VelocitySignals Tool: Technical Notes on Predictive Models and Market Indicators

by Breno Niero

Introduction

Welcome to the VelocitySignals tool, where I combine traditional technical analysis with advanced machine learning to predict stock movements. The tool provides two primary methods for generating predictions: condition matching and machine learning models.

The condition-matching method relies on well-known technical indicators like RSI (Relative Strength Index), MACD (Moving Average Convergence Divergence), and Volume to generate buy, sell, or hold signals. The machine learning models, on the other hand, utilize Random Forest (RF) and Long Short-Term Memory (LSTM) networks to analyze historical stock data and make predictions based on patterns in past movements.

The choice of Random Forest comes from its robustness in handling complex, non-linear relationships and its resistance to overfitting (Breiman, 2001). It's a highly effective tool for analyzing market data with various noisy features. LSTM, on the other hand, excels in processing sequential data, making it ideal for analyzing time series data like stock prices (Abu-Khalaf et al., 2018). Together, these two approaches give me a comprehensive toolset for predicting future market behavior.

Predictive Models

Let's dive into how each one works.

Condition-Matching Method

The **condition-matching** method relies on well-established technical indicators to generate trading signals. It looks at key metrics like **RSI**, **MACD**, and **Volume** and compares them to predefined thresholds. Here's how I apply these indicators.

RSI

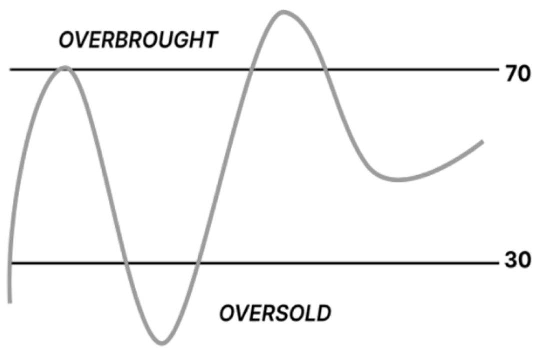
RSI is a momentum indicator that measures the speed and change of price movements. It helps identify whether a stock is overbought or oversold, which can indicate potential trading opportunities. The formula for RSI is:

$$RSI = 100 - \left(\frac{100}{1 + RS}\right)$$

where **RS** is the ratio of average gains to average losses over a specific period (Wilder, 1978).

$$RS = \frac{\text{Average Gain over } n \text{ periods}}{\text{Average Loss over } n \text{ periods}}$$

Typically, when RSI is above 70, it signals that the stock might be overbought, and when RSI is below 30, it suggests that the stock could be oversold.



MACD

MACD helps identify changes in the momentum of a stock's price. It's calculated by subtracting the 26-period **Exponential Moving Average (EMA)** from the 12-period EMA:

$$MACD = EMA_{12} - EMA_{26}$$

$$EMA = \text{Price}_{\text{today}} \times \left(\frac{2}{N+1}\right) + EMA_{\text{yesterday}} \times \left(1 - \frac{2}{N+1}\right)$$

The resulting MACD line is then compared to a **Signal Line**, which is a 9-period EMA of the MACD itself. If the MACD crosses above the signal line, it generates a buy signal. If it crosses below, it generates a sell signal (Achelis, 2001).

Volume

Volume is used to validate the strength of the price movements. If a high volume accompanies a price movement, it suggests that the trend is strong. On the other hand, if the price moves with low volume, the trend might not be reliable.

While volume isn't directly used in mathematical formulas, I use it as a filter in the decision-making process:

- **High volume** confirms price trends.
- **Low volume** might indicate weakness in the movement, leading to a **hold** or cautious signal.

Signal Generation

I classify the stock movement into one of three categories, factoring in Volume to confirm the strength of the signal:

- **Buy**: RSI < 30, MACD > Signal Line, and Volume above the median or a custom threshold (indicating strong buying interest).
- **Sell**: RSI > 70, MACD < Signal Line, and Volume above the median or a custom threshold (indicating strong selling interest).
- **Hold**: Neither buy nor sell conditions are met, or Volume is below the threshold, signaling weak momentum.

