Capstone Project

Machine Learning Engineer Nanodegree

Definition

Project Overview

The problem domain is financial markets time series. The purpose of this project is to find a model which is able to profit from the market moves of some stock. For the scope of this project, we chose Apple Inc. stock (ticker: AAPL), although any model we use can be applied to any other stock financial data. Our aim is to make higher profits than some chosen benchmark with lower risk than the given benchmark. The dataset comes from Quandl, a marketplace for financial and economic data offering open-source data. We enclosed a copy of it in our github repo.

Problem Statement

The problem we aim at solving is to find an optimal model to buy and sell some given stock in order to make profits out of difference between buying price and selling price. We will try to find a solution to this problem through a variety of machine learning approaches. More detail about this can be found in the Analysis section and in particular in the Algorithms and Techniques subsection. We will choose a set of relevant features, on the assumption that their combination might have some predictive power.

In a nutshell, we will be satisfied with the solution if the graph of cumulative profits over time has an upward trending slope and the ratio between the sum of all profits and the sum of all losses is at least greater than 2. This will mean that our model makes twice as much money than it loses, thus ensuring both profitability and low volatility.

Metrics

- Graph of cumulative profits over time: this shows if the amount cumulative profits made by our agent are a positive amount, how smoothly profits are made with respect to the losses and the downside risk of the periods of negative performance
- Profits minus losses: how much the agent has made minus how much the agent has lost
- Average profit or loss per trade
- Standard deviation of profits and losses
- Percentage of winning trades over all trades (n. of winning trades + n. of losing trades)
- Average profit of winning trades

- Average loss of losing trades
- Drawdowns: amount of money lost from any cumulative peak in the profit curve
    - Worst peak-to-valley drawdown
    - Average drawdown
    - Median drawdown
    - 3rd quartile drawdown
- Risk-adjusted profit or loss measures:
    - Average profit or loss per trade divided by standard deviation of profits and losses
    - Average profit or loss per trade divided by standard deviation of losses only
    - Average profit or loss per trade divided by worst peak-to-valley drawdown
    - Sum of money made of winning trades divided by sum of money lost on losing trades

Our ideal model:

- makes more money than it loses,
- shows a smooth upward trending profits curve,
- makes high average profits,
- makes low average losses,
- makes high percentage of winning trades,
- makes small drawdowns,

Analysis

Data Exploration

The dataset consists of financial time series of Apple Inc stock (ticker: AAPL) and comes from Quandl's open source wiki. (https://www.quandl.com/data/WIKI/AAPL-Apple-Inc-AAPL-Prices-Dividends-Splits-and-Trading-Volume).

Along with its daily timestamp as row index, each row contains the following inputs:

- Open: price at which the stock was traded when market opened
- High: highest price at which the stock was traded during the day
- Low: lowest price at which the stock was traded during the day
- Close: price at which the stock was traded when market closed
- Volume: number of stocks traded during the day
- Dividends: amount of money paid by Apple to its stockholders per share
- Splits: see note below
- Split/dividend adjusted open: see note below
- Split/dividend adjusted high: see note below
- Split/dividend adjusted low: see note below

- Split/dividend adjusted close: see note below
- Split/dividend adjusted volume: see note below

Note regarding split/dividend adjusted price: for an intuitive explanation of how this work, please watch the following video from Udacity's Machine Learning for Trading course: https://youtu.be/M2res0zhqjo

The data per se - as it is given - offers little useful information until it has been preprocessed in order to extract relevant information out of the given data. We postpone documenting such preprocessing steps to the Methodology section.

The only useful information contained in the raw data is the adjusted volume. From now on, we will just refer to it as volume instead of adjusted volume in that it is the only data regarding volume that we are interested in.

The reason why we are interested in volume data is that our fundamental assumption is that price changes are driven by supply and demand and the number of stocks traded indicates whether the price changes are due to a handful of players or to a much larger crowd.

