Machine Learning Engineer Nanodegree

Capstone Project

Sverre Lillelien
December 3rd, 2018

I. Definition

Project Overview

Automation of stock trading has been among the most popular uses of advances in hardware and machine learning, and has been a topic of interest to academics and practitioners alike. Using algorithms to attempt to identify over- or undervalued companies traded on the stock market is a strategy employed by many investment funds and banks. The stock market is a natural candidate for such analysis because of the vast amount of data, even real-time data, generated by stock markets and companies. For a human it is impossible to filter through all of this information in a meaningful way, so perhaps machines can help us do this dirty work for us, and provide some new insight.

As there are countless variables impacting stock prices, and a high degree of randomness involved, predicting stock prices with any degree of certainty and consistency seems impossible. However, this approach is employed by such a vast number of different funds and analysts that it would be a useful exercise in its own right to attempt to apply some machine learning techniques to this field in order to better understand its strengths and limitations. With luck, even finding some small insight which could help us filter through some of the noise in the stock market which might give us an edge and narrow our focus when selecting stocks, would be valuable.

The data set in this project consists of five years of stock data for S&P 500 companies. The data includes opening prices, as well as high, low and closing prices. The data set can be found here: https://www.kaggle.com/camnugent/sandp500/home

Problem Statement

The problem I wish to analyze in this project is whether we can predict the future price movements of a stock. Specifically, I will attempt to label stocks as either buy or hold based on changes in its price and the prices of other stocks in the S&P 500 within a period of time, and treat this as a classification problem.

The tasks I will go through are the following:

1. Load the data from a csv-file
2. Explore and visualize the data
3. Preprocess data for machine learning
4. Train a classifier to label stocks as buy or hold
5. Refine to model improve it over the benchmark model

"Solving" the problem is measured by classifying some stock as either a buy, or to keep holding the stock. As such it is quantifiable and easy to understand. To help me solve this problem I will utilize supervised learning, more specifically SVM and random forest classifiers.

Metrics

Accuracy, precision and AUC could all be relevant metrics to measure performance of a model in this project. For random forest, AUC will be our primary metric to evaluate model performance.

Accuracy requires the data set classes to be balanced, and this is only true to a certain extent in this case. We will still consider the accuracy of the model, but I believe *precision* could be a better metric to judge the model on. One way this project might influence our trading strategy would be to use the model to identify candidate stocks to buy. In this case, we want to be fairly certain that if the model classifies a stock as buy, this can be trusted. We want to limit the amount of false positives, as this could cause us to lose a lot of money. This favours precision as a metric. Conversely, we can accept some type-II errors from the model. That is, failing to identify possible buy-candidates. This would be a lost opportunity, but not something that would cause us to lose money. All of this points towards precision as a valid metric.

II. Analysis

Data Exploration

The data set contains price data and trade volume for all S&P500 stocks over a five year period. The data is separated into seven columns:

- date - In format year-month-day, starting from earliest date
- open - stock price in USD at market opening (float)
- high - daily high of the stock price (float)
- low - daily low of the stock price (float)
- close - stock price at market close (float)
- volume - number of stocks traded during the day (integer)
- name - the stock ticker (string)

The data set contains 619 040 rows of data. Each row contains a set of prices for a certain stock ticker, for a certain date. Data has been captured from February 2003 and going till February 2018. The data is fairly clean without abnormalities, we will simply need to normalize it before running any classification, as well as deal with some missing values since not all stocks have prices for the full period.

```
In [5]: df.head()
```

Out[5]:

	date	open	high	low	close	volume	Name
0	2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
1	2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2	2013-02-12	14.45	14.51	14.10	14.27	8126000	AAL
3	2013-02-13	14.30	14.94	14.25	14.66	10259500	AAL
4	2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL

Exploratory Visualization

The focus of this study will be on closing prices for each day, ignoring intra-day fluctuations. After pivoting the data frame using date as the index, each row is now a date containing price data for all stocks in the index, reducing the total number of rows in the data set from 619 040 to 1 259.

```
In [12]: df.head()
```