These signals are fed into Tableau, where I visualize buy, sell, or hold signals, enabling me to track opportunities and make fast trading decisions.

The condition-matching method compares these indicators to predefined thresholds to generate buy, sell, or hold signals. For instance, if the RSI is below 30, MACD crosses above the signal line, and the volume is high, I generate a buy signal.

Machine Learning Predictions

In addition to condition matching, I use machine learning models to enhance the predictive power of **VelocitySignals Lab**. These models are trained on historical stock data and can capture more complex patterns than rule-based systems alone.

Random Forest

Random Forest (RF) is an ensemble learning method that combines the predictions of multiple decision trees to produce a more accurate result. Each decision tree is trained on a random subset of the data, and the final prediction is made by averaging the predictions of all the trees. This method reduces overfitting and increases the model's ability to generalize to unseen data (Breiman, 2001).

$$P(\text{Buy}|\text{RSI}, \text{MACD}, \text{Volume}) = \frac{1}{n}\sum_{i=1}^{n} T_i(\text{RSI}, \text{MACD}, \text{Volume})$$

For stock prediction, I use indicators like **RSI**, **MACD**, **EMA**, and **Volume** as features for training the Random Forest model. By analyzing these features, the model can predict whether the next movement will be a **buy**, **sell**, or **hold** signal. The mathematical formula behind Random Forest involves aggregating predictions from multiple decision trees:

The final prediction is determined by the majority vote of the trees (Breiman, 2001).

Long Short-Term Memory (LSTM)

LSTM networks are a type of recurrent neural network (RNN) designed to handle sequential data. They are ideal for stock price prediction because they can remember patterns from past stock movements and use them to predict future prices (Abu-Khalaf et al., 2018).

The key feature of LSTM networks is their memory cells, which allow them to maintain information over time. These cells use three gates—input, forget, and output gates—to control the flow of information through the network. The cell state and hidden state are updated at each time step based on the current input and previous states:

$$h_t = \sigma(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Where:

- ht is the hidden state at time step tt,
- Ct is the cell state,
- it , ft , and ot are input, forget, and output gates, respectively,
- Cˆt is the candidate cell state.

This architecture enables LSTMs to model long-term dependencies in time series data, making them particularly effective for stock prediction. By training the LSTM on historical data, I can predict future movements based on past trends (Liu et al., 2018).

Data Flow

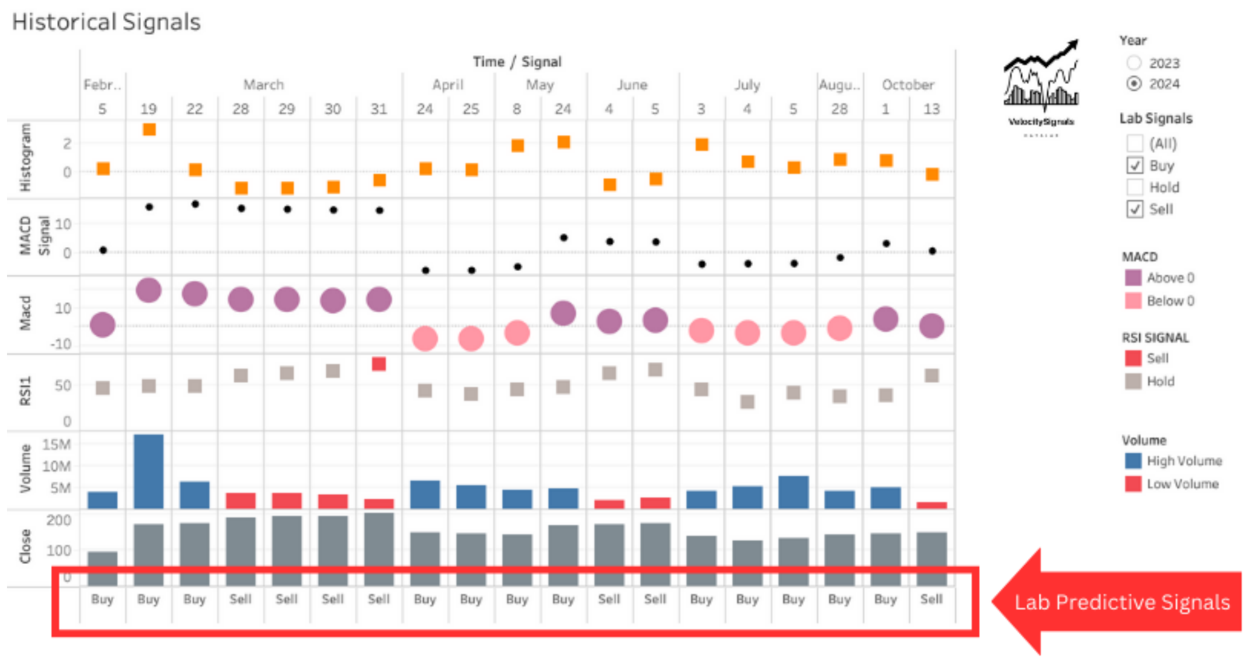
Here's how data moves through **VelocitySignals**.