For example, let's imagine a market where there are very few buyers and sellers. Then, the price will be determined by the supply and demand of those few players. On the contrary, if the crowd trading some market is very large, the equilibrium price will be the result of much wider negotiations. Therefore, we tend to assume that large volumes contribute to more reliable buying and selling signals. In any case, the size of the market - i.e. the number of stocks traded - does matter and we will try to use it to predict price movements, together with other human-engineered features.

Here are a few observations regarding the volume statistics:

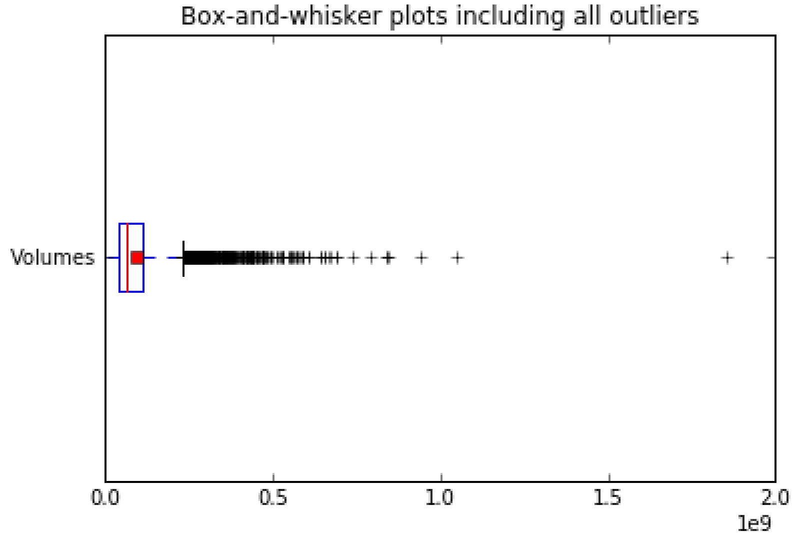
- There are 9011 data points, one per trading day from 1980-12-12 to 2016-09-06
- The mean volume is about 91 mln and the median volume is about 63 mln
- The standard deviation of the volume is about 88 mln, which looks pretty high
- The skewness is above 3, which looks high too. We can also notice that the mean is much higher than the median. This suggests us that there are some large outliers on the right side of the distribution.
- The kurtosis is above 28, which is a very high reading. This indicates the tails of the distribution are quite fat.
- The minimum volume was around 250k on 1985-09-27, so during the 80s
- The maximum volume was around 1.9bn (!) on 2000-09-29, during the tech bubble
- There are 616 outliers and 191 major outliers. That is respectively 6.8% and 2.1% of the observations, which looks high. Also, given the outlier lower bound is negative, we infer that all outliers are on the right side of the distribution, since it is not possible the number of stocks traded on some given day is below zero. This tells us there were several days

of extraordinary activity in the trading of Apple stocks and it is certainly something to keep in mind.

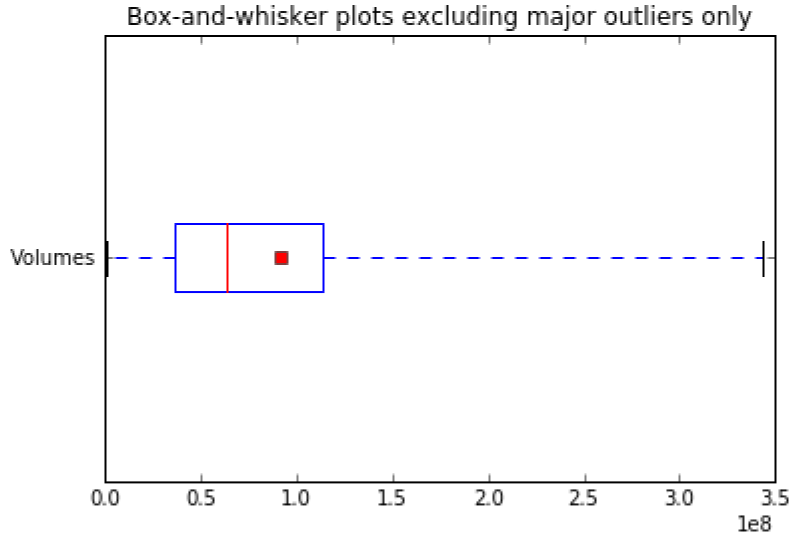
Please see table in box n.3 in the notebook.