Out[12]:

Name	A	AAL	AAP	AAPL	ABBV	ABC	ABT	ACN	ADBE	ADI	...	XL	XLNX
date													
2013-02-08	45.08	14.75	78.90	67.8542	36.25	46.89	34.41	73.31	39.12	45.70	...	28.24	37.51
2013-02-11	44.60	14.46	78.39	68.5614	35.85	46.76	34.26	73.07	38.64	46.08	...	28.31	37.46
2013-02-12	44.62	14.27	78.60	66.8428	35.42	46.96	34.30	73.37	38.89	46.27	...	28.41	37.58
2013-02-13	44.75	14.66	78.97	66.7156	35.27	46.64	34.46	73.56	38.81	46.26	...	28.42	37.80
2013-02-14	44.58	13.99	78.84	66.6556	36.57	46.77	34.70	73.13	38.61	46.54	...	28.22	38.44

5 rows × 505 columns

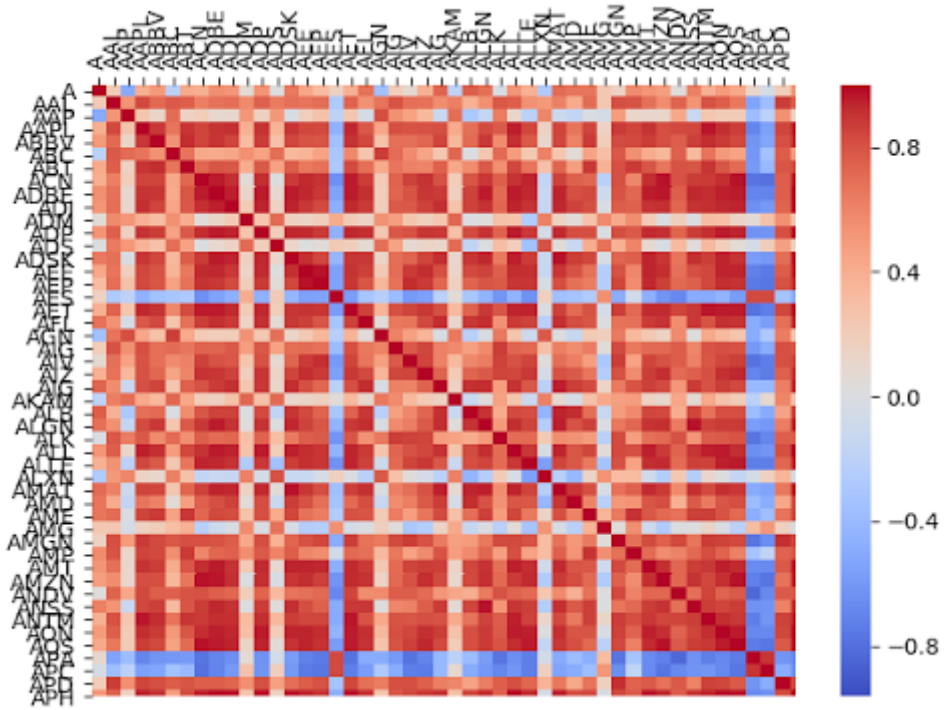
So what can we use all of this pricing data for? Plotting a stock (column) over time gives us a nice visualization of the stocks performance over time. This is a useful starting point for any analysis, and gives us hints of general performance, volatility, seasonality and other factors.

```
In [19]: # Google
stock_visualization('GOOGL')
```

As we can see from looking at the share prices of Google (above) and Apple (below), their stocks have performed well over the last five years. This is not surprising, as we are looking at two of the most successful tech companies operating today. However, it is interesting to note both stocks seem to experience a marked decline at the end of our time period. In fact, this probably did not have anything to do with the companies themselves. A lot of stocks fell in February 2018, which was the worth month for the S&P 500 in two years. Some stocks fell more than others, while some didn't fall at all.

```
In [18]: # Apple
stock_visualization('AAPL')
```

We can also look at the correlation between stocks, and visualize this as a heat map. Plotting all companies together produces the following graph, here zoomed in to look at only the first companies in the index.



