

Machine Learning Cookbook

1 MACHINE LEARNING BLUEPRINT

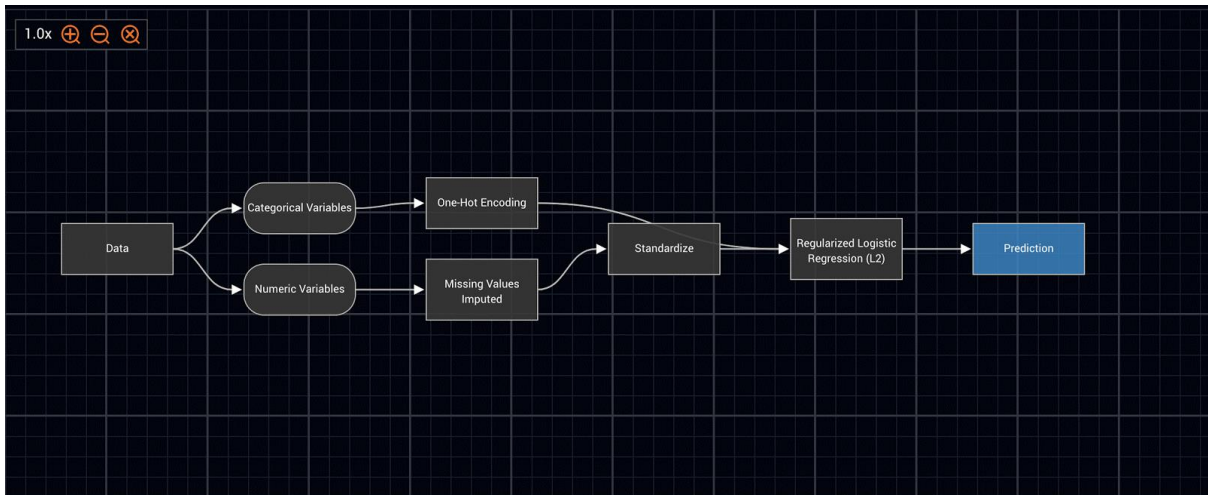


Figure 1: Basic flow of a Machine Learning Model

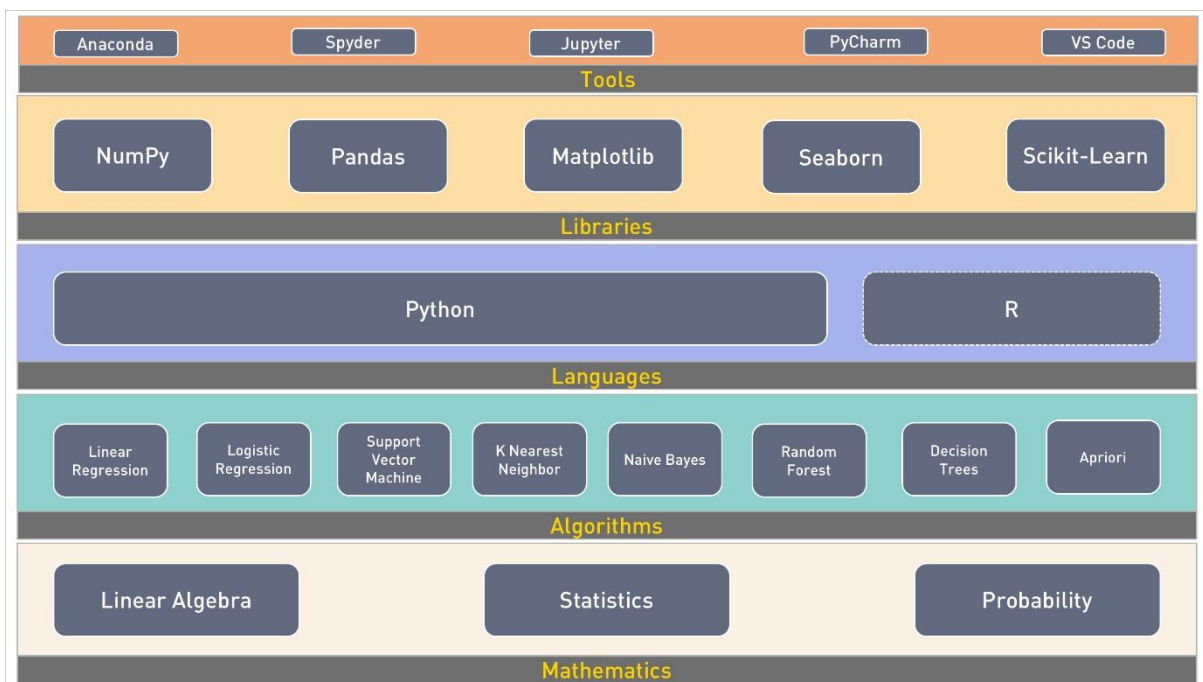


Figure 2: Blueprint for Machine Learning

2 10 STAGES OF A MACHINE LEARNING PROJECT

2.1 Problem Definition

Problem definition is the initial stage of a Computer Vision/ML project, and it focuses on gaining an understanding of the problem poised to be solved by applying ML.

It usually involves a problem descriptor that records, in a selected form, a scenario-based description of first-hand experience of an encounter of the problem to be solved.

This stage also captures what an ideal solution to a problem will be from the problem descriptor's perspective.

A problem descriptor can be clients, customers, users or colleagues.

2.1.1 Deliverables

The deliverable for this stage is a document (word or pdf) with the following included (but not limited to):

1. Problem Statement
2. Ideal Problem Solution
3. Understanding and insight into the problem
4. Technical requirements

2.2 RESEARCH

This stage sets the foundation for later stages, along with the planning of implementation and development work carried out within subsequent stages.

An exploration into the form a solution will take is conducted, along with information into the data structures, formats, and sources.

A combination of an understanding of the problem, unified with proposed solutions, and available data, will enable a suitable ML model selection process to achieve the ideal solution result.

At this stage, it is helpful to research the hardware and software requirements for the algorithms and model implementation; this saves a lot of time in later stages.

2.2.1 Deliverables

The deliverable for this stage is a document (word or pdf) with research into the following included:

1. Data Structure and Source
2. Solution form
3. Neural Network / Model Architecture
4. Algorithm Research
5. Hardware Requirements
6. Software Requirements

2.3 DATA AGGREGATION/ MINING / SCRAPING

Data is the fuel for an ML/CV application. Data aggregation is a crucial step that sets a precedent for the effectiveness and performance of the trained model.

The output of the agreed-upon solution defines the data aggregated.

