Machine Learning Capstone Project

Klemen Čas

Date: 2016-11-03

Github: https://github.com/KlemenCas/ML_Final

Readme to the technical implementation:

https://github.com/KlemenCas/ML_Final/blob/master/README.md

Abstract

This is the final project to the Machine Learning Nanodegree. The area of focus is Investment and Trading, narrowed to trading with SP500 stocks. Data is mostly sourced from Quandl, the SP500 composition originates from Wikipedia. Following information is included; stock prices, index prices, index composition, fundamentals (earnings per share and price book ratio), short selling volumes and sentiment. With this input, 16 features and 8 labels have been calculated. For optimization, the processed data has been stored locally.

Scikit-learn Support Vector Machines, Decision Trees, Random Forrest, kNeighbors, Adaptive Boosting and Naïve Bayes have been trained, on individual stock level and one the whole data set. Where grid search was available, it has been used for parameter optimization. Equally, Principal Component Analysis was used to reduce the dimensionality of the input, and clustering for reduction of label's complexity, for forecasting of price changes. The accuracy has been logged locally, to be able to select the best performing model during the forecasting. Furthermore, configured models have been saved locally, so that they don't have to be trained on the fly.

The models forecast the next state of the stock – state being the labels. The models do not forecast the actual buy/sell action. Q-Learning approach helps deriving the later; by using the forecasted labels as state.

The results are being visualized in two ways; first, the simulation module calculates the development of a kind of index ETF; a portfolio that starts with the SP500 composition, sells according to the system recommendation, and when tries to realign to the index. WIKI only provides today's index composition, therefore the implementation does not account for changes. Due to this benchmark's composition is being calculated daily, based on the market capitalization of the industry's stocks.

Second, there is a small module for individual stock prediction, with a very simple Tkinter interface.

The last chapter shows the performance of the setup compared to the benchmark.

Machine Learning Capstone Project

The report is divided into 5 chapters; Chapter I, Data Retrieval, describes the data retrieval and processing. Its outcome is a separate pandas DataFrame for each stock, stored locally, containing features and labels. Chapter II, Training, explains how the data is being used to train the scikit models, and how the results of the training are being logged. This chapter also looks at the performance of the individual training approaches.

Simulation, chapter III, describes the simulation of the stock and the index values.

Chapter IV briefly describes the setup for individual recommendations, in which the user selects a stock symbol and the system provides a recommendation.

Finally, chapter V, briefly summarizes the results.

I. Data Retrieval

Figure 1 visualizes the data retrieval process and the subsequent calculation of features and labels. First step is to retrieve the SP500 index composition. This is done by querying the Wikipedia's page List of S&P 500 companies[1]. The composition is stored in 3 dictionaries:

1. sp500_ticker; with stock ticker as the key and the industry as the value
2. sp500_composition; with industry as the key and the list of included stocks as the value
3. sp500_index; with the industry as the key and the quandl access key as the value

The information is then being used to access the following through Quandl available databases: Google Finance (GOOG) for index prices, Wiki EOD Stock Prices (WIKI) for stock prices, splits and dividends, Free US Fundamentals Data (SF0) for Earnings Per Share and Price Book Value, Core US Fundamentals Data (SF1)

[1] https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

for market capitalization, Financial Industry Regulatory Authority (FINRA) for short selling volumes and American Association of Individual Investors (AAII) for the sentiment.

