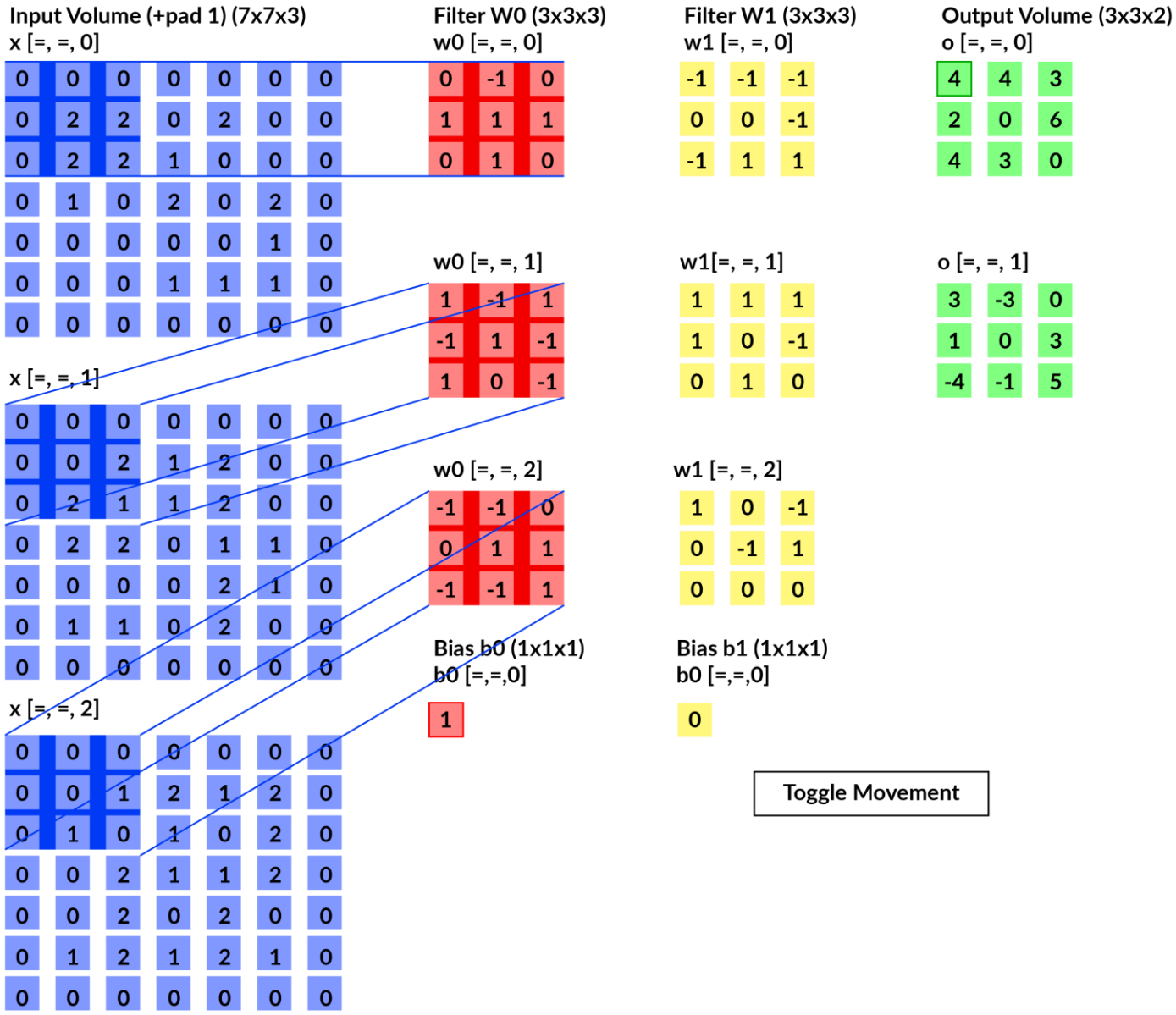
V.1 Capas de Keras

V.1.3 Capa de Convolución Keras

Es la primera capa para extraer características de la imagen de entrada. Aquí definimos el kernel como parámetro de la capa. Realizamos operaciones de multiplicación matricial en la imagen de entrada utilizando el núcleo.

Example:
Suppose a 3*3 image pixel and a 2*2 filter as shown:
pixel : [[1,0,1],
[0,1,0],
[1,0,1]]
filter : [[1,0],
[0,1]]
The restaurant matrix after convolution of filter would be:
[[2,0],
[0,2]]

V.1 Capas de Keras



V.1 Capas de Keras

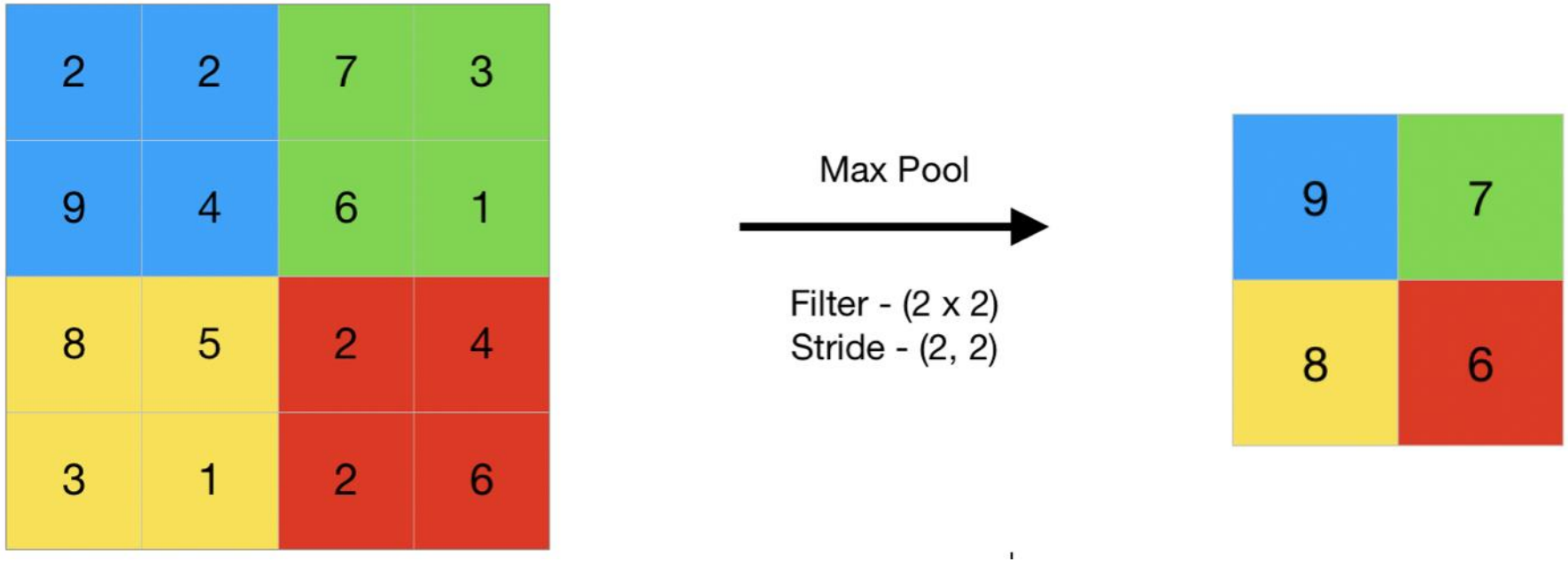
V.1.4 Capa de Agrupación Keras

Después de la convolución, realizamos un pooling para reducir el número de parámetros y cálculos. Hay diferentes tipos de operaciones de pooling, las más comunes son el pooling máximo y el pooling medio.