Data understanding is paramount, and any sourced data should be examined and analysed utilizing visualization tools or statistical methods.

Data examination promoted data integrity and credibility by ensuring the data sourced is the expected data.

Data analysis and exploration carried on the data also ensure the following requirements are met:

- The data gathered needs to be diverse enough to ensure that the model predictions capabilities accommodate a variety of possible scenarios.
- The data gathered needs to aspire to be unbiased to ensure that the model can generalize appropriately during inference.
- The data gathered needs to be abundant.

Tools for collecting data will vary. Data sources could come in the form of APIs, XML feeds, CSV, or Excel files. In some scenarios, data mining/scraping from online sources is required. Ensure to check on third party websites scraping/mining policies before conducting scrapes.

2.3.1 Deliverable

The deliverable for this stage is a folder with raw source data along with annotation files within each subfolder.

2.4 DATA PREPARATION / PRE-PROCESSING / AUGMENTATION

Pre-processing steps for data are based mainly on the model input requirements. Refer to the research stage and recall input parameters and requirements that the selected model / neural network architecture requires.

The pre-processing step transforms the raw sourced data into a format that enables successful model training.

Data pre-processing could include the identified steps below, but not limited to the mentioned steps:

- Data Reformatting (resizing images, modification to colour channels, noise reduction, image enhancement)
- Data Cleaning
- Data Normalisation

Data augmentation is a step that is carried out to improve the diversification of data that has been sourced. Augmentation of image data could take the following forms:

- Rotation of an image by any arbitrary degrees
- Scaling of an image either to create zoomed in/out effects
- Cropping of an image
- Flipping (horizontal or vertical) of an image
- Mean Subtraction

2.4.1 Deliverable

The deliverable for this stage is a folder with subfolders labelled train, test, and validation along with annotation files within each subfolder.