Exploratory Visualization

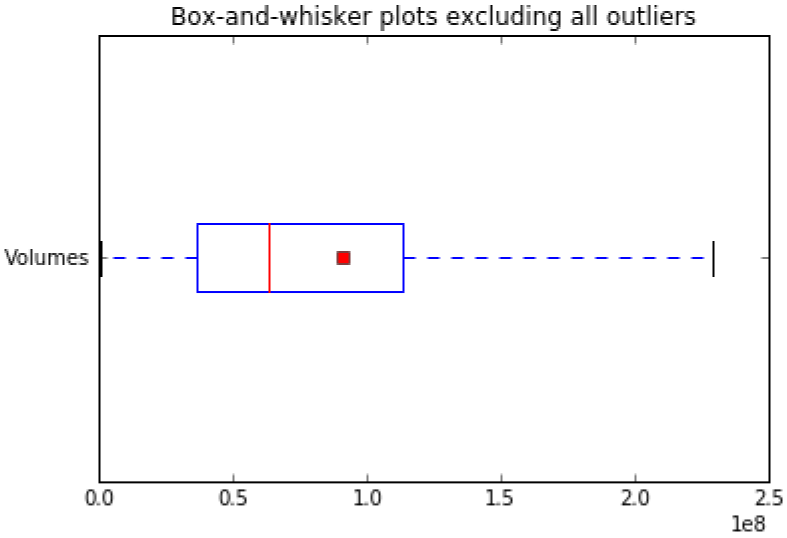
As we can see from the box-and-whisker plot below there are many outliers and some of them are completely elsewhere with respect to the rest of the sample. This confirms the impression we got in the Data Exploration subsection.



The plot changes substantially when we remove all major outliers from the picture, i.e. all outliers below Q1 - IQR * 3 and above Q3 + IQR * 3.

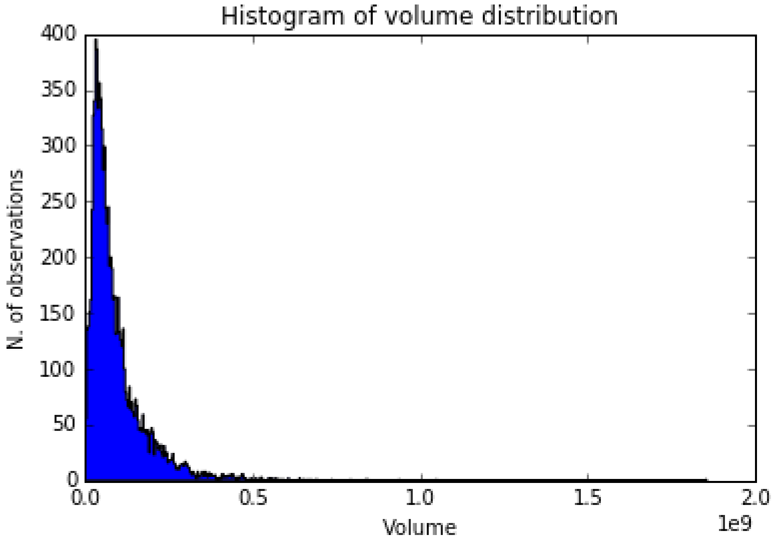


Eventually, the plot changes even more when we remove all outliers - both major and minor - from the picture, i.e. all outliers below Q1 - IQR * 1.5 and above Q3 + IQR * 1.5.