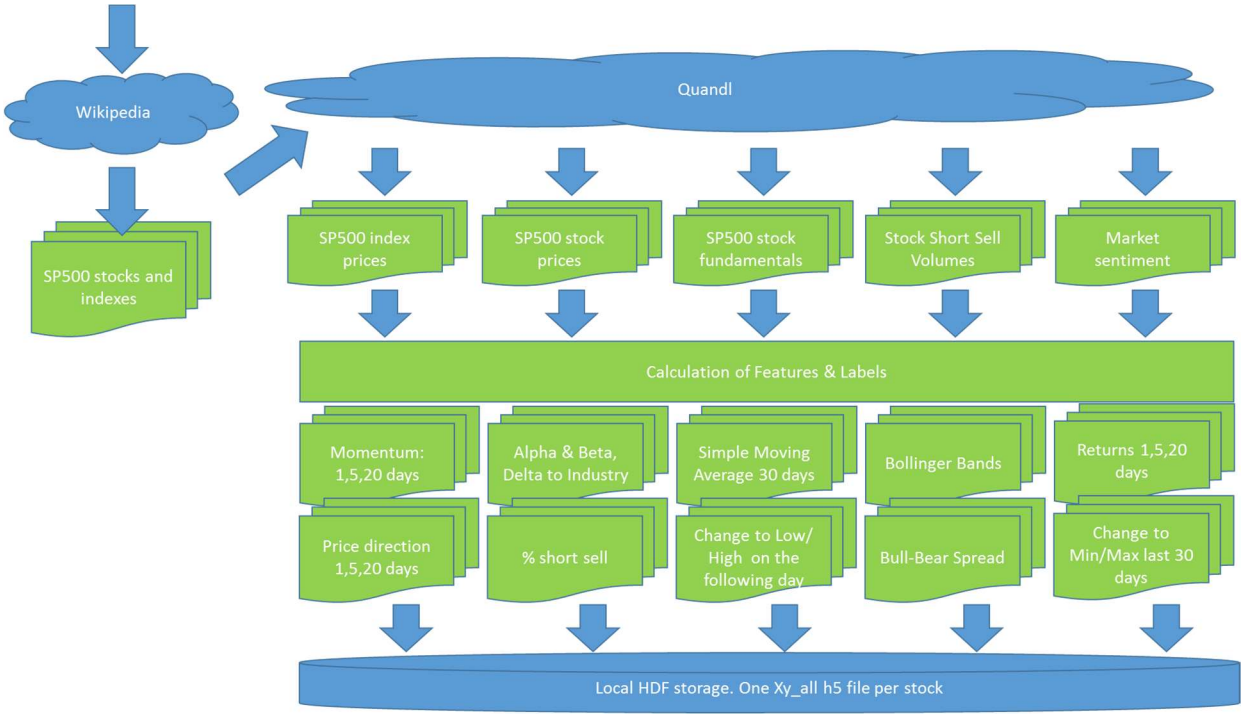

Figure 1:Data Retrieval Process[2]

The retrieved data is loaded into pandas DataFrames, which are stored locally on hard disc. Delta handling takes the local storage into account, so that at each future update it only queries the missing delta from Quandl. If there was a dividend payment or a stock split then the complete stock history is being reloaded.

After the retrieval, there are many NaN values in the data sets. For example, the fundamentals data source delivers the value on the date when the indicator has been reported, but it does not back or forward propagate any values. Thus, all data sets are being first back propagated, and then the remaining NaNs are

[2] The process is being controlled by the python module *mydata_get_delta.py*. To reduce the data volume and the processing times, a demo mode has been implemented. It's controlled by the flag *demo_scenario* is the module *commons.py*.

overwritten through forward propagation. This is in line with the thinking that reported figures apply to the past, and only in cases when we don't know a certain value today, we look for the last known indication.

In next phase the system calculates features and labels. All calculations are done on adjusted prices, so that dividend payments and splits are considered. Features:

1. *Momentum* for 1,5 and 20 days; the % change in the Close price over the last 1,5 and 20 days,
2. *Alpha and Beta* of the stock, based on which further 3 indicators are being calculated, *Delta to Industry Development* for the past 1,2 and 5 days,
3. *Delta to Simple Moving Average* of the last 30 days,
4. *Bollinger Bands,*
5. *Volatility* last 30 days,
6. *Delta to Min and Max* of the last 30 days,
7. *Percentage of Short Sell,* NASDAQ and New York Stock Exchange,
8. *Bull-Bear Spread*

and labels:

1. *Price Change* last 1,5 and 20 days
2. *Direction of Price Change* last 1,5 and 20 days
3. *Price Change from Close to Low* last day
4. *Price Change from Close to High* last day

In the next step scikit's MinMaxScaler transforms Momentum, SMA, Min, Max, Volatility, Bollinger Bands, Bull-Bear-Spread, Price Book Value, Earnings Per Share and Delta to Industry Returns to the range 0..1. All these transformations are done on by stock base, so that there is no cross dependency between stocks.

