KSCHOOL
Master in Data Science
edition 19

# MACHINE LEARNING FOR ALGORITHMIC TRADING
## An example with APPLE equity

Authored by
**Isabel Puche Marín**

Academic Year 2019-2020

# Table of Contents

**List of Abbreviations....4**

**1. Introduction....1**

**2. Selection of trading platform....2**

**3. Methodological Approach....3**

3.1. Trading Strategies....3

3.2. Process Flow....3

3.3. ML Model development....4

3.3.1. Time Series....4

3.3.2. Multi-target Regression....4

3.3.3. The choice of model....5

3.4. Model backtesting....6

3.4.1. Trading Algorithm....6

3.4.2. Backtesting....6

Backtest's report....6

Pyfolio's tear-sheets....7

3.5. Visualisation....7

**4. Technological issues....8**

4.1. Setting up the workspace....8

4.2. Pitfalls....8

4.2.1. Zipline's version updates....8

4.2.2. Package versions compatibility....8

4.2.3. Common errors....8

Pyfolio and Pandas for backtesting....8

Zipline and Empyrical/Numpy interactions....9

**5. Main Results and Conclusions....10**

5.1. Main Results....10

5.2. Main Conclusions....10

## List of Abbreviations

| | |
|---|---|
| TA | Technical Analysis |
| ML | Machine Learning |
| DL | Deep Learning |
| IDE | Integrated Development Environment |

# 1. Introduction

Despite newly available Machine/Deep Learning techniques, financial time series are considered hard to predict even the more so in the presence of efficient and transparent traded exchanges. However, there are opportunities to monetise forecast through rule-based trading and gain returns over the risk incurred. This is called **algorithmic trading,** **"**a process for executing orders utilizing automated and pre-programmed trading instructions based on complex formulas, combined with mathematical models and human oversight in order to make decisions to buy or sell financial securities on an exchange", according to investopedia.

In the last decades, with the availability of information machine learning (deep learning) for algorithmic trading has gained relevance for high frequency trading (minute frequency) and scalping strategies (profiting from short price movements). It is said that trading bots dominate exchanges nowadays in a race for improving forecasting techniques and scalping trading strategies. However, nowadays, more advanced techniques and strategies still coexist with more traditional ones, that have not been totally overridden, depending on the investor's profile and objectives.

In this context, **backtesting** plays an important role as it allows for a proper strategy's performance and viability evaluation thus conducting a simulation and analysis for the risk and profitability of trading strategy over a period of time. It is conducted with historical data, assuming that all things being equal, conditions will hold.

In a novel incursion to this topic, this project aims at assessing whether ML based *buy-and-hold* trading strategies have the power to outperform more traditional ones based on technical analysis.

## 2. Selection of trading platform

Both trading and backtesting can barely be programmed in Python with Numpy, Pandas and Scipy, but there exist already built-in libraries in Python, such as Zipline, with a somewhat large supporting community.

According to literature, **Zipline** is one of the most mature of all currently available choices. It has a rich functionality and scales well with large amounts of data. It has been developed and maintained by Quantopian, a Boston based investment firm, aiming at untapping talent among users through open competitions. The selected strategy is allocated a sum of money and the author paid based on performance. The Quantopian website uses the backtesting engine Zipline, but the library can be installed locally to reproduce a research environment. Trading in the website is in principle easier, but robust quantitative modelling and trading with non-US equities need a local environment.

Zipline comes in with the functionality of extracting free data source from Quandl, a financial provider. It provides US equities from 1990 until 2018, the year that the service was discontinued. It also integrates other useful libraries for financial statistics (**Empyrical)** and backtesting (**Pyfolio**), and is compatible with other traditional Python libraries, such as Pandas, Numpy, Scipy and Matplotlib. Zipline also offers a wide range of built-in factors for modelling.

However, local installation is non-trivial, as it requires a conda environment installation for 3.5 (or 2.7) version of Python and it has its downsides in its use, as we shall see.

# 3. Methodological Approach

## 3.1. Trading Strategies

The project is based on two momentum strategies, consisting of exploiting upward and downward trends in the belief they will continue its current direction: a **dual moving average crossover strategy** (TA Strategy) and a **machine learning based strategy** (ML Strategy).