2.5 MODEL IMPLEMENTATION

Typically, model implementation is simplified by leveraging existing models that are available from a variety of online sources. Most ML/DL framework such as [PyTorch](#) or [TensorFlow](#), have pre-trained models that are leveraged to speed up the model implementation stage.

These pre-trained models have been trained on robust datasets and mimic the state-of-the-art neural network architectures' performance and structure.

You rarely must implement a model from scratch. The following might be expected to be conducted during the model implementation stage:

- Removal of last layers within a neural network to repurpose models for specific tasks. For example, removing the last layer of a Resnet neural network architecture enables the utilization of a descriptor provided by the model within an encoder-decoder neural network architecture
- Fine-tuning pre-trained models

2.5.1 Deliverable

The deliverable for this stage is a model that is ready to be trained.

2.6 TRAINING

The training data delivered from the previous Data stages are utilized within the training stage. The implementation of model training involves passing the refined aggregated training data through the implemented model to create a model that can perform its dedicated task well.

The training of the implemented model involves iteratively passing mini batches of the training data through the model for a specified number of epochs. During the early stages of training, model performance and accuracy can be very unimpressive. Still, as the model conducts predictions and a comparison of predicted values is made to the desired/target value, backpropagation takes place within the neural networks, the model begins to improve and gets better at the task it's designed and implemented to do.

Just before training can commence, we have to set hyperparameters and network parameters that will steer the effectiveness of our training stage on the model.

Hyperparameters: These are values that are defined before the training of the network begins; they are initialized to help steer the network to a positive training outcome. Their effect is on the machine / deep learning algorithm, but they are not affected by the algorithm. Their values do not change during training. Examples of hyperparameters are regularization values, learning rates, number of layers, etc.

Network parameter: These are components of our network that are not manually initialized. They are embedded network values that are manipulated by the network directly. An example of a network parameter is the weights internal to the network.

When conducting training, it is vital to ensure that metrics are recorded of each training process and at each epoch. The metrics that are generally collected are the following:

- Training accuracy
- Validation accuracy
- Training Loss

- Validation Loss

To collate and visualize training metrics, tools such as visualization tools Matplotlib and Tensorboard can be utilized.

By visualizing the training metrics, it is possible to identify some common ML model training pitfalls, such as underfitting and overfitting.

- **Underfitting:** This occurs when a machine learning algorithm fails to learn the patterns in a dataset. Underfitting can be fixed by using a better algorithm or model that is more suited for the task. Underfitting can also be adjusted fixed by recognizing more features within the data and presenting it to the algorithm.
- **Overfitting:** This problem involves the algorithm predicting new instances of patterns presented to it, based too closely on instances of patterns it observed during training. This can cause the machine-learning algorithm to not generalize accurately to unseen data. Overfitting can occur if the training data does not accurately represent the distribution of test data. Overfitting can be fixed by reducing the number of features in the training data and reducing the complexity of the network through various techniques.

2.6.1 Deliverable

The deliverable for this stage is a developed model and training metrics

2.7 EVALUATION

At this stage, you should have a trained model and are ready to conduct evaluation techniques on its performance.

For evaluation, we utilize a partition of the refined data, usually referred to as the ‘test data’. The test data have not been seen during the model during training. They are also representative of examples of data that are expected to be encountered in practical scenarios.

Some examples of evaluation strategies that can be leveraged are as follows:

- **Confusion matrix (error matrix):** Provides a visual illustration of the number of matches or mismatches the annotation of the ground truth to the classifier results. A confusion matrix is typically structured in tabular form, where the rows are filled with the observational results from the ground-truth, and the columns are filled with inference results from the classifier.
- **Precision-Recall:** These are performance metrics that are used to evaluate classification algorithms, visual search systems, and more. Using the example of evaluating a visual search

system(find similar images based on a query image), precision captures the number of results returned that are relevant, while recall captures the number of relevant results in your dataset that are returned.

2.7.1 Deliverables

The deliverable for this stage is a document containing the evaluation results, and evaluation strategies outputs are also included.

2.8 PARAMETER TUNING AND INFERENCE

Parameter tuning is the process of model refinement that is conducted by making modifications to hyperparameter values. The purpose of parameter tuning is to increase the model performance, and this correlates to improvements in evaluation results.