The output of the process is one hdf file by stock symbol, named Xa_all_<symbol> and stored locally.

II. Training

The input for the training are the Xy_all hdf files from the Data Retrieval phase. Six machine learning algorithms are being trained, all from the scikit learn: Support Vector Machines, Random Forrest, Decision Trees, AdaBoost, kNeighbors and Naïve Bayes. Figure 2 displays the flow.

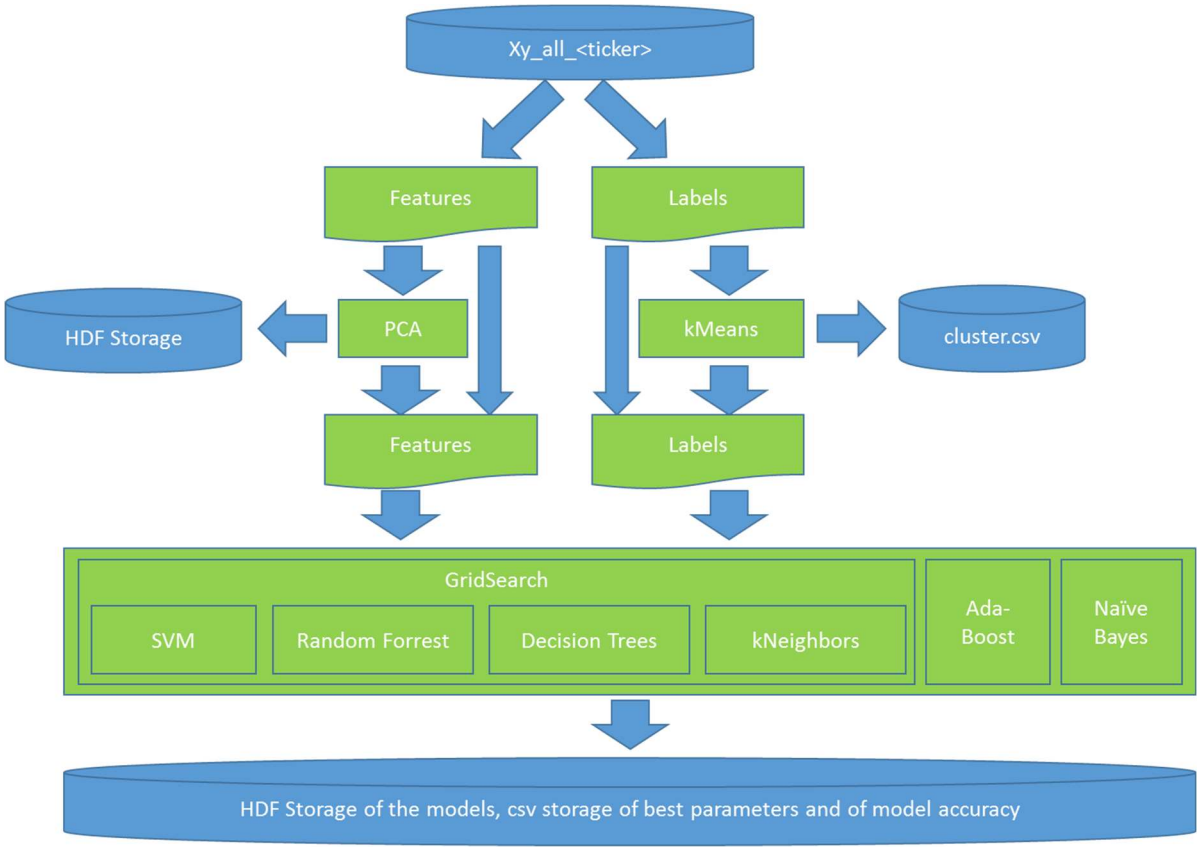

Figure 2:Process Flow for Training[3]

For SVC, Random Forrest, Decision Trees and kNeighbors optimal parameters have been determined with the help of Grid Search. For some settings trainings with PCA transformed features – from 16 to 9 dimensions delivered a higher accuracy, presumably due to less overfitting of the models.