Example:

Take a sample case of max pooling with 2*2 filter and stride 2.
Image pixels:
[[1,2,3,4],
[5,6,7,8],
[3,4,5,6],
[6,7,8,9]]
The resultant matrix after max-pooling would be:
[[6,8],
[7,9]]

V.1 Capas de Keras



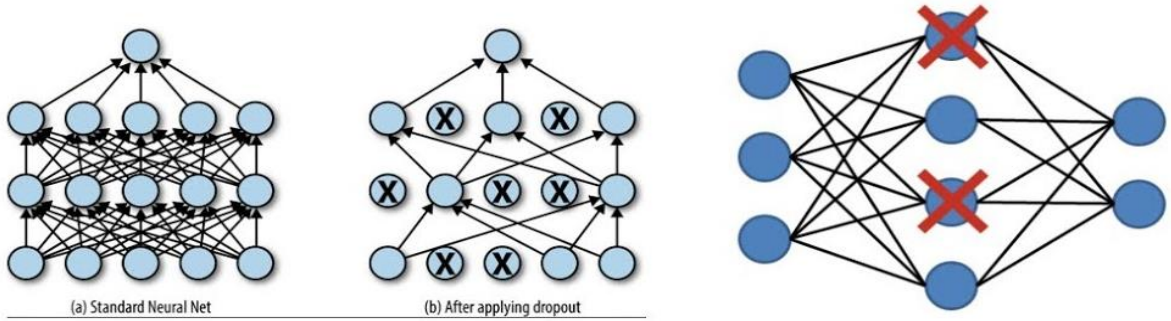
V.1 Capas de Keras

V.1.5 Capa de Abandono de Keras

Se utiliza para evitar que la red se sobreajuste. En esta capa, una parte de las unidades de la red se elimina en el entrenamiento, de modo que el modelo se entrena con todas las unidades.

Para la extracción de características se utilizan una serie de capas de convolución y de agrupación. Después, construimos capas densamente conectadas para realizar la clasificación basada en estas características.

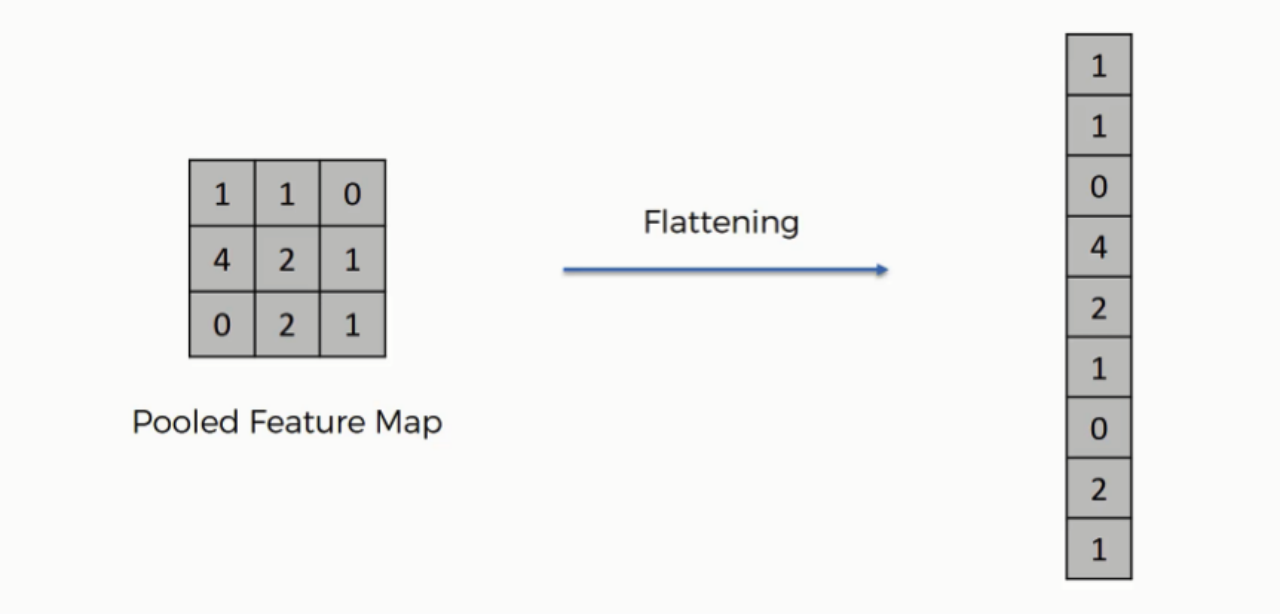
Capa de Abandono



V.1 Capas de Keras

V.1.6 Capa de Aplanamiento Keras

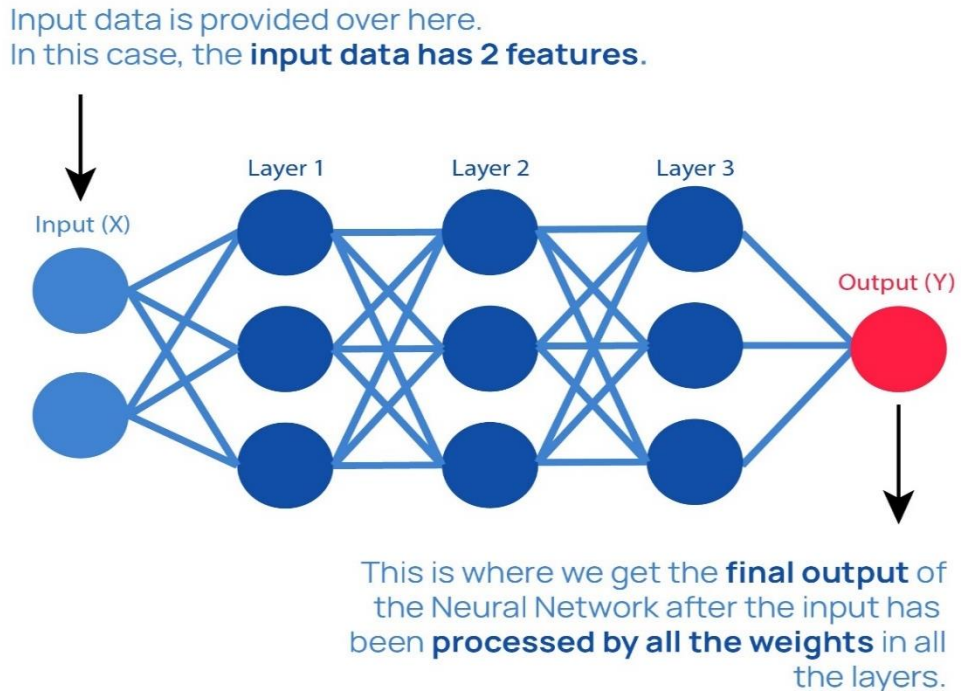
Se utiliza para convertir los datos en matrices 1D para crear un único vector de características. Tras el aplanamiento, enviamos los datos a una capa totalmente conectada para la clasificación final.