Once hyperparameters are tuned and new values are selected, training and evaluation commence again.

The process of parameter tuning is carried out until a suitable enough model is generated.

Inference is a real-world test of our model. It involves utilizing real-world data that have been sourced from applicable environments. At this stage, we should be confident in our model performance.

2.8.1 Deliverable

The deliverable for this stage is a refined model.

2.9 MODEL CONVERSION TO AN APPROPRIATE FORMAT

Once we have our refined model, we are ready to place it on devices where it can be utilized.

Model conversion is a step that is required when developing models that are to be used within edge devices such as mobile phones or IoT devices.

Model conversion involves ML models trained in a GPU/CPU environment and converting them into an optimized and efficient version. The streamlined model is small enough to be stored on devices and sufficiently accurate to conduct suitable inference.

Examples of tools that enable model conversion to the mobile-optimized model are:

- [Core ML](#): This is a framework released by Apple to create iOS only dedicated models. CoreML provides some models for common machine learning tasks such as recognition and detection. It's an iOS-only alternative to TensorFlow Lite.

- [PyTorch Mobile](#): PyTorch is a popular machine learning framework and is used extensively in machine learning-related research. PyTorch mobile can be compared to TensorFlow Lite, as it enables the conversion of PyTorch trained model to a mobile-optimized version that can be leveraged on iOS and Android devices. Although, PyTorch Mobile is still in its infancy and currently in experimental release status.
- [TensorFlow Lite](#): takes existing TensorFlow models and converts them into an optimized and efficient version in the form of a tflite file. The streamlined model is small enough to be stored on devices and sufficiently accurate to conduct suitable inference.

2.9.1 Deliverable

The deliverable for this stage is an ML model that has been optimized for on-device usage.

2.10 MODEL DEPLOYMENT

Deploying our final trained model is the last step within all the identified stages. Integrating our model within a broader ecosystem of application or tool, or simply building an interactive web interface around our model, is an essential step of model deployment.

There is also a monitoring responsibility that should be undertaken to assess the performance of the model while in a production environment. This is to ensure that the model is performing sufficiently well, and it still fit for purpose.

Model retraining and updating is also a process within the model deployment stage. Model updating ensures the credibility and reliability of our model for the desired task.

2.10.1 Deliverables

The deliverables for this stage could be the following:

1. Model performance monitoring system
2. Web UI Interface to access model functionalities
3. Continuous integration pipelines that enable model redeployment

3 TOP OPEN SOURCE AI TECHNOLOGIES IN MACHINE LEARNING USED BY US

-  [TensorFlow](#)
-  [Keras](#)
-  [Scikit-Learn](#)
-  [Microsoft Cognitive Toolkit](#)
-  [Caffe](#)
-  [PyTorch](#)
-  [Darknet](#)
-  [MXNet](#)

4 MODEL DEPLOYMENT EXAMPLES

There is no one-model fits all ideology behind deploying machine learning models. Thereby this section is being left open-ended for the students to explore. A couple of examples and a list of technologies has been discussed as follows.