[3] The training on stock symbol level is performed by the module *train.py*. The generic training, for models that train with data to all stocks of an industry takes place in *train_generic.py*.

Similarly, the number of values for the Close to Low and Close to High price change has been reduced by kMeans clustering. The training and the forecasting is hence not predicting the exact percentage change anymore, but only the affiliation of the percentage change with one of the 5 clusters.

The accuracy with a certain model/parameter setting, the a specific trained model and the PCA object have been logged and stored as a pickle object (the models) or in a csv file. This way the best performing setup can be loaded during forecasting and reused.

The training started with trying out different numbers of PCA dimensions for 20 stocks. This indicated that in case of a dimension reduction 9 dimensions delivered the best results. Therefore, subsequently, the whole dataset has been trained twice; once without a dimension reduction and once with 9 input dimensions. In total, more than 75,000 trainings have been performed.

On the average dimension reduction delivered slightly higher accuracy, 73.5% versus 72.5% (Figure 3). The labels are:

1. *x dd_Close:* Direction of price movement, x day(s) into the future
2. *chr_cluster_x:* The percentage change between the Closing price and the High price on the follow-up day in the cluster x
3. *clr_cluster_x:* The percentage change between the Closing price and the Low price on the follow-up day in the cluster x

Average of value	Column Labels	
Row Labels	0	9
1dd_Close	51.0%	51.6%
20dd_Close	63.6%	64.1%
5dd_Close	56.1%	56.9%
chr_cluster_0	81.9%	83.8%
chr_cluster_1	60.9%	62.5%
chr_cluster_2	65.1%	65.7%
chr_cluster_3	85.2%	85.9%
chr_cluster_4	96.2%	97.1%
clr_cluster_0	96.6%	97.7%
clr_cluster_1	86.7%	87.5%
clr_cluster_2	66.7%	67.5%
clr_cluster_3	60.1%	61.4%
clr_cluster_4	79.6%	81.0%
Grand Total	**72.5%**	**73.5%**

Figure 3: Accuracy with no dimension reduction (column 0) and with reduction from 16 to 9 dimensions

As for the models, on average SVC delivers the highest accuracy (Figure 4).

	AB		DT		GNB		kN		RF		SVC	
	0	9	0	9	0	9	0	9	0	9	0	9
Average of value	**72.2%**	**72.3%**	**72.0%**	**72.1%**	**68.7%**	**73.0%**	**72.6%**	**72.4%**	**73.7%**	**73.8%**	**75.9%**	**77.2%**

Figure 4: Accuracy, broken down by model

These though are average values, from which the individual cases deviate. As one can see in the *stats for report.xlsx* file in the project directory, in more than 40% of the setups a model different to SVC delivered the highest accuracy (Figure 5).

	SVC_9	SVC_0	kN_0	RF_0	RF_9	AB_0	kN_9	AB_9	GNB_9	DT_9	DT_0	GNB_0
Count	3251	1453	809	684	595	363	334	281	193	108	107	49
Percentage	40%	18%	10%	8%	7%	4%	4%	3%	2%	1%	1%	1%

Figure 5:Number of highest accuracies by model and the number of input dimensions

III. Trading Simulation

The simulation module picks the last 6 months[4], assembles an index portfolio based on the index composition as described in the data retrieval chapter, and an identical simulation portfolio. It does this 10 times. For each day in a simulation run there is an update calculation:

1. The Index Portfolio is being reassembled based on the market capitalization of the stocks on the day before.
2. The Simulation Portfolio is being changed according to the labels forecasted by the models and a Q-learning approach in the mapping from the labels to the actual buy/sell actions.

Figure 6 shows the flow in detail; The simulation runs by the index, and by an individual stock within the index. The overall performance is the comparison between the value of the index portfolio and the simulation portfolio. After the index and the symbol have been selected, the accuracy statistics from the training phase is being used to select, load and execute the best model. The labels part of the Xy_all HDF storage is being ignored.