V.1 Capas de Keras

V.1.7 Capa Densa Keras

Es una capa totalmente conectada. Cada nodo de esta capa está conectado a la capa anterior, es decir, densamente conectado. Esta capa se utiliza en la fase final de la CNN para realizar la clasificación.



V.1 Capas de Keras

V1.8 Implementación de la CNN en el conjunto de datos CIFAR 10

El conjunto de datos CIFAR 10 consta de 10 clases de imágenes. Las clases de imágenes disponibles son:

- Car - Carro
- Airplane - Avión
- Bird - Pajaro
- Cat - Gato
- Deer - Ciervo
- Dog - Perro
- Frog - Rana
- Horse - Caballo
- Ship - Oveja
- Truck - Camion

Se trata de uno de los conjuntos de datos más populares que permiten a los investigadores practicar diferentes algoritmos de reconocimiento de objetos. Las redes neuronales de convolución han mostrado los mejores resultados en la resolución del problema CIFAR-10.

V.1 Capas de Keras

Construyamos nuestro modelo de convolución para reconocer las clases CIFAR-10.

1. **Cargar el conjunto de datos del módulo keras datasets**

```
from keras.datasets import cifar10
import matplotlib.pyplot as plt

(train_X,train_Y),(test_X,test_Y)=cifar10.load_data()
```

2. **Para visualizar el conjunto de datos**

```
n=6
plt.figure(figsize=(20,10))
for i in range(n):
plt.subplot(330+1+i)
plt.imshow(train_X[i])
plt.show()
```

3. **El conjunto de datos tiene el siguiente aspecto:**


```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
```

V.1 Capas de Keras

4. Normalización de las entradas

```
train_x=train_X.astype('float32')
test_X=test_X.astype('float32')

train_X=train_X/255.0
test_X=test_X/255.0
```

5. Una codificación en caliente

```
train_Y=np_utils.to_categorical(train_Y)
test_Y=np_utils.to_categorical(test_Y)

num_classes=test_Y.shape[1]
```

6. Construir el modelo

```
model=Sequential()
model.add(Conv2D(32,(3,3),input_shape=
(32,32,3),padding='same',activation='relu',kernel_constraint=maxnorm(3))
model.add(Dropout(0.2))
model.add(Conv2D(32,
(3,3),activation='relu',padding='same',kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(512,activation='relu',kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

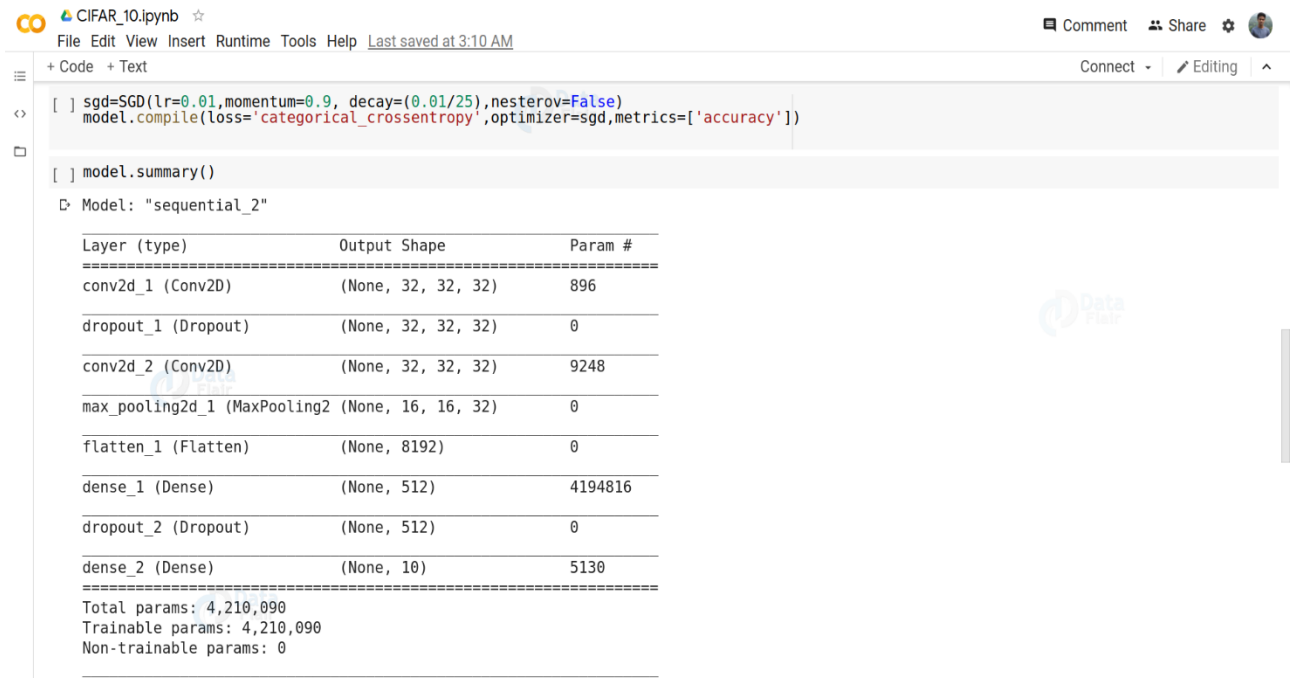
7. Compilación del modelo

```
sgd=SGD(lr=0.01,momentum=0.9, decay=(0.01/25),nesterov=False)
model.compile(loss='categorical_crossentropy',optimizer=sgd,metrics=
['accuracy'])
```


V.1 Capas de Keras

8. Análisis de la síntesis del modelo

```
1.  model.summary()
```



V.1 Capas de Keras

9. Entrenar el modelo y comprobar su precisión en los datos de prueba

```
model.fit(train_X,train_Y,validation_data=
(test_X,test_Y),epochs=10,batch_size=32)
```

10. Evaluar el modelo

```
_,acc=model.evaluate(test_X,test_Y)
print(acc*100)
```

V.1.9 Implementación de la CNN en la base de datos Fashion MNIST

El conjunto de datos **The Fashion MNIST** consta de un conjunto de entrenamiento de 60000 imágenes y un conjunto de prueba de 10000 imágenes. Hay 10 clases de imágenes en este conjunto de datos y cada clase tiene un mapeo correspondiente a las siguientes etiquetas:

1. T-shirt/top – Camiseta/Top
2. Trouser - Pantalones
3. Pullover - Jersey
4. Dress - Vestido
5. Coat - Abrigo
6. Sandals - Sandalias
7. Shirt – Camiisa
8. Sneaker – Zapatillas de deporte
9. Bag - Bolso
10. Ankle boot - Botin

Vamos a construir nuestro modelo CNN en este conjunto de datos.

V.1 Capas de Keras

1. Importar los módulos necesarios

```
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
from keras.datasets import fashion_mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
```

2. Cargar el conjunto de datos