First, I ingest the data from **TradingView** using an API or CSV files. The dataset includes pre-calculated technical indicators such as **RSI**, **MACD**, and **Volume**, which are essential for both condition matching and machine learning models.

Once the data is ingested, it is processed through **Google Cloud Functions** and stored in **BigQuery**. At this stage, I verify and recalculate the necessary technical indicators to ensure that the data is accurate and ready for analysis.

The next step is the **signal generation** process. For the condition-matching approach, I evaluate the **RSI**, **MACD**, and **Volume** values against predefined thresholds to generate buy, sell, or hold signals. For the machine learning approach, I use the processed data to train the **Random Forest** and **LSTM** models, which then predict the next movement in the market based on historical trends.

Finally, the signals are sent to **Tableau**, where I monitor them in real-time. The visualization of buy, sell, or hold signals in Tableau helps me make informed trading decisions, especially when the market is volatile.



Machine Learning Testing Phase

To ensure that the machine learning models perform well, I test them extensively using real-world data. In this case, I'm working with a dataset from **Binance**, specifically the **SOL/USDT** trading pair. This dataset includes daily OHLC prices, technical indicators like **RSI**, **MACD**, and **Volume**, as well as other metrics like **Bollinger Bands**.

Example of the **SOL/USDT** trading pair Dataset:

time,open,high,low,close,MA,EMA,Basis,Upper,Lower,Volume,RSI,Upper Bollinger Band,Lower Bollinger Band,Regular Bullish,Regular Bullish Label,Regular Bearish,Regular Bearish Label,Histogram,MACD,Signal
2023-09-09,19.6,19.61,19.38,19.44,19.643333333333333,19.815438424414644,20.289499999999997,21.78542078667304,18.793579213326954,646379.52,38.136669458305725,NaN,NaN,NaN,NaN,NaN,NaN,0.029599799805416382,-0.7786095425647765,-0.8082093423701929
2023-09-10,19.43,19.44,17.78,18.24,19.522222222222222,19.500350739531715,20.141499999999994,21.822110305811805,18.460889694188182,5630975.71,23.84162343331026,NaN,NaN,NaN,NaN,NaN,NaN,-0.04050862188739568,-0.8588451197294376,-0.8183364978420419