The model predicts the state of the stock symbol; the expected direction of the price movement for the next 1,5 and 20 days and the affiliation to Close-Low and Close-High clusters.

[4] The time span is controlled by the start date in *market.py*.

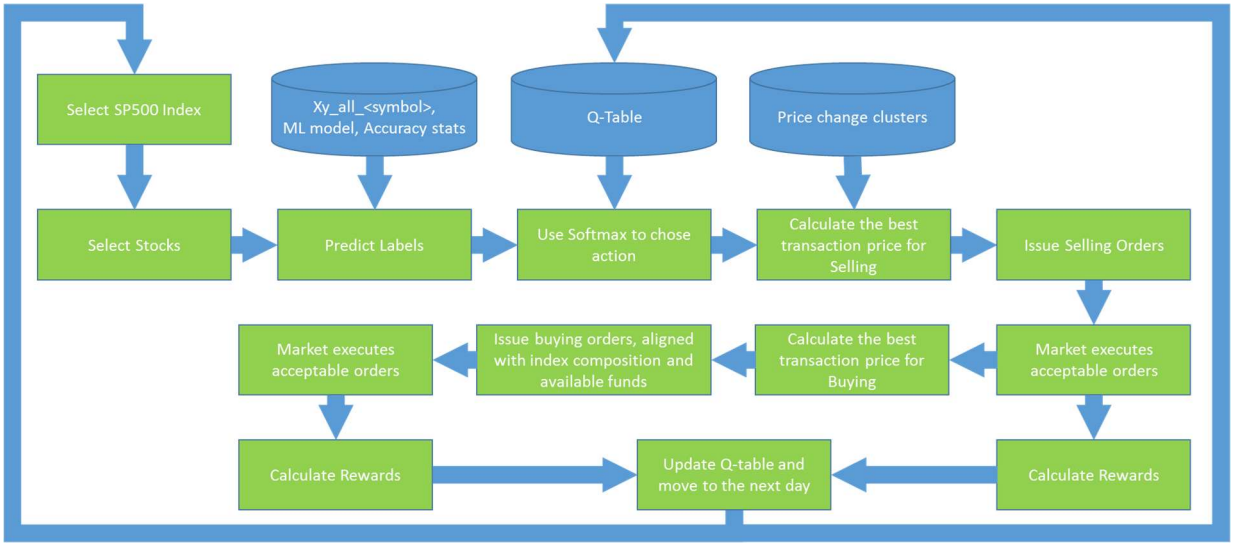

Figure 6: Simulation Flow

Softmax determines the recommended action, with which then the order price is being calculated, based on the price change clusters. This is the price with which the transaction will be issued. The market will reject it, if it's not above daily Low (for buying orders) or if it is above the daily High for selling orders. To avoid rejections and to optimize the price, two more checks are in place;

1. if the suggested price for a buying order is above the opening price of the follow-up day then the order price is reduced to the opening price. This assumes that the orders are only being released after the markets open. Similarly, for selling orders, if the selling price is below the opening price, then the order price will be lifted to the opening price.
2. If the order has not been executed by x minutes before the markets close, then the order price will be changed to the anticipated close price. The algorithm assumes that this is, on average, not more than 10 basis points from the actual closing price (= .1%).[5]

[5] The 10 basis points are picked arbitrarily, and would need to be verified with actual intraday stock movement statistics.

Selling order always contains the full volume of the stock in the portfolio. The buying orders are only being released, after the selling orders have been executed and the system knows how much cash it has available for the purchases. Once released, the purchase volume by stock is being calculated. The aim is to have a portfolio that is close to the index. Hence, the system first calculates how much of each recommended stock – given the overall value of the portfolio – it should purchase, to realign with the index portfolio. Once that is executed, it purchases all recommended stocks for the remaining amount in the same distribution as what the index portfolio has.