```
(train_X,train_Y),(test_X,test_Y)=fashion_mnist.load_data()
```

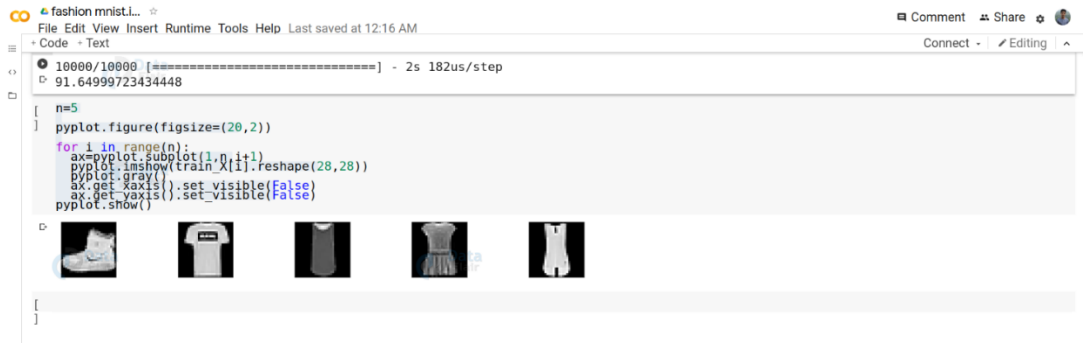
3. Reformulación y codificación en caliente

```
train_X=train_X.reshape((train_X.shape[0],28,28,1))
test_X=test_X.reshape((test_X.shape[0],28,28,1))

train_Y=to_categorical(train_Y)
test_Y=to_categorical(test_Y)
```


V.1 Capas de Keras

4. Visualizar el conjunto de datos con matplotlib

```
n=5

pyplot.figure(figsize=(20,2))

for i in range(n):
ax=pyplot.subplot(1,n,i+1)
pyplot.imshow(train_X[i].reshape(28,28))
pyplot.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
pyplot.show()
```



5. Normalización de datos

```
train_X=train_X.astype('float32')
test_X=test_X.astype('float32')

train_X=train_X/255.0
test_X=test_X/255.0
```

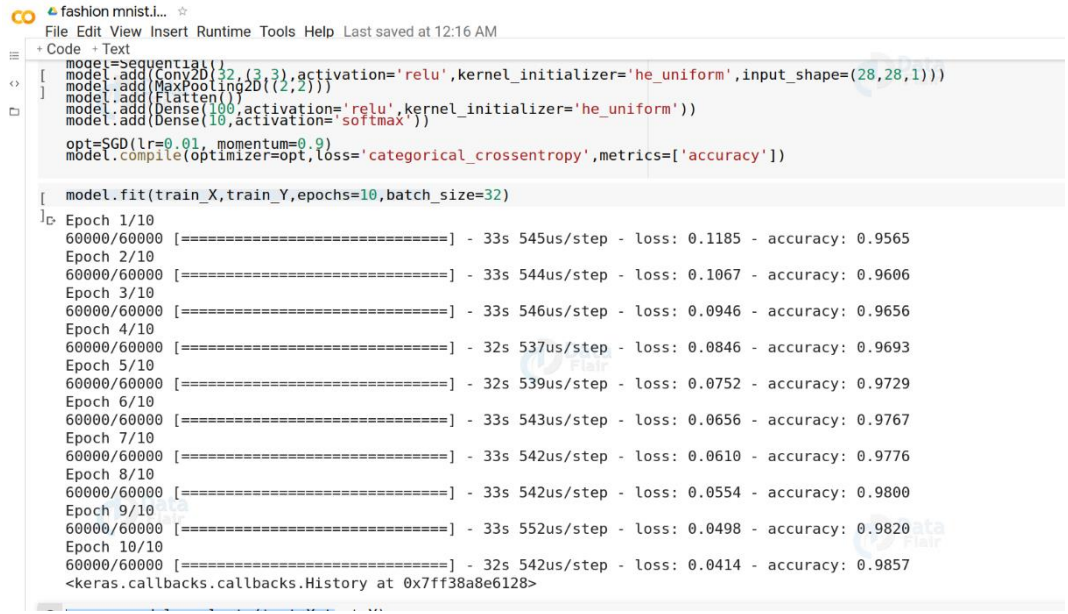
6. Construcción del Modelo

```
model=Sequential()
model.add(Conv2D(32,
(3,3),activation='relu',kernel_initializer='he_uniform',input_shape=
(28,28,1)))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())

model.add(Dense(100,activation='relu',kernel_initializer='he_uniform'))
model.add(Dense(10,activation='softmax'))

opt=SGD(lr=0.01, momentum=0.9)
model.compile(optimizer=opt,loss='categorical_crossentropy',metrics=
['accuracy'])
```

V.1 Capas de Keras

7. Entrenamiento de nuestro modelo

```
model.fit(train_X,train_Y,epochs=10,batch_size=32)
```



8. Evaluar el rendimiento de nuestro modelo

```
_, acc=model.evaluate(test_X,test_Y)
print(acc*100)
```

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

En esta sección de Keras, recorreremos el **aprendizaje profundo con keras** y un importante **algoritmo de aprendizaje profundo** utilizado en keras. Estudiaremos las aplicaciones de este algoritmo y también su implementación en Keras.

El aprendizaje profundo es un subconjunto del aprendizaje automático que se refiere a los algoritmos inspirados en la arquitectura del cerebro. En la última década ha habido muchos desarrollos importantes para apoyar la investigación del aprendizaje profundo. Keras es el resultado de uno de estos desarrollos recientes que nos permiten definir y crear modelos de redes neuronales en unas pocas líneas de código.

Se ha producido un boom en la investigación de **algoritmos de Deep Learning**. Keras asegura la facilidad de los usuarios para crear estos algoritmos.

Pero antes de empezar con el artículo Tensorflow Keras Deep learning, vamos a hacer la **keras installation**.

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

A continuación se mencionan algunos de los algoritmos más populares en el aprendizaje profundo:

- Auto-Encodificadores
- Redes neuronales de convolución
- Redes neuronales recurrentes
- Redes de memoria a corto plazo
- Máquina profunda de Boltzmann (DBM)
- Redes de creencia profunda (DBN)

Existen implementaciones de **redes neuronales de convolución, redes neuronales recurrentes y LSTM**

Aquí haremos un recorrido por el algoritmo de **Auto Codificadores del aprendizaje profundo.**

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

Autocodificadores

Este tipo de redes neuronales son capaces de comprimir los datos de entrada y reconstruirlos de nuevo. Son algoritmos de aprendizaje profundo muy antiguos. Codifican la entrada hasta una capa de cuello de botella y luego la decodifican para recuperar la entrada. En la capa cuello de botella, obtenemos una forma comprimida de la entrada.

La detección de anomalías y la eliminación de ruido de una imagen son algunas de las principales aplicaciones de los autocodificadores.

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

Tipos de autocodificadores

Existen siete tipos de autocodificadores de aprendizaje profundo que se mencionan a continuación:

- Autocodificadores de eliminación de ruido
- Autocodificadores profundos
- Autocodificadores dispersos
- Autocodificadores contractivos
- Autocodificadores convolucionales
- Autocodificadores variacionales
- Autocodificadores incompletos

Para nuestro estudio, crearemos un autocodificador Denoising.

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

Implementación del autocodificador de denostación en Keras

Para su implementación en Keras, trabajaremos sobre el conjunto de datos de dígitos manuscritos MNIST.

En primer lugar, introduciremos algo de ruido en las imágenes MNIST. A continuación, crearemos un **Auto – Encoder** para eliminar el ruido de las imágenes y reconstruir las imágenes originales.

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

1. Importar los módulos necesarios

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.layers import Input,Dense,Conv2D,MaxPooling2D,UpSampling2D

from keras.models import Model
from keras import backend as K
```