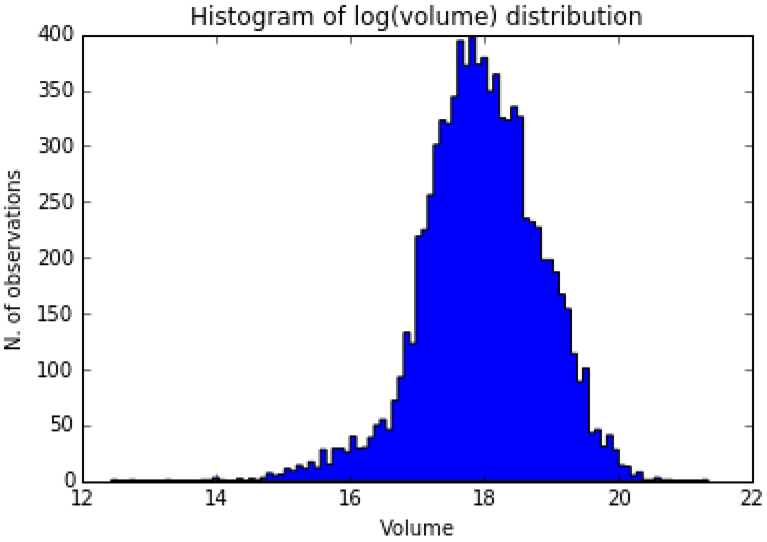


Such visualization gives us a visual intuition of what we observed about outliers in the Data Exploration subsection. The number of outliers is high and this will have to be taken into consideration when building the model.

Visualizing the histogram in order to get a visual intuition of the distribution of our volume data confirms what we observed in the Data Exploration subsection. The distribution is clearly positively skewed and it has a very long right tail. Also, it seems the distribution could be approximated by a lognormal distribution.



This is confirmed by the fact that if we do the same plot using the natural logarithm of the data we have a relatively nice bell curve. We will keep this in mind and see if it is of any help for the rest of the project.



Algorithms and Techniques

Algorithms we will use:

- Logistic Regression
- Support Vector Machines
- K-Nearest Neighbors

There exist countless more machine learning algorithms but for the sake of this project we had to choose a few. Our arbitrary choice fell on this shortlist for a couple of reason: simplicity and diversity. As far as simplicity is concerned, we believe the simpler the model the more chances it has to work. This is often true both in science and in finance. As far as diversity is concerned, we choose a mix of parametric (e.g. logistic regression) and non-parametric models (e.g. SVMs, KNN). We did not include any unsupervised learning method in that the nature of our problem is such that we are in possession of the labels for all training sets.

General note on cross-validation for all algorithms

Given the nature of our data, our choice will be to train our models on some given subset of the whole dataset at regular intervals (training set) and test its outcome on some immediately subsequent unseen subset (test set). This is known as roll forward cross validation. Its rationale is that learning some model on the whole dataset is more prone to overfitting than learning some model multiple times on small subsets of the data, thus enabling to run multiple tests with the same data.

The reason why this rolling window or roll forward cross validation is a common practice in the case of financial time series is that we really do not want to look into the future while training some model. Hence, it is of utmost importance that the training sets always precede their respective test set.

Logistic regression

The reason for using logistic regression is to classify the data points to determine whether the stock should be bought or sold. We will train the model on the training sets based on the human-engineered features discussed in the Methodology section.

For each data point, we will assign some probability of profiting by buying and some probability of profiting by selling based on whether buying or selling the stock would have made a profit or loss. Eventually, we will label data points in the training set by using a sigmoid function and choosing some minimum threshold in order to classify data points in the ‘buy’ class or in the ‘sell’ class and put all other data points in the ‘do nothing’ class.