At the end of a trading day there can be cash left, or the cash position can be non-zero. This happens because the alignment of the buying list is performed at the beginning of the day, based on the planned purchase order price. The actual price can deviate though, either due to lowering to the opening price, or due to lifting to the close price towards the end of the day.

The rewards for Q-Learning are calculated at each requested transaction:

1. Close price has risen and the recommendation was Buy; Reward= +200
2. Close price has risen and the recommendation was Sell; Reward= -200
3. Close price has fallen and the recommendation was Buy; Reward= -200
4. Close price has fallen and the recommendation was Sell; Reward= +200
5. Close price has not changed: Reward = 100

With the reward, alpha=.2 and gamma=.4 the q-table is then updated.[6]

[6] The gamma and alpha values are arbitrary; it's simply the two values that lead to good performance in the Smartcab project. To be more accurate, different values would have to be simulated.

IV. Individual Stock Recommendation

The individual stock recommendation is a two-step process. First the missing data must be queried from Quandl, to update the feature tables (Xy_all<symbol>, see chapter I to data retrieval). This is a time-consuming step that is being executed before the actual user forecasts are being ran. Once done, the user interface can provide stock price recommendations for the day[7].

Another difficulty with the daily update is that two of the data sources are not free, and can only be queried by having the Quandl subscription in place. Hence, for demonstration purposes, the user forecast date has been set to Oct 31st 2016, which is after the training for all models.[8] These have been trained with data up to Oct 7th.

As for the process, it's a special case of what has already been described in the chapter about simulation:

1. The user selects the stock symbol that she would like to have forecast to. (Figure 7)
2. The system loads the feature data and the per training statistics best forecast model
3. Model is being ran and the user receives the forecasted direction of the price change, cluster centers to the Close-to-Low and Close-to-High forecasted percentage change and the expectation in which cluster the change will fall in.

Figure 7: Stock selection

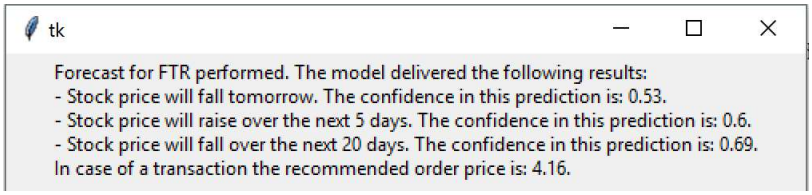

Figure 8: Forecast

[7] The recommendations are called by executing the module *UI_forecast.py*.

[8] See *self.demo_date* in *UI_forecast.py*.

V. Model Performance

Simulation results indicate performance above the index, which was out benchmark; for the SP500-50 the model delivers roughly 10% better results than the index (see Figure 9).[9] The reliability of this number depends though heavily on the assumption that the trading strategy is feasible in real markets. Four hypotheses need to be verified:

Figure 9:Examplary performance. (green=ML portfolio)

1. Stocks that are to be sold can be sold at the market opening price. This is for instances where the forecasted High price is below the opening price.
2. Stocks that are to be bought can be bought at the opening price, for instances where the forecasted Low price is above the opening price.
3. A transaction envisioned shortly before the market close can, on average, be executed within 10 basis points of the actual Close price.
4. The model is not overfitting on the historical data.

VI. Final Thoughts

The implementation delivers a much better performance than the index, but a lot of further verification and kind of quality assurance would be needed to trust the results. A challenge is computing power. It took several days to train the models for all 500 stocks. Equally, changes in the calculation of features could take days. Hence, optimizations and verification of the outcome is a tedious process.

---

[9] See csv files *1performance_log ..10performance_log* in */stats/* directory.

Furthermore, there are some potentially problematic assumptions in implementation. The highest risk is the assumption that the purchasing/selling price can be executed on average very close to the actual opening or closing price. The author did not have any intraday trading data at his disposal, hence does not know.

In general, data is an issue. We know that there is much more to the stock development than the stock price development, two fundamentals, short selling and sentiment. For the approach to improve, additional data sources would need to be implemented.

Finally, this document should stay in the range of 8-12 pages. This is too short to address all the details of the implementation, the findings and the challenges along the way. The author has therefore focused on the high-level description.