2. Cargar imágenes MNIST desde el módulo datasets de keras

```
from keras.datasets import mnist
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

3. Convert dataset in range of 0 to 1

```
x_train=x_train.astype('float32')/255
x_test=x_test.astype('float32')/255


x_train=np.reshape(x_train,(len(x_train),28,28,1))
x_test=np.reshape(x_test,(len(x_test),28,28,1))
```

4. Introducir ruido en las imágenes MNIST utilizando una distribución gaussiana

```
noise_factor=0.5

x_train_noisy=x_train + noise_factor * np.random.normal(loc=0.0,
scale=1.0,size=x_train.shape)
x_test_noisy=x_test + noise_factor * np.random.normal(loc=0.0,
scale=1.0,size=x_test.shape)

x_train_noisy= np.clip(x_train_noisy,0.,1.)
x_test_noisy= np.clip(x_test_noisy,0.,1.)
```

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

5. Visualizar el ruido introducido

```
1.  n=5
2.
3.  plt.figure(figsize=(20,2))
4.
5.  for i in range(n):
6.  ax=plt.subplot(1,n,i+1)
7.  plt.imshow(x_test_noisy[i].reshape(28,28))
8.  plt.gray()
9.  ax.get_xaxis().set_visible(False)
10. ax.get_yaxis().set_visible(False)
11. plt.show()
```

jupyter denoising autoencoder Last Checkpoint: 32 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

```
x_test=np.reshape(x_test,(len(x_test),28,28,1))

noise_factor=0.5

x_train_noisy=x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,size=x_train.sh
x_test_noisy=x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,size=x_test.shape

x_train_noisy= np.clip(x_train_noisy,0.,1.)
x_test_noisy= np.clip(x_test_noisy,0.,1.)
```

In [12]:

```
n=5

plt.figure(figsize=(20,2))

for i in range(n):
    ax=plt.subplot(1,n,i+1)
    plt.imshow(x_test_noisy[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

In [ ]:

V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

6. **Especificar la capa de entrada y crear el modelo**

```
input_img=Input(shape=(28,28,1))

x=Conv2D(32,(3,3),activation='relu',padding='same')(input_img)
x=MaxPooling2D((2,2),padding='same')(x)
x=Conv2D(32,(3,3),activation='relu',padding='same')(x)
encoded=MaxPooling2D((2,2),padding='same')(x)
```

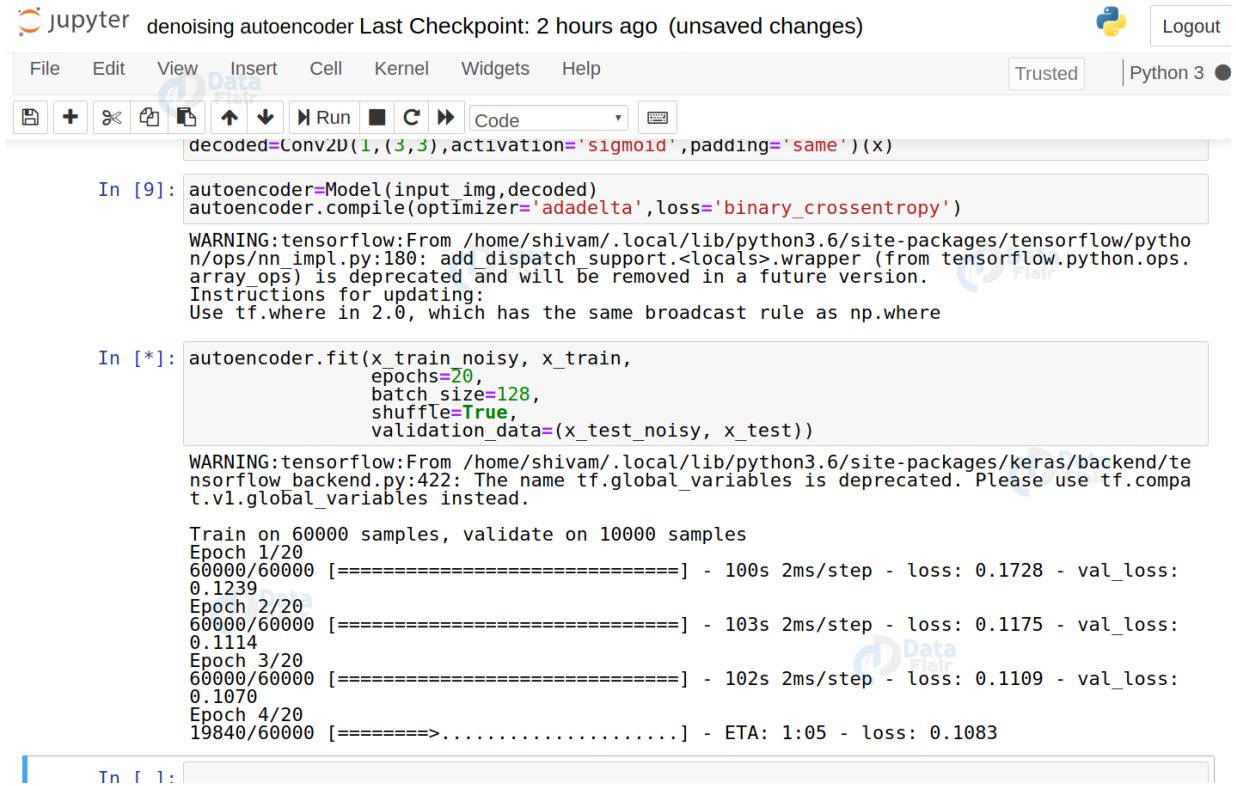
7. **La capa codificada es el cuello de botella y consiste en una forma comprimida de imágenes**

```
x=Conv2D(32,(3,3),activation='relu',padding='same')(encoded)
x=UpSampling2D((2,2))(x)
x=Conv2D(32,(3,3),activation='relu',padding='same')(x)
x=UpSampling2D((2,2))(x)
decoded=Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)
```


V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

8. Entrenar el autocodificador

```
1. x=Conv2D(32,(3,3),activation='relu',padding='same')(encoded)
2. x=UpSampling2D((2,2))(x)
3. x=Conv2D(32,(3,3),activation='relu',padding='same')(x)
4. x=UpSampling2D((2,2))(x)
5. decoded=Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)
```