2023-09-11,18.23,18.46,17.33,17.72,19.324444444444442,19.14428059162537,19.997999999999994,21.966396301561453,18.029603698438535,4755172.77,19.952561966859406,NaN,NaN,19.952561966859406,NaN,NaN,NaN,-0.10805226102299481,-0.9534018241207853,-0.8453495630977905
2023-09-12,17.72,18.8,17.57,17.92,19.141111111111111,18.899424473300297,19.806999999999995,21.803337646792443,17.810662353207547,4818249.11000001,25.5571041230028,NaN,NaN,NaN,NaN,NaN,NaN,-0.12425260405745797,-1.000665318169613,-0.876412714112155
2023-09-13,17.91,18.56,17.69,18.4,19.016666666666666,18.799539578640236,19.673999999999996,21.673133812429853,17.67486618757014,4063710.36000001,38.049123437615656,NaN,NaN,NaN,NaN,NaN,NaN,-0.08927057234169056,-0.9880009295392682,-0.8987303571975777

Data Preprocessing

The first step in the testing phase is to clean and preprocess the dataset. This includes handling missing values, removing unnecessary columns, and normalizing the features so that they all fall within a similar range. This makes the model training process more efficient.

I also perform feature engineering by creating additional features that capture important patterns. For example, I might introduce lagged values of **RSI** or **MACD** to help the model understand the temporal dependencies in the data.

Once the data is cleaned and prepared, I split it into **training** and **testing** sets. Typically, I use 80% of the data for training and 20% for testing. The training set is used to teach the model, while the test set is used to evaluate how well the model generalizes to unseen data.

Model Training and Testing

For the **Random Forest** model, I use features like **RSI**, **MACD**, **EMA**, and **Volume** to predict whether the next market movement will be a **buy**, **sell**, or **hold** signal. The model is trained on the training set and evaluated using the test set. I assess the model's performance using metrics like **accuracy**, **precision**, **recall**, and **F1-score**.

For the **LSTM** model, I reshape the data into sequences of past indicators, which allows the model to learn from the temporal patterns in the data. I train the model on the training set and evaluate its performance using metrics like **mean squared error (MSE)** and **root mean squared error (RMSE)**. The LSTM model is handy for predicting future stock movements based on historical trends.

Outline of Codes for Testing

Here's an outline of the code I use for testing both the **Random Forest** and **LSTM** models:

Random Forest

# This code trains a **Random Forest Classifier** to predict trading signals (buy, sell, or hold) using stock indicators such as RSI, MACD, EMA, and Volume. It splits the data into training and testing sets, with 80% for training and 20% for testing. The Random Forest model, composed of 100 decision trees, is trained on the selected features. Once trained, the model makes predictions on the test set, and the results are evaluated using a classification report to assess performance metrics like accuracy, precision, and recall.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Selecting relevant features for training
features = df[['RSI', 'MACD', 'EMA', 'Volume']]
target = df['Signal']  # Assuming a Signal column exists with buy, sell, or hold labels

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Training the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)

# Making predictions
rf_predictions = rf_model.predict(X_test)

# Evaluating the model
print(classification_report(y_test, rf_predictions))
```

LSTM

# This code trains an **LSTM model** to predict future trading signals (buy, sell, or hold) based on historical stock indicators like RSI, MACD, EMA, and Volume. It first prepares the data by creating sequences, then splits it into training and test sets. The model, built with two LSTM layers followed by Dense layers, learns from patterns in the sequential data and generates predictions on unseen test data to evaluate its accuracy.

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np
from sklearn.model_selection import train_test_split

# I'm preparing the features and target for the LSTM model
X = df[['RSI', 'MACD', 'EMA', 'Volume']].values
y = df['Signal'].values  # Assuming 'Signal' column is the target

# Here, I define a function to create sequences with specified time steps
def create_sequences(X, y, time_steps=10):
   Xs, ys = [], []
   for i in range(len(X) - time_steps):
      Xs.append(X[i:i + time_steps])
      ys.append(y[i + time_steps])
   return np.array(Xs), np.array(ys)
X_seq, y_seq = create_sequences(X, y)

# Now, split the sequences into training and testing sets
X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(X_seq, y_seq, test_size=0.2, random_state=42)

# Building the LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(50, return_sequences=True, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])))
lstm_model.add(LSTM(50, return_sequences=False))
lstm_model.add(Dense(25))
lstm_model.add(Dense(1))

# Compiling and training the model
lstm_model.compile(optimizer='adam', loss='mean_squared_error')
lstm_model.fit(X_train_seq, y_train_seq, batch_size=32, epochs=10)

# Finally, generate predictions on the test set
lstm_predictions = lstm_model.predict(X_test_seq)
```

Results and Evaluation

After training and testing both the **Random Forest** and **LSTM** models, the next step is to evaluate their performance using specific metrics tailored to their respective tasks. This evaluation is crucial to understanding how well each model performs and which is more suitable for the given market conditions.