4.1 EXAMPLES OF DEPLOYMENT

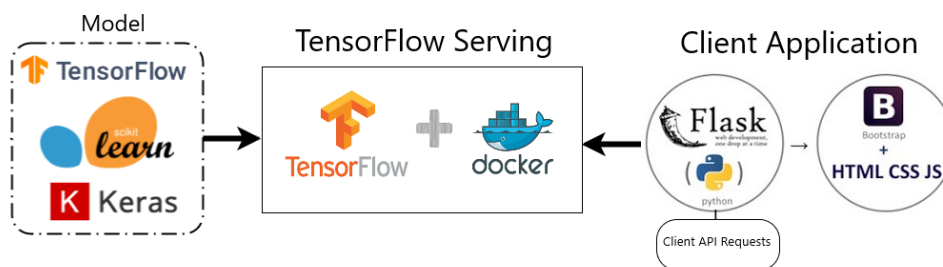


Figure 3: Standard TensorFlow Serving based deployment

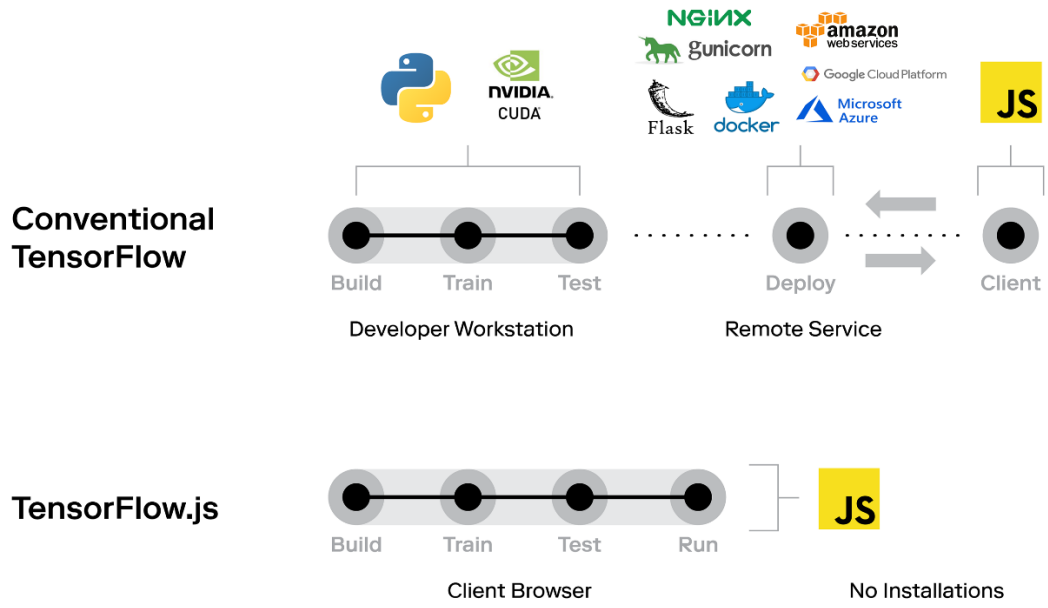


Figure 4: TensorFlow vs TensorFlow JS

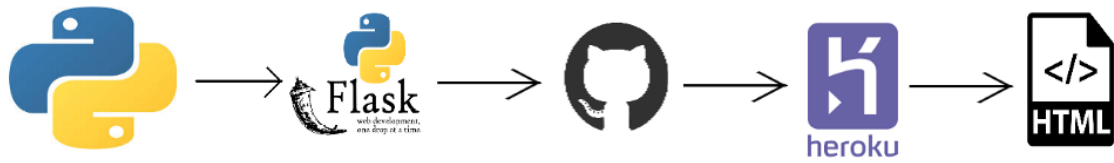


Figure 5: Deploying using Heroku and Flask

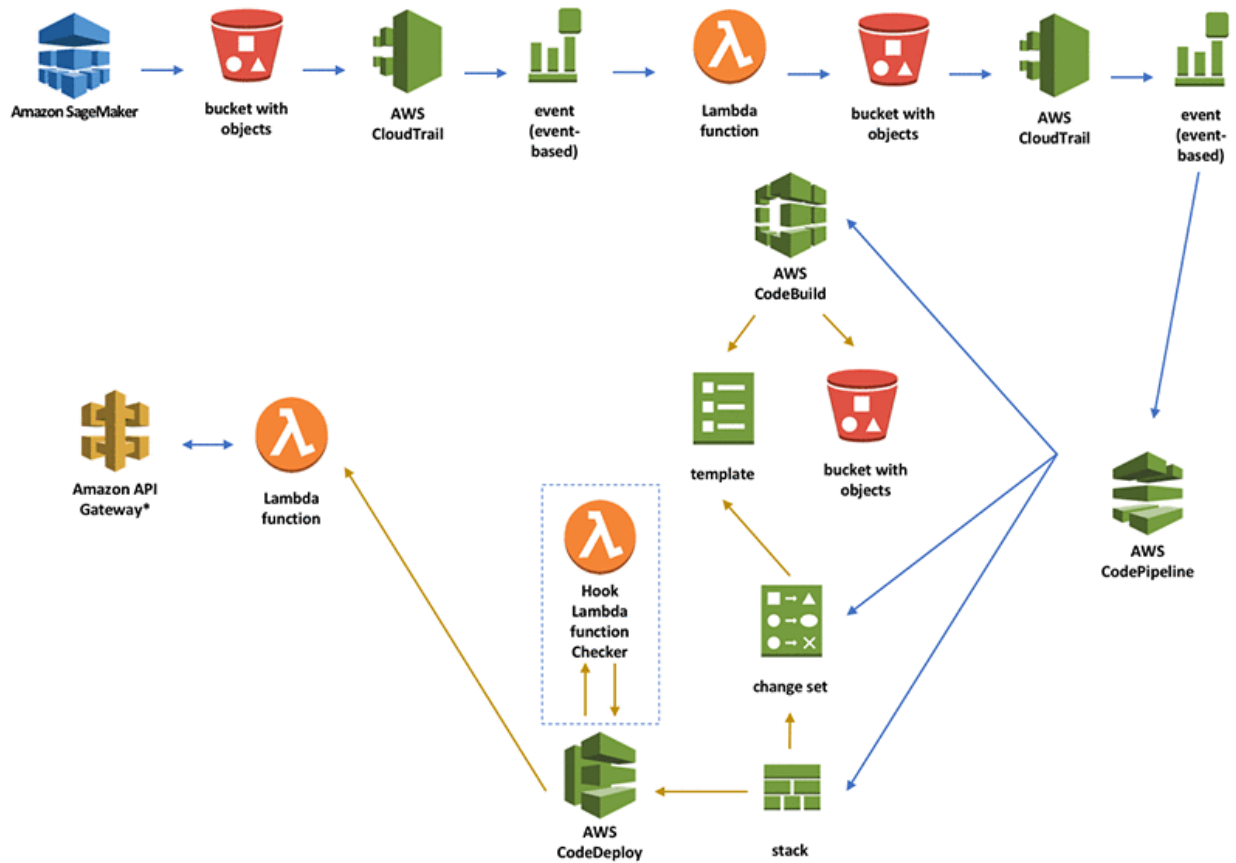







Figure 6: Build, Test and Deploy using Amazon Sage Maker

4.2 TECHNOLOGIES INVOLVED

- ✚ [Docker](#) / [Virtual machine](#)
- ✚ TensorFlow Serving
- ✚ PyTorch Serving
- ✚ [Flask](#) / [Django](#)
- ✚ [Angular](#) / [React](#) and [Node](#)
- ✚ Various cloud platforms for pre-baked solutions

Setting up your Machine learning environment

5 ON CLOUD

-  [Gradient by Paperspace](#)
-  [Collaboratory by Google](#)
-  [Kaggle Kernels by Kaggle](#)
-  [FloydHub Kernels](#)
-  [Deepcognition.ai](#)

6 LOCAL MACHINE

Most of the project these days can be run on a local machine unless we ought to design something that outperforms the current state of art. This is possible using the concept called **Transfer Learning** (Must read).

In order to set up your local machine for various Machine Learning/ Deep Learning projects kindly ensure that the following conditions are met. If you feel you are lagging in either of the aspects as discussed below, we highly recommend that you use one of the free services listed in the category above.

So, the following is the recommended configuration:

1. Central Processing Unit (CPU) — Intel Core i3 8th Generation or i5 6th Generation processor or higher. An AMD equivalent processor will also be optimal.
2. RAM — 8 GB minimum, 16 GB or higher is recommended.
3. Graphics Processing Unit (GPU) — NVIDIA GeForce GTX 960 or higher. AMD GPUs are not able to perform deep learning regardless. For more information on NVIDIA GPUs for deep learning please visit <https://developer.nvidia.com/cuda-gpus>.
4. Operating System — Ubuntu (Any LTS Version) or Microsoft Windows 10. We recommend updating Windows 10 to the latest version before proceeding forward.

Note: You could refer to the guided setup [here](#). If you are facing any issues kindly let us know, our team shall be assisting you with that. We also recommend using Docker for running the environment on a local machine. For instructions refer to the this [link](#).