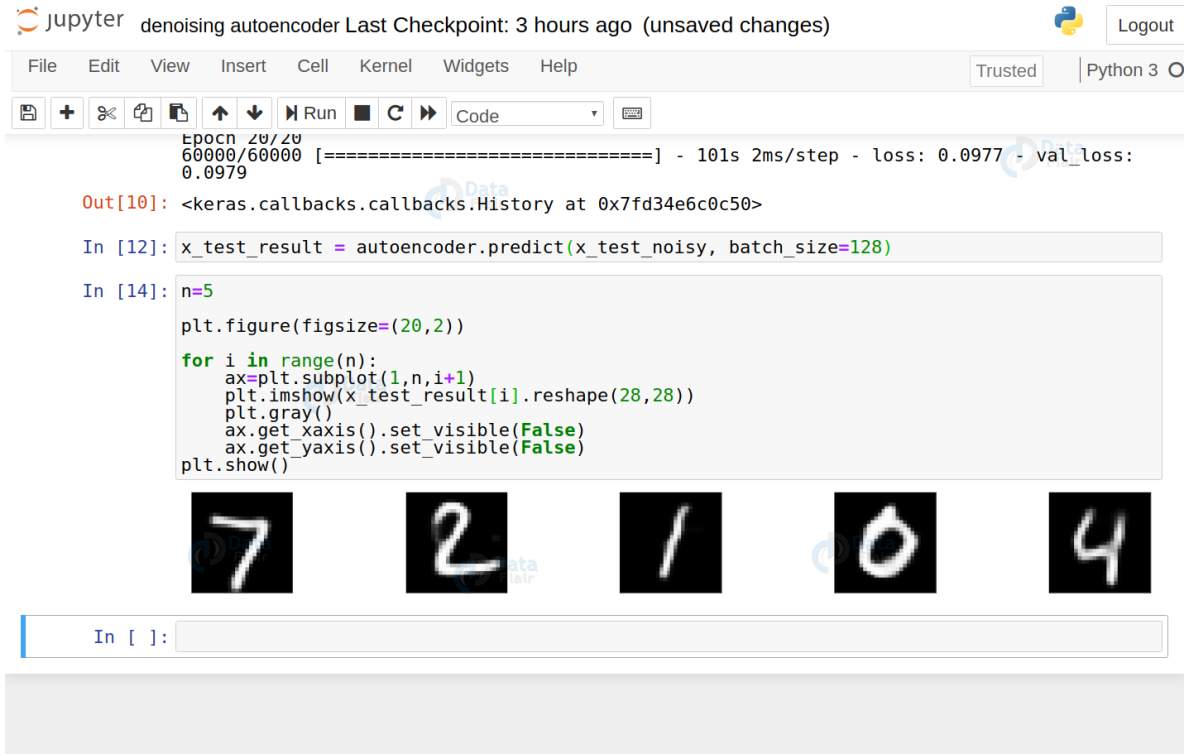
V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

9. **Obtenga una predicción de los datos con ruido**

```
1. x_test_result = autoencoder.predict(x_test_noisy, batch_size=128)
```

10. **Visualizar de nuevo las imágenes reconstruidas**

```
1. n=5
2.
3. plt.figure(figsize=(20,2))
4.
5. for i in range(n):
6. ax=plt.subplot(1,n,i+1)
7. plt.imshow(x_test_result[i].reshape(28,28))
8. plt.gray()
9. ax.get_xaxis().set_visible(False)
10. ax.get_yaxis().set_visible(False)
11. plt.show()
```



V.2 Implementación y Ejemplo de Aprendizaje con Aprendizaje Profundo

Se puede ver que nuestro codificador automático es capaz de reconstruir las imágenes y eliminar su ruido. Obtendremos mejor calidad si aumentamos el número de épocas de entrenamiento.

Para concluir, hemos visto la implementación y el ejemplo de Deep learning con Keras. Este artículo se refiere a la librería Keras y su soporte para implementar los principales algoritmos de deep learning. También se introduce a los Auto-Encodificadores, sus diferentes tipos, sus aplicaciones y su implementación. Se explica cómo construir una red neuronal para eliminar el ruido de nuestros datos.

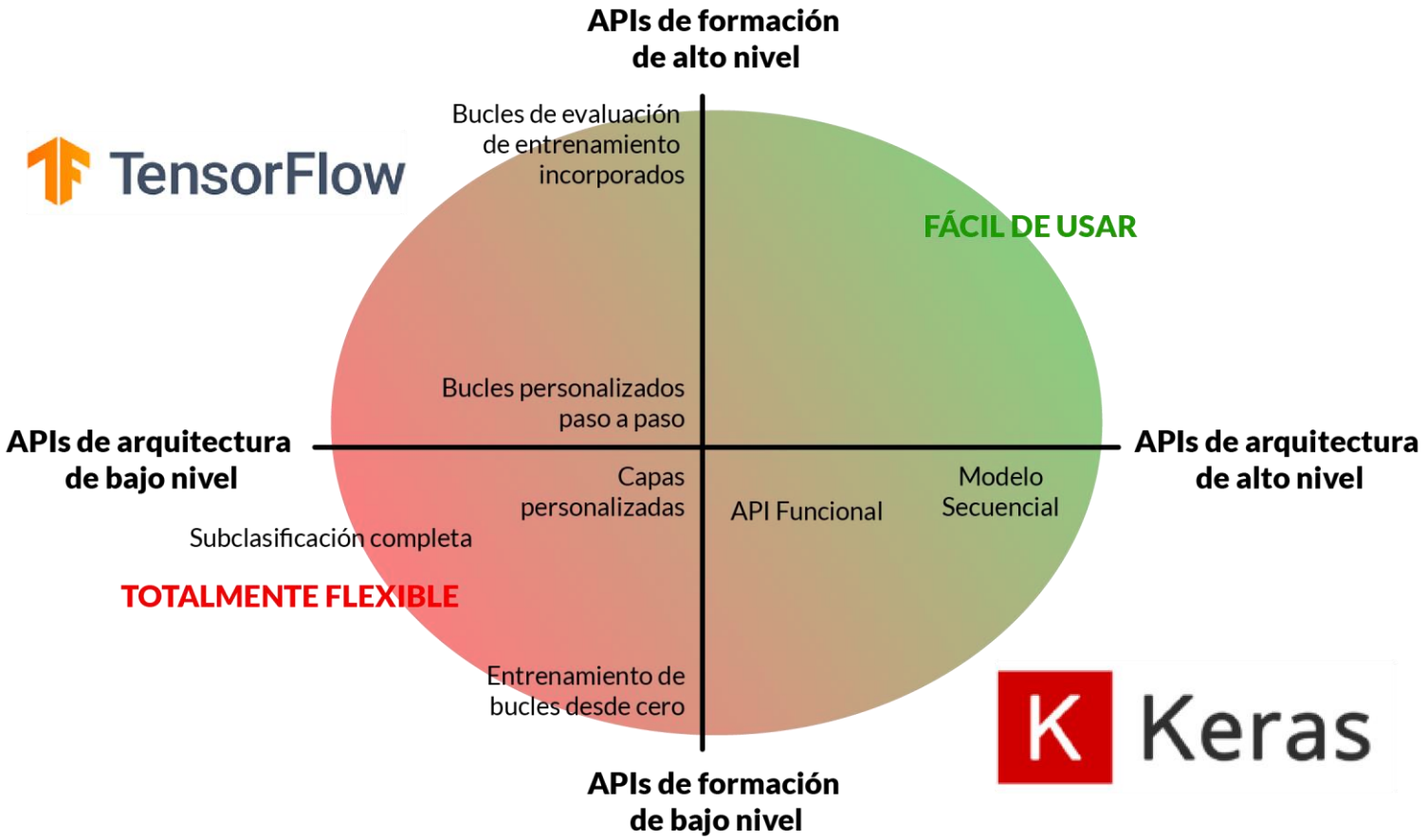
V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

Keras y Tensorflow son dos marcos de aprendizaje profundo muy populares. Los profesionales del aprendizaje profundo utilizan más ampliamente Keras y Tensorflow. Ambos marcos tienen un gran apoyo de la comunidad. Ambos marcos capturan una fracción importante de la producción de aprendizaje profundo.