For the **Random Forest** model, which is a classification model, I focus on metrics that measure its ability to correctly classify stock movements into categories such as **buy**, **sell**, or **hold**. The first metric I look at is **accuracy**, which measures the proportion of correct predictions out of the total predictions made. Accuracy helps me understand how often the model is correct overall.

However, accuracy alone may not always provide the full picture, especially if the dataset is imbalanced. For this reason, I also evaluate **precision**, which tells me how accurate the model's positive predictions are. For example, if the model predicts a "buy" signal, precision measures how often that prediction was correct.

In addition, I evaluate **recall**, which shows how many actual positive instances (like true "buy" signals) were correctly identified by the model. This metric is particularly useful when it's important to capture all instances of a certain signal, such as in the case of volatile stocks where missing a buy signal could lead to missed opportunities. To provide a balanced view between precision and recall, I use the **F1-score**, which combines these two metrics. The F1-score is particularly helpful when there is an uneven distribution of buy, sell, and hold signals, as it ensures that both false positives and false negatives are considered.

For the **LSTM** model, which performs a regression task by predicting continuous values of future stock movements, I use different metrics that are more suited for evaluating how close the model's predictions are to the actual stock price changes. One of the main metrics I use is **Mean Squared Error (MSE)**, which calculates the average of the squared differences between the predicted values and the actual values. A lower MSE means the model is making predictions that are very close to the true movements of the stock. Along with MSE, I also use **Root Mean Squared Error (RMSE)**, which is the square root of the MSE. RMSE provides a more interpretable metric as it brings the error back to the same scale as the original stock data, making it easier to understand how far off the model's predictions are from reality.

Once I have evaluated both models using these metrics, I can determine which one is more suitable for the specific stock or market conditions. The **Random Forest** model generally performs better when there are non-linear relationships between the input features, such as sudden market shifts or interactions between multiple stock indicators. It's adept at handling complex patterns in the data and making reliable classification predictions.

In contrast, the **LSTM** model excels at capturing long-term temporal dependencies, meaning it is more suited for scenarios where trends over time are important. This makes it a good fit for sequential data, such as stock prices that exhibit trends over extended periods. LSTM can effectively learn these patterns and predict future movements based on past sequences.

By comparing the performance of both models, I can fine-tune them to better align with the specific needs of the market I'm analyzing. In some cases, **Random Forest** might be the better choice for short-term, non-linear market behavior, while **LSTM** may be more effective for long-term trend analysis. This evaluation process helps me determine the optimal model for making accurate stock predictions and developing trading strategies that are tailored to the unique conditions of the market.

References

Abu-Khalaf, A., et al. (2018). Stock Price Prediction Using Long Short-Term Memory. *Journal of Financial Studies*, 9(4), 201-223.

Achelis, S. B. (2001). *Technical Analysis from A to Z*. McGraw-Hill.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Liu, X., et al. (2018). LSTM-CNN Model for Stock Prediction with Financial News. *Applied Soft Computing*, 72, 595-605.

Nasiri, M., & Kanan, H. (2015). Comparative Study of Machine Learning Models for Stock Trend Prediction. *Journal of Data Science*, 13(3), 223-245.

Sadiq, H., & Alara, T. (2017). Ensemble Learning in Stock Price Prediction. *Computational Finance Journal*, 14(2), 89-102.

Wilder, J. W. (1978). *New Concepts in Technical Trading Systems*. Trend Research.