The heat map reveals correlation between groups of companies. We know that stocks in general have tended to increase in value over time, but the heat map reveals both positive and negative correlation, of different strength. This is useful for at least two reasons: To hold a diversified portfolio, we want to avoid having all of our assets being positively correlated. Secondly, if we infer from this that the price of groups of companies tend to move together, can we gain an edge by studying this relationship using machine learning? The assumption that certain companies

move together makes a lot of sense. For instance, we expect the profits (and thus share price) of companies in the oil industry to be linked to the oil price, and we expect retail companies to linked to e.g. the disposable income of consumers or the unemployment rate. There might also be relationships that are less obvious to a human, but might be revealed using machine learning. In most cases, we would not expect the stock prices to move simultaneously for all companies. Some companies will lead, some will lag. It is this relationship I want to study more closely, which might give us an edge in deciding whether to buy, sell or hold a certain stock.

Algorithms and Techniques

Our primary classifier will be an ensemble method: a random forest, training each decision tree on a different subset of training data. By using a random forest we hope to reduce some of the overfitting issues with decision trees. Random forests have quite fast runtimes and works well on unbalanced data. The potential weakness that they can't predict beyond the range of the training data shouldn't be an issue for our purpose.

The other classifier will be a Support Vector Machine, one of the most popular algorithms in modern machine learning. It was introduced by Vapnik in 1992, and owes a lot of its popularity to providing impressive classification performance on reasonably sized data sets. Due to poor scaling with the number of training examples, SVMs do not work very well on large data sets, but it should be suitable for our data set of stock prices.

The following parameters can be used to optimize the random forest classifier:

- bootstrap
- max_depth
- max_features
- min_samples_leaf / min_samples_split
- n_estimators

For SVM we can look at C, dual and loss.

Benchmark

As a benchmark an out-of-the-box Random Forest Classifier will be used, which we can hopefully outperform. Specifically, we want to primarily improve upon a AUC of 0.48, or very close to 0.5,

which, perhaps not so surprisingly, implies a random split. Secondly, we want to achieve a higher precision than 43.6 % which we get from using default values.

III. Methodology

Data Preprocessing

The data preprocessing is done in the capstone project notebook, and contains the following steps to prepare the data for analysis:

1. Dropping irrelevant columns and pivoting the data frame using date as index, with one column for each stock ticker.
2. Filling missing values with 0.
3. Normalizing the data by converting stock prices to percentage change. These percentage changes will be the features determining our labels (buy or hold).
4. Labelling the stocks as buy or hold based on a required percentage change in price.
5. Dividing the data into training and testing sets.

Implementation

The implementation process took place in Jupyter notebook, and all implementation steps have been made into python functions. The classifiers were trained on preprocessed data, using stock ticker as input to predict whether or not to buy the stock. Implementation follows the following steps:

1. Getting features for the ticker via the helper function get_features, returning X - our feature set, y - buy or hold labels, df - updated data frame
2. Feature and label data is split 80/20 into training and testing sets with train_test_split from sklearn's CrossValidation.
3. Classifier is instantiated with default hyperparameters, and trained on the data
4. Predictions are made on the test data, and results printed out
5. The model is saved using pickle

Different functions were made for SVM and Random Forest and both were tested to compare results.

```
def ml_rf(ticker):
    '''
    Baseline out-of-the-box Random Forest as benchmark.

    INPUT:
    ticker - stock price ticker

    OUTPUT:
    Data spread, accuracy, prediction spread
    '''
    # Getting the relevant features and df for the ticker
    X, y, df = get_features(ticker)

    # Splitting data for crossvalidation
    X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        test_size = 0.2)
    # Instantiating model
    clf = RandomForestClassifier()

    # Training model on the training data
    clf.fit(X_train, y_train)

    confidence = clf.score(X_test, y_test)
    print('Accuracy:', confidence)

    # Predicting on test data
    predictions = clf.predict(X_test)
    print('Predicted spread:', Counter(predictions))

    return clf, predictions, X_train, X_test, y_train, y_test
```

```
In [29]: # Test random forest
         model, predictions, X_train, X_test, y_train, y_test = ml_rf('GOOGL')

         Data spread: Counter({'0': 821, '1': 438})
         Accuracy: 0.638888888889
         Predicted spread: Counter({0: 215, 1: 37})
```