Hay algunas diferencias entre Keras y Tensorflow, que le ayudarán a elegir entre los dos. Le proporcionaremos una mejor visión de estos dos marcos.

Los siguientes puntos le ayudarán a aprender la comparación entre Tensorflow y Keras para encontrar cuál es más adecuado para usted.

V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow



V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

Complejidad

Keras permite el desarrollo de modelos sin preocuparse de los detalles del backend. Mientras que en TensorFlow hay que ocuparse de los detalles computacionales en forma de tensores y gráficos.

Esta característica de Keras proporciona más comodidad y lo hace menos complejo que TensorFlow.

API fácil de usar Keras es una API de alto nivel.

Keras utiliza Tensorflow, Theano o CNTK como motores de backend.Tensorflow ofrece APIs de alto y bajo nivel. Tensorflow es una biblioteca matemática que utiliza la programación de flujo de datos para una amplia variedad de tareas.Si buscas una herramienta de redes neuronales que sea fácil de usar y tenga una sintaxis sencilla, entonces estarás mejor servido por Keras.

V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

Desarrollo rápido

Si quieres desplegar y probar rápidamente tus modelos de aprendizaje profundo, elige Keras. Usando Keras, puedes crear tus modelos con muy pocas líneas de código y en pocos minutos. Keras proporciona dos APIs para escribir tu red neuronal. Estas son:

- Modelo (API funcional)
- Secuencial

Con estas APIs, puedes crear fácilmente cualquier red neuronal compleja.

V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

Rendimiento

Dado que Keras no es directamente responsable del cálculo del backend, Keras es más lento. Keras depende de sus motores de backend para las tareas de computación. Proporciona una abstracción sobre su backend. Para realizar los cálculos subyacentes y el entrenamiento, Keras llama a su backend.Por otro lado, Tensorflow es una biblioteca matemática simbólica. Su compleja arquitectura se centra en reducir la carga cognitiva de los cálculos. Por lo tanto, Tensorflow es rápido y proporciona un alto rendimiento.

Funcionalidad y flexibilidad

Tensorflow te da más flexibilidad, más control y funciones avanzadas para la creación de topologías complejas. Proporciona más control sobre su red. Por lo tanto, si quieres definir tu propia función de coste, métrica o capa, o si quieres realizar operaciones sobre los pesos de entrada o los gradientes, elige TensorFlow.

V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

Conjunto de Datos

Preferimos Keras si el tamaño del conjunto de datos es relativamente pequeño o mediano. Mientras que si el conjunto de datos es grande, preferimos TensorFlow porque tiene menos gastos generales. Además, TensorFlow proporciona un mayor nivel de control, por lo que tenemos más opciones para manejar grandes conjuntos de datos.

TensorFlow proporciona un mayor número de conjuntos de datos incorporados que Keras. Contiene todos los conjuntos de datos que están disponibles en Keras y el módulo tf.datasets de TensorFlow contiene una amplia gama de conjuntos de datos y estos se clasifican en los siguientes apartados: Audio, imagen, clasificación de imágenes, detección de objetos, respuesta a preguntas, estructurado, resumen, texto, traducción y vídeo.

Los conjuntos de datos en Keras están presentes en el módulo Keras.datasets.

V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

Depuración

La depuración del código TensorFlow es muy difícil. En general, realizamos la depuración en el depurador de TensorFlow y lo hacemos a través de la línea de comandos. Comenzamos envolviendo la sesión de TensorFlow con, **tf_debug.LocalCLIDebugWrapperSession(session)**, y luego ejecutamos el archivo con diferentes banderas de depuración necesarias.

Keras es de alto nivel y no se ocupa de la computación del backend, por lo tanto la depuración es fácil. También podemos comprobar la salida de cada capa en **Keras** usando **keras.backend.function().**

V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

Popularidad

Keras tiene 48,7k estrellas en github y 18,4k forks en github. Mientras que TensorFlow tiene 146k estrellas y 81.7k forks en github.

Dado que tanto Keras como TensorFlow fueron lanzados en 2015, está claro que TensorFlow tiene una mayor comunidad de desarrolladores.

Aparte de los factores anteriores, debes saber que Tensorflow también proporciona soporte para Keras. Tensorflow proporciona el submódulo tf.keras que le permite soltar el código de Tensorflow directamente en los modelos de Keras. Puedes obtener características tanto de Keras como de Tensorflow usando tf.keras, es decir, puedes obtener lo mejor de ambos mundos.

V.3 Keras Vs TensorFlow - Diferencia entre Keras y Tensorflow

El siguiente código describe cómo utilizar tf.keras para crear sus modelos:

```
1.  import tensorflow as tf
2.  from tensorflow.keras import layers
3.
4.  model= tf.keras.Sequential()
5.  model.add(layers.Dense(64,activation='relu'))
6.  model.add(layers.Dense(32,activation='relu'))
7.  model.add(layers.Dense(10))
```


VI. Referencias

Documentos

[1] L. Xu, F. Cai, Y. Hu, Z. Lin, and Q. Liu, "Using deep learning algorithms to perform accurate spectral classification," Optik, vol. 231, p. 166423, Apr. 2021, doi: 10.1016/j.ijleo.2021.166423.

[2] J. Gordon and J. M. Hernández-Lobato, "Combining deep generative and discriminative models for Bayesian semi-supervised learning," Pattern Recognit., vol. 100, p. 107156, Apr. 2020, doi: 10.1016/j.patcog.2019.107156.

[3] L. Zeng et al., "Deep learning trained algorithm maintains the quality of half-dose contrast-enhanced liver computed tomography images: Comparison with hybrid iterative reconstruction: Study for the application of deep learning noise reduction technology in low dose," Eur. J. Radiol., vol. 135, p. 109487, Feb. 2021, doi: 10.1016/j.ejrad.2020.109487.

[4] S. Khan, N. Islam, Z. Jan, I. Ud Din, and J. J. P. C. Rodrigues, "A novel deep learning based framework for the detection and classification of breast cancer using transfer learning," Pattern Recognit. Lett., vol. 125, pp. 1–6, Jul. 2019, doi: 10.1016/j.patrec.2019.03.022.