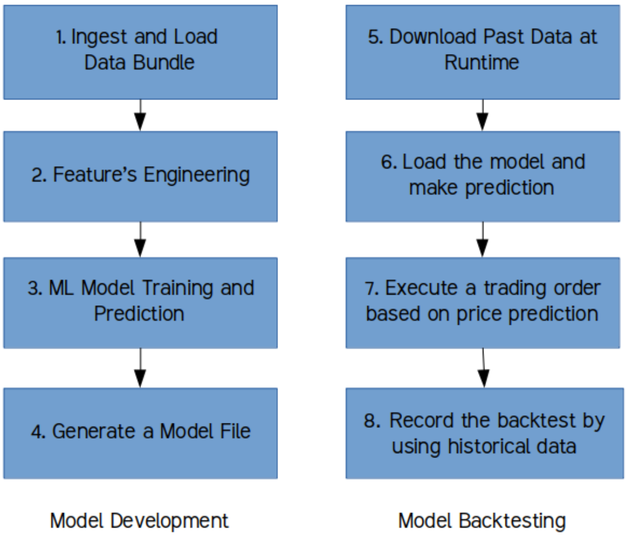
TA strategy's signals are triggered each time two rolling moving averages with different windows, slow (long) and fast (short), cross each other. When the fast moving average crosses up the slow moving average, a buying signal is recorded and vice-versa. ML strategy's signals are triggered at the end of each trading window (8 days) by comparing price prediction (with ML techniques) with (a subset of) historical data (32 days): if the prediction's maximum value is bigger than the mean of the historical sub-sample, a buying signal is activated and the other way round.

These are built on top of a *buy-and-hold strategy*, as defined by investopedia as "a passive investment strategy in which an investor buys stocks (or other types of securities such as ETFs) and holds them for a long period regardless of fluctuations in the market (…) and short-term price movements". Exit/selling conditions have not been specifically addressed.

TA Strategy will often be referred to as the **baseline strategy.**

## 3.2. Process Flow

In ML strategy, there are two major processes —one on **model development** and another on **model backtesting**. In TA strategy, there is only a model backtesting process.


1. Ingest and Load Data Bundle
2. Feature's Engineering
3. ML Model Training and Prediction
4. Generate a Model File
Model Development
5. Download Past Data at Runtime
6. Load the model and make prediction
7. Execute a trading order based on price prediction
8. Record the backtest by using historical data
Model Backtesting

## 3.3. ML Model development

### 3.3.1. Time Series

As aforementioned, US equities from 1990 until 2018 are made available by Quandl in a data bundle. A **bundle** is an interface to import and read data into Zipline and store it in its own preferred format (OHLCV plus dividends and splits). The library is able to read data incrementally, and only holds a part of the data in its memory at any given time. **Ingesting** is then the process of reading data with the help of a bundle, and storing it in Zipline's format. For data processing, the bundle needs to be **loaded** beforehand.

For this project, **APPLE** daily time series has been used, for no particular reason other than being the first U.S. company that recently hit $2 trillion. The original idea was to employ IBEX35 bluechips, more concretely Repsol's data, but despite creating a custom data bundle, for which a specific process is required in Zipline (see 00 notebook), an assertion error occurred at algorithm runs that could not be handled.

Despite the benefits of using (simple or log) returns instead of price in terms of stationnarity, a low correlation between features and targets and low explanatory power of regressors were reported, probably due to time series' loss of memory. That is the reason for using **close price**.

### 3.3.2. Multi-target Regression

For the sake of interpretability, a subset of historical data (32 lagged variables) was used to predict some values ahead (8 lead variables). Other financial factors, such as fundamental or alternative (e.g. sentiment) have been disregarded since model's specifications were deemed to be good enough and avoided to add more noise. There is another reason of trading strategy compatibility, since TA strategy is solely based on price.