We print the spread in the data prior to training the model, indicating 821 instances of “hold” and 438 instances of “buy” for Google. Here it should be noted that only instances where the stock price actually rises 2 percentage points or more are counted as a correct “buy” classification. In reality, we would still be making a profit if the stock only rose 1 percentage points, or anywhere between 0 and 2 percentage points.

Some complications arose as a default SVM simply predicted “hold” for all cases, even when using a linear kernel. Using LinearSVC yielded more useful results, but here I ran into a problem when trying to plot the AUC, as LinearSVC does not have the method predict_proba needed to draw the curve. Furthermore, in theory using SVC with a linear kernel should be equivalent to using LinearSVC. However, it turns out these are two different implementations even though the documentation states these should be similar. Investigating this further it seems some users advise against using LinearSVC.

Due to the limited time scope of this project I focused my efforts on Random Forest from this point, to avoid these complications.

Refinement

The final model is a result of an iterative process, where various algorithms have been tested and hyperparameters tuned. Focusing on Random Forest, a parameter list containing bootstrap, max_depth, max_features, min_samples_leaf, min_samples_split and n_estimators were all tested and tuned. GridSearchCV was used to find better parameters for the model, to try and improve the AUC. A higher AUC was achieved using precision score rather than roc_auc as scorer in Grid Search. Finally, using GridSearch can be very time consuming, and on the hardware used for this project it took a long time to run each iteration. This limited the range of values that could be used for experimentation.

```
# Final dictionary of test parameters
parameters = {'max_depth': [6, 10, None],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10],
              'n_estimators': [800, 1000]}
```

Using the parameters above as input in the final iteration of GridSearchCV, we ended up with the following model parameters for the final model, compared to the benchmark:

Hyperparameters	Benchmark	Final model
max_depth	None	6
min_samples_leaf	1	2
min_samples_split	2	5

Hyperparameters	Benchmark	Final model
n_estimators	10	800

Interestingly, increasing n_estimators only improved the model up to a certain level. Including values above 800 did not improve the model.

IV. Results

Model Evaluation and Validation

The following hyperparameters were chosen because they performed the best among the different combinations tried.

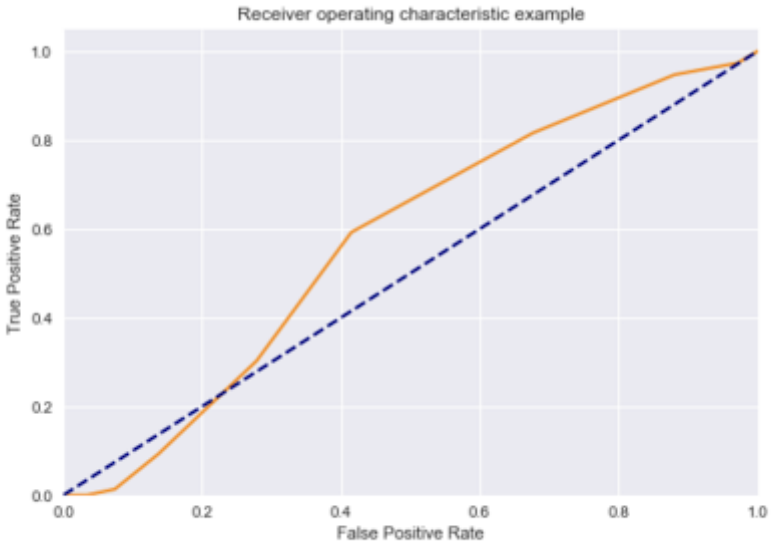
Hyperparameters	Final model
Bootstrap	True
max_depth	6
max_features	auto
min_samples_leaf	2
min_samples_split	5
n_estimators	800

max_depth of 6 seems like a reasonable value, and should help keep the complexity of the learned models low, thus reducing overfitting. n_estimators of 800 is a large number compared to the default value of 10, and we know that a higher number of estimators decreases the chance of overfitting the model. Given what knowledge I have of these hyperparameters, the values seem to make sense, and would be expected to be an improvement upon the default values.