VI. Referencias

[5] Y. He, P. Wu, Y. Li, Y. Wang, F. Tao, and Y. Wang, "A generic energy prediction model of machine tools using deep learning algorithms," Appl. Energy, vol. 275, p. 115402, Oct. 2020, doi: 10.1016/j.apenergy.2020.115402.

[6] M. Jiang, J. Liu, L. Zhang, and C. Liu, "An improved Stacking framework for stock index prediction by leveraging tree-based ensemble models and deep learning algorithms," Phys. Stat. Mech. Its Appl., vol. 541, p. 122272, Mar. 2020, doi: 10.1016/j.physa.2019.122272.

[7] D. Kißkalt, A. Mayr, B. Lutz, A. Rögele, and J. Franke, "Streamlining the development of data-driven industrial applications by automated machine learning," Procedia CIRP, vol. 93, pp. 401–406, Jan. 2020, doi: 10.1016/j.procir.2020.04.009.

[8] Y. Chen, X. Zou, K. Li, K. Li, X. Yang, and C. Chen, "Multiple local 3D CNNs for region-based prediction in smart cities," Inf. Sci., vol. 542, pp. 476–491, Jan. 2021, doi: 10.1016/j.ins.2020.06.026.

VI. Referencias

[9] T. D. Akinosho et al., “Deep learning in the construction industry: A review of present status and future innovations,” J. Build. Eng., vol. 32, p. 101827, Nov. 2020, doi: 10.1016/j.jobe.2020.101827.

[10] M.-A. Zamora-Hernández, J. A. Castro-Vargas, J. Azorin-Lopez, and J. Garcia-Rodriguez, “Deep learning-based visual control assistant for assembly in Industry 4.0,” Comput. Ind., vol. 131, p. 103485, Oct. 2021, doi: 10.1016/j.compind.2021.103485.

[11] R. Espinosa, H. Ponce, and S. Gutiérrez, “Click-event sound detection in automotive industry using machine/deep learning,” Appl. Soft Comput., vol. 108, p. 107465, Sep. 2021, doi: 10.1016/j.asoc.2021.107465.

[12] J. Leng et al., “A loosely-coupled deep reinforcement learning approach for order acceptance decision of mass-individualized printed circuit board manufacturing in industry 4.0,” J. Clean. Prod., vol. 280, p. 124405, Jan. 2021, doi: 10.1016/j.jclepro.2020.124405.

VI. Referencias

[13] M. Mishra, J. Nayak, B. Naik, and A. Abraham, “Deep learning in electrical utility industry: A comprehensive review of a decade of research,” Eng. Appl. Artif. Intell., vol. 96, p. 104000, Nov. 2020, doi: 10.1016/j.engappai.2020.104000.

[14] P. Tripicchio and S. D’Avella, “Is Deep Learning ready to satisfy Industry needs?,” Procedia Manuf., vol. 51, pp. 1192–1199, Jan. 2020, doi: 10.1016/j.promfg.2020.10.167.

[15] R. Oberleitner and J. Schwartz, “5.29 Integrating Deep Learning With Behavior Imaging to Accelerate Industry Learning of Autism Core Deficits,” J. Am. Acad. Child Adolesc. Psychiatry, vol. 56, no. 10, Supplement, p. S263, Oct. 2017, doi: 10.1016/j.jaac.2017.09.312.

[16] T. Kotsiopoulos, P. Sarigiannidis, D. Ioannidis, and D. Tzovaras, “Machine Learning and Deep Learning in smart manufacturing: The Smart Grid paradigm,” Comput. Sci. Rev., vol. 40, p. 100341, May 2021, doi: 10.1016/j.cosrev.2020.100341.

VI. Referencias

[17] C. Yang, H. Lan, F. Gao, and F. Gao, "Review of deep learning for photoacoustic imaging," Photoacoustics, vol. 21, p. 100215, Mar. 2021, doi: 10.1016/j.pacs.2020.100215.

[18] L. Zhu, P. Spachos, E. Pensini, and K. N. Plataniotis, "Deep learning and machine vision for food processing: A survey," Curr. Res. Food Sci., vol. 4, pp. 233–249, Jan. 2021, doi: 10.1016/j.crfs.2021.03.009.

[19] X. Xu, J. Wang, B. Zhong, W. Ming, and M. Chen, "Deep learning-based tool wear prediction and its application for machining process using multi-scale feature fusion and channel attention mechanism," Measurement, vol. 177, p. 109254, Jun. 2021, doi: 10.1016/j.measurement.2021.109254.

[20] S. Shajun Nisha, M. Mohamed Sathik, and M. Nagoor Meeral, "3 - Application, algorithm, tools directly related to deep learning," in Handbook of Deep Learning in Biomedical Engineering, V. E. Balas, B. K. Mishra, and R. Kumar, Eds. Academic Press, 2021, pp. 61–84. doi: 10.1016/B978-0-12-823014-5.00007-7.

VI. Referencias

Libros

- https://www.pdfdrive.com/introduction-to-deep-learning-using-r-a-step-by-step-guide-to-learning-and-implementing-deep-learning-models-using-r-e158252417.html
- https://www.pdfdrive.com/learn-keras-for-deep-neural-networks-a-fast-track-approach-to-modern-deep-learning-with-python-e185770502.html
- https://www.pdfdrive.com/applied-deep-learning-a-case-based-approach-to-understanding-deep-neural-networks-e176380114.html
- https://www.pdfdrive.com/deep-learning-adaptive-computation-and-machine-learning-e176370174.html
- https://www.pdfdrive.com/deep-learning-in-python-master-data-science-and-machine-learning-with-modern-neural-networks-written-in-python-theano-and-tensorflow-e196480537.html
- https://www.pdfdrive.com/deep-learning-with-python-e54511249.html
- https://www.pdfdrive.com/learning-tensorflow-a-guide-to-building-deep-learning-systems-e158557113.html
- https://www.pdfdrive.com/deep-learning-with-applications-using-python-chatbots-and-face-object-and-speech-recognition-with-tensorflow-and-keras-e184016771.html
- https://www.pdfdrive.com/mastering-machine-learning-with-python-in-six-steps-a-practical-implementation-guide-to-predictive-data-analytics-using-python-e168776616.html
- https://hackr.io/blog/artificial-intelligence-books

VI. Referencias

Tutoriales

- https://www.fast.ai
- https://www.coursera.org/learn/machine-learning
- https://www.coursera.org/specializations/deep-learning
- https://www.udemy.com/course/machinelearning/
- https://www.edx.org/professional-certificate/harvardx-data-science
- https://www.udacity.com/course/intro-to-machine-learning-nanodegree--nd229
- https://online.stanford.edu/courses/cs229-machine-learning
- https://www.edx.org/learn/machine-learning
- https://learn.datacamp.com/courses/introduction-to-machine-learning-with-r

VI. Referencias

Charlas y Webinars

- https://www.brighttalk.com/topic/deep-learning/
- https://www.dataiku.com/webinars/

COMPARTE Y VERIFICA TUS LOGROS DE APRENDIZAJE FÁCILMENTE

#CAIEC #certiprof

certiprof®

¡Síguenos, ponte en contacto!

www.certiprof.com