Hence, this is a problem of **multi-target regression**, which involve predicting two or more numerical values given an input example. Some algorithms do support multiouput regression inherently, such as linear regression and decision trees. However there are also special workaround models that can be used to wrap and use those algorithms that do not natively support predicting multiple outputs. This is where Scikit's function multi-output regressor comes at hand for it fits one regressor per target.

As a drawback, it is not feasible to build a model's pipeline for data scaling and model fitting, which can ultimately help to select the best model when some of them need to be passed a wrapper.

### 3.3.3. The choice of model

Several models have been fitted in order to allow for a robust comparison. Here is a list with each models specificities and results.

– **Linear regression** captures general (linear) trend but tends to miss details and to output biased models. However, this has not been the case, being one of the most performing models in terms of goodness of fit and error term (though the latter is not specially surprising)

– **Decision trees** can also capture non-linear relations but do not model well enough variance data (since they have limited depth). Contrary to our intuition, it works well with training data and does predict well enough test data.

– **Random forest** instead provide a good balance between bias-variance models. This model (i) is a collection or ensemble of decision trees, (ii) uses bootstrapping aggregating (bagging) - with replacement and, (iii) uses a sample of features at each split (thus helping reduce the model's variance).

– **Gradient Boosting** is similar to random forest models, but the difference is that trees are built successively. With each iteration, the next tree fits the residual errors from the previous tree in order to improve the fit.

Gradient Boosting and Random Forest have been the most performing models, however the error term in both are bigger than in linear regression. Despite that, they both have successively plugged into the trading strategy.

– **Adaboost** (subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers)**, Kneighbors** (measures euclidean distance to find similar points for predictions) and **Support Vector Machines** (maps data into separated categories) have also been fitted with poorer results than those achieved by Gradient Boosting and Random Forest.

In order to prevent big features outweigh small one, which is the case with increasing time series like AAPLE value stock, it is preferable to **scale data**. This has been done for KNN's model and SVM.

**Hyperparameters** have been tuned with ParameterGrid function and GridSearchCV.

**Cross-validation** has also been performed with TimeSeriesSplit thus preventing data to reshuffle and fitting the models on the oldest data and evaluating on the newest data.

Finally, the model was saved in a **Joblib** format.

## 3.4. Model backtesting

### 3.4.1. Trading Algorithm

Zipline Quickstart guide provides a thoroughful documentation on trading algorithm but the structure is briefly laid out herebelow (see DataCamp tutorial for more information):

– initialize(): the first function is called when the program is started and performs one-time startup logic. It takes an object context, which is used to store the state during a backtest or live trading and can be referenced in different parts of the algorithm

– handle_data(): it is a function is called once per minute during simulation or live-trading to decide what orders, if any, should be placed each minute. The function requires context and data as input, the latter being another object that stores several API functions, such as current() to retrieve the most recent value of a given field for a given asset or history() to get trailing windows of historical pricing or volume data. The data object allows you to retrieve the price, which is the forward-filled, returning last known price, if there is one. If there is none, an NaN value will be returned. Another object is the portfolio, accessible through the context object. It allows to enter the portfolio positions object, including number of shares and price paid. Other portfolio's properties are cash (current amount of cash in your portfolio) and amount (whole number of shares in a certain position). order_target() places an order to adjust a position to a target number of shares.

– _test_args() define start and end date and capital base.

– analyze() settles the backtesting report

### 3.4.2. Backtesting

- **Backtest's report**

After running the algorithm a backtest's report is issued. Possible columns to check can be envisioned in run_algorithm script under strategies' folder. Most importantly, these reports return profit and loss measures, capital used, portfolio value, turnover and leverage ratios, etc (more in reports folder).

- **Pyfolio’s tear-sheets**

Pyfolio provide backtest ratios and graphs most commonly used in financial analysis, such as returns (annual / monthly, cumulative and its distribution), measures of volatility including Sharpe and Sortino ratios, drawdowns, first to fourth moments of time series’distribution, among others.

## 3.5. Visualisation

Flask has been used to create a web page to display backtest results. User guide on Annex.