To test the robustness of the model I have trained, tested and refined it on multiple stocks. The results are fairly consistent. Again, this is a very time consuming process on the hardware used for this project, and it is likely further testing would improve the model and its robustness.

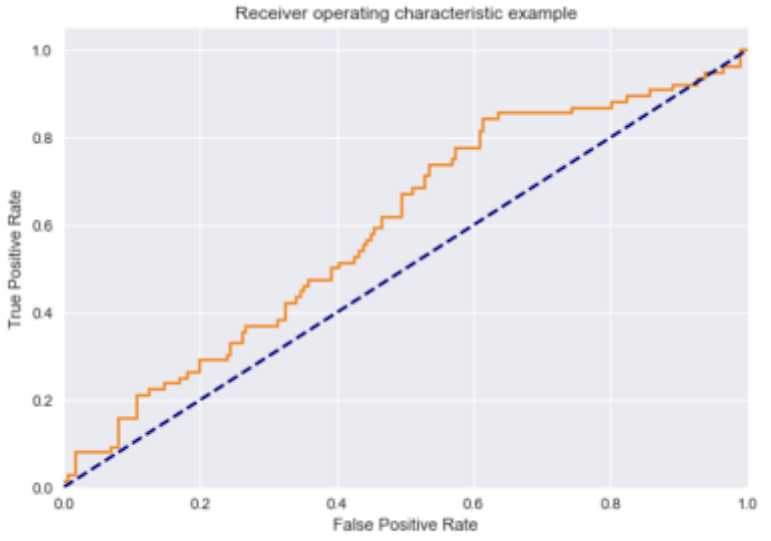
Justification

The benchmark model gave an AUC of 0.48. This indicates a random split, and that there's no clear pattern in the data.




```
Out[67]: 0.47787081339712917
```

By refining the model we managed to increase the AUC to 0.53. This is an improvement on the benchmark, but still a fairly linear curve.




```
Out[77]: 0.53095095693779903
```

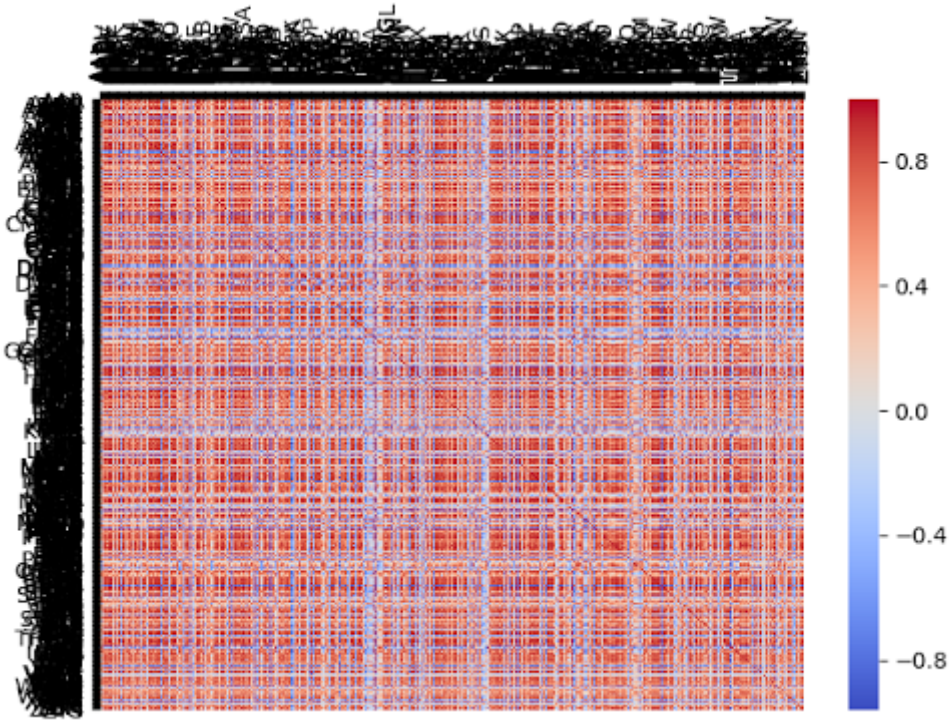
We also want to look at precision as a metric to evaluate the model. The benchmark model had a precision of 43.6 % for "buy" classifications. The final model has a precision of 100 % for "buy" classifications. That is, all stocks classified as a "buy" were indeed true positives. This indicates that the requirement inputs and model parameters have yielded a conservative model, which is