Reference Materials

- ✚ [Collection of random sources \(Medium\)](#): This is a medium blogpost that contains a variety of resources that one can refer to regarding machine learning and deep learning.
- ✚ [Machine Learning in Python](#)
- ✚ [Introduction of Machine Learning Stack](#)
- ✚ [Datasets – Kaggle](#)
- ✚ [Datasets – UCI](#)
- ✚ [The 50 Best Free Datasets for Machine Learning](#)
- ✚ [Free MOOC by Fast.AI on Deep Learning](#)
- ✚ [Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp by Peter Norvig \(1992\), with Code](#)
- ✚ [Machine Learning crash course – Google](#)
- ✚ [Open source machine learning tools](#)
- ✚ [Python Basics to Advanced – Guided course](#)
- ✚ [MIT Open courseware – Artificial Intelligence](#)
- ✚ [Neural Networks and Deep Learning – Informative read](#)
- ✚ [Neural networks class at University of Sherbrooke](#)
- ✚ [The Microsoft Cognitive Toolkit](#)
- ✚ [Understanding Machine Learning, © 2014 by Shai Shalev-Shwartz and Shai Ben-David - PDF](#)
- ✚ [Play with neural networks in the browser – A better intuitive understanding](#)
- ✚ [Computer vision, Deep Learning and OpenCV](#)
- ✚ The following books shall be found on [Google Drive Link](#):
 - Advanced Applied Deep Learning Convolutional Neural Networks and Object Detection by Umberto Michelucci
 - Programming Collective Intelligence by O'Reilly
 - Kubernetes Cookbook
 - Hands-On Neuro-evolution with Python: Build high-performing artificial neural network architectures using neuro-evolution-based algorithms by Iaroslav Omelianenko
 - Programming PyTorch for Deep Learning by O'Reilly
 - Machine Learning for Cybersecurity Cookbook: Over 80 recipes on how to implement machine learning algorithms for building security systems using Python by Emmanuel Tsukerman
 - Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow by Aurelien Geron

References for Recommendation

Systems

- ✚ [A Collaborative Location Based Travel Recommendation System through Enhanced Rating Prediction for the Group of Users](#)
- ✚ [A Machine Learning Approach — Building a Hotel Recommendation Engine](#)
- ✚ [A recommendation engine for travel products based on topic sequential patterns](#)
- ✚ [Data Science and AI in the Travel Industry: 12 Real-Life Use Cases](#)
- ✚ [A recommendation engine for travel products based on topic sequential patterns](#)
- ✚ [A simple way to explain the Recommendation Engine in AI](#)
- ✚ [Intelligent Travel Recommendation System by Mining Attributes from Community Contributed Photos](#)
- ✚ [GuideMe-A Tourist Guide with a Recommender System and Social Interaction](#)
- ✚ [How Recommendation Engines Actually Work – Strategies and Principles](#)
- ✚ [Smart Itinerary Recommendation based on User-Generated GPS Trajectories](#)
- ✚ [A recommender system for tourism industry using cluster ensemble and prediction machine learning techniques](#)
- ✚ [Kaggle – Recommender Systems](#)
- ✚ [Tourist Recommender Systems](#)
- ✚ [9 Must-Have Datasets for Investigating Recommender Systems](#)
- ✚ [LightFM Hybrid Recommendation system](#)
- ✚ [Powered by AI: Instagram’s Explore recommender system](#)
- ✚ [RecSys - Basics](#)
- ✚ [High-performing Travel Recommendation Engine built with AI/ML models](#)
- ✚ [Evolving OYO’s Ranking Systems using Wide and Deep Networks](#)
- ✚ [Using Machine Learning to Improve Customer Retention](#)
- ✚ [Image Similarity Detection in Action with Tensorflow 2.0](#)
- ✚ [Building a Personalized Real-Time Fashion Collection Recommender](#)
- ✚ [Predicting Hotel Cancellations with Machine Learning](#)
- ✚ [Shortest Path Distance with Deep Learning](#)
- ✚ [BERT NLP—How To Build a Question Answering Bot](#)
- ✚ [Video Content-Based Advertisement Recommendation using NLP](#)
- ✚ [Creating a production ready recommender system](#)
- ✚ [Deep Dive into Netflix’s Recommender System](#)
- ✚ [Incremental Recommender System](#)
- ✚ [Book Recommendation System](#)
- ✚ [A guide to Collaborative Topic Modelling recommender systems - Theory and implementation of a recommender system with out-of-matrix prediction capabilities.](#)
- ✚ [How to build a content-based movie recommender system with Natural Language Processing](#)