# 4. Technological issues

## 4.1. Setting up the workspace

Following documentation, a **conda environment** has been created in order to (i) run Zipline in a Python version 3.5 environment, (ii) isolate Zipline's dependencies and (iii) control for possible interactions with base environment. The following libraries have also been installed: more importantly, Pandas, Numpy, Matplotlib, Seaborn and Altair, Scikit Learn, Joblib, Pyfolio and Flask. For replication purposes, check environment.yml file.

The IDE used for this project (ML trading algorithm) is PyCharm as it enables to write high quality Python code. The Autoenv magic for automating the environment activation and deactivation has also been implemented.

## 4.2. Pitfalls

### 4.2.1. Zipline's version updates

Quantopian released in July 2020 a new Zipline **version 1.4**. Two issues should be then considered: (i) set benchmark to false (as the library was experiencing benchmark download problems during algorithm runs), (ii) set time to timestamp.

### 4.2.2. Package versions compatibility

For Python v3.5 compatibility purposes, older versions of libraries have been installed, thus limiting full potential. As an example, Scikit Learn v0.20 does not include stacking function for ensemble models. In other cases, libraries were directly not supported in py3.5 such as Keras and Tensorflow for deep learning techniques and Streamlit and Altair-saver for visualisation tasks.

### 4.2.3. Common errors

- **Pyfolio and Pandas for backtesting**

Pyfolio's backtesting tear sheets need a specific trading's report format for which context.has_position variable should be indicated in trading's algorithm handle_data function.

- **Zipline and Empyrical/Numpy interactions**

Errors during trading algorithm implementation occurred pointing to (i) returns calculation (empyrical), (ii) use of numpy's log1p and (iii) other deprecated numpy's functions. These errors were in some cases partially solved, in others, test and trial with algorithm examples was conducted.

# 5. Main Results and Conclusions

## 5.1. Main Results

- **Different strategy performance**

TA strategy records cumulative returns of 34,1%, consistent with small investors' conservative strategies, whereas ML strategy reports 6050%. Capital used at the end of the period amounts to 14.454 USD and 116.175 USD respectively.

- **Number of trading signals not comparable**

Trading signals vary in a 1:3 proportion according to trading strategy: (i) 26 signals stem from arithmetic moving averages crossover whereas (ii) 80 signals were issued from 8-day trading window (out 635 days) in ML stock's strategy. Considering that most of them are buying signals (see paragraph 3.1), more trading (buying) opportunities have arouse and more capital has fueled-in and has been capitalised in ML based strategy compared to TA strategy leading to extraordinary cumulative returns.

- **Increased volatility in ML strategy**

When the stock price trend becomes steeper from mid-2016 until mid-2017, volatility skyrockets at impossible levels, which is also reflected on the variability of the rolling sharpe (excess return over volatility) and risk exposure. The absence of shorts positions (paragraph 3.1), exit conditions, such as stop-losses, contribute to it. As a direct consequence, not only profits but also losses tend to be very large.

- **Huge drawdowns in ML strategy**

Drawdowns are huge and occur at the beginning of the trading period (until beginning 2017), when portfolio value does not compensate for losses. Whereas ML strategy's drawdowns are more dependent on volume, TA strategy's drawdowns are more constraint to stock's price variability.

## 5.2. Main Conclusions

Generally speaking, it can be stated that ML strategy is better than TA strategy in reaping positive momentum benefits as well as losses from negative momentum.

Nonetheless, in light of results, this seem a weak statement. In purity, both strategies are not comparable in terms of trading opportunities and capital used, all the more so, considering that basic trading features, such as the definition of short trades, stop losses, capital constraint and risk management have not been defined.

However, this project has been conceived as a *research in progress* and as such, it has the vocation to further delve into stock prediction and algorithmic trading. Possible further steps can be:

– explore different **trading strategies** other than *buy-and-hold* with aforementioned features

– contain **volatility** with VaR (Value-at-Risk) and CVaR (Conditional-Value-at-Risk) metrics, that can also be predicted with (i) parametric estimations, such as Monte Carlo estimations and/or (ii) non-parametric estimations with ML, such as SVR and KDE

– explore other machine learning and **deep learning** models for stock prediction

– expand features' engineering with readily available **built-in factors** at Quantopian's and,

– include more assets for portfolio **diversification** and optimisation.