exactly what we want. Our goal is to develop a model which can be used to identify potential stocks to invest in, and while we might have failed to identify some good investment opportunities, all of the ones we have identified have been classified correctly. As our purpose isn't to fully automate stock trading, but rather use the model as a tool to give us an edge, this final model yields results that shows some promise.

V. Conclusion

Free-Form Visualization

Visualizing results from the project does not make a lot of sense, but the premise of the entire project is the fact that stocks are correlated to each other. Some more than others, and some even negatively correlated. These are the relationships we hope can give us an edge in the stock market. So let's look at the heatmap of the entire S&P 500 index.



There is enough red in this graph to give some support to the claim that certain stocks tend to move together. The labels on the axes aren't readable without zooming in due to the fact we're showing all 500 companies.

Reflection

The initial challenge for me was to narrow the scope of the project sufficiently, and gather data that would allow me to complete the project. I ended up looking strictly at stock prices from a classification perspective, and chose to leave fundamentals like account data to the side. It was not obvious to me that treating this as a regression problem wouldn't have been a good choice.

The next challenge was working with the data in order to prepare it for machine learning. I ran into lots of problems over the course of the project, but managed to solve the ones related to the data itself. However, I would have liked to try out even more machine learning algorithms than I have. Time was my main constraint here.

And perhaps the largest challenge with this entire project is the fact that we are aiming to get some marginal advantage over others when it comes to investing in stocks. This means that if we can increase our chances of picking the correct stock to buy from 50 % of the time to 51 % of the time, this would be a major advantage. However, judging whether or not the model is good enough, and interpreting all of the results, is something I find very challenging given the nature of this problem.

What I do find interesting is the seemingly high precision we consistently seem to get from refining the model. Not all of my iterations have given 100 % precision, but all of them have been a strong improvement upon the benchmark, ranging from 70 - 100 % as opposed to around 40 % for the benchmark model. It's hard to attribute this to randomness, and my suspicion going in to this project was that we were unlikely to produce any significant, consistent results due to the arbitrary nature of stock price fluctuations.

Improvement

In general I would expect increasing the number of estimators in the model would yield better results and reduce the risk of overfitting. I found that increasing n_estimators from 800 to 1000 did not improve the model, but it is plausible that even an even higher number would improve the model. This is constrained by computing power, and better hardware would be required to test this.

A potential source of improvement could also be to alter the data we use to train the model. We are training on five years of data, checking for correlation. Training over such a long period of time might be a problem due to companies' relationships changing over time. Perhaps using three years to train the model would give us better predictions. We could also tweak both the number of days we are measuring price changes for (from five), and the required price change (2 percentage points).

Like I discussed previously, augmenting the model and expanding it with e.g. financial accounts would be an interesting way to expand the project further.

Finally, there were other algorithms I would have liked to try and implement. XGBoost and LightGBM were suggestions I would want to try, given more time.

It seems clear that even though we have some promising results from our final model, it should be possible to improve upon this further, and that we could get a better solution by applying some of the suggestions above.