Support vector machines

The reason for using support vector machines is again to classify the data points to determine whether the the stock should be bought or sold. The purpose is the same as logistic regression but we use a different approach here: we aim at finding a decision boundary that maximizes the margin, i.e. finding a line or hyperplane that is consistent with the data while committing the least to it. The reason why we want to commit the least to our data is that we want to avoid overfitting.

Since it is possible that our data is not linearly separable given out human-engineered features, we will consider using the kernel trick, i.e. we will use some function to add some features by using some linear transformation of our own features so that our non-linearly separable data becomes linearly separable. This is necessary with SVMs in that this algorithm puts first and foremost correct classification and only after it maximizes the margin. We will also play with the C and gamma parameters in order to find a model that fits the training data well enough but without falling into the trap of overfitting.

Nearest neighbors classification

The reason for using neighbors-based classification is again to classify the data points to determine whether the stock should be bought or sold, but also to diversify the classification approach we will use with logistic regression and support vector machines. By using this algorithm, we do not attempt to construct a general internal model, but simply store instances of the training data and classify via a simple majority vote of the nearest neighbors of each point.

We will try both the basic uniform weighting and the distance-based weighting. In the case of uniform weighting, the value assigned to a query point will be computed from a simple majority vote of the nearest neighbors. In the case of distance-based weighting, we will weight the neighbors such that nearer neighbors contribute more to the fit.

Benchmark

A common practice in the domain of finance is to consider the performance you would have gotten by just buying the stock at the beginning of the period you are examining and holding it until the end. In our case, we will use the metrics we have listed in the Definition section and see how our models fare against these.

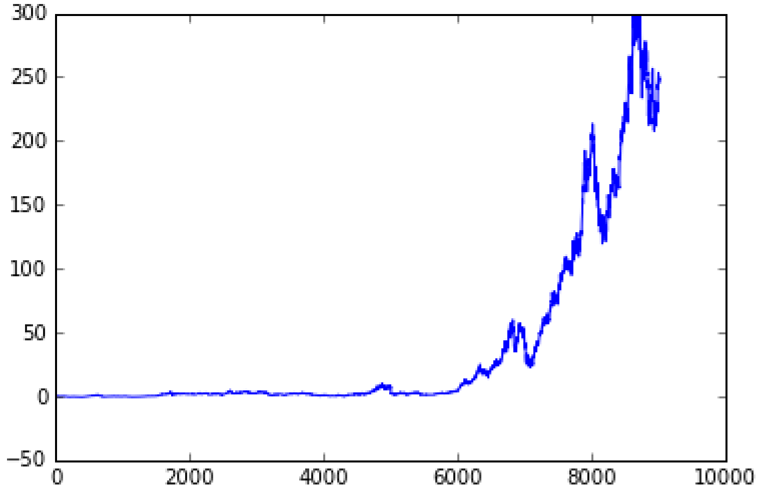
In order to be able to use our metrics for benchmarking purposes, we had to do some preprocessing on our data. We leave the discussion on our preprocessing steps to the Methodology section.

Among these values, the ones that matter the most to us are the information ratio, the adjusted Sterling ratio and the Omega ratio.

The information ratio matters in that it divides the average annual return by the annual standard deviation. In this particular case, we can see the importance of it since we have a high annual return - almost 27% - but even higher standard deviation, which gives us an idea of the risk we took in order to get the reward we got. Therefore, dividing the average return by the standard deviation of the returns seems to be a sound way to measure risk-adjusted returns. Ideally, we would like to have an information ratio greater than 2 for our model.

The adjusted Sterling ratio is quite similar to the information ratio but it differs from it by dividing the average annual return by the annual standard deviation of the losses only, which is interesting in that we might not care about the standard deviation as long it goes in our favour.

The omega ratio also matters since it divides the amount of money we made when we won by the amount of money we lost. For our model, we would be happy to have an omega ratio greater than 3.



Graph of cumulative profits over time

Please see the table in box n.10 in the notebook.

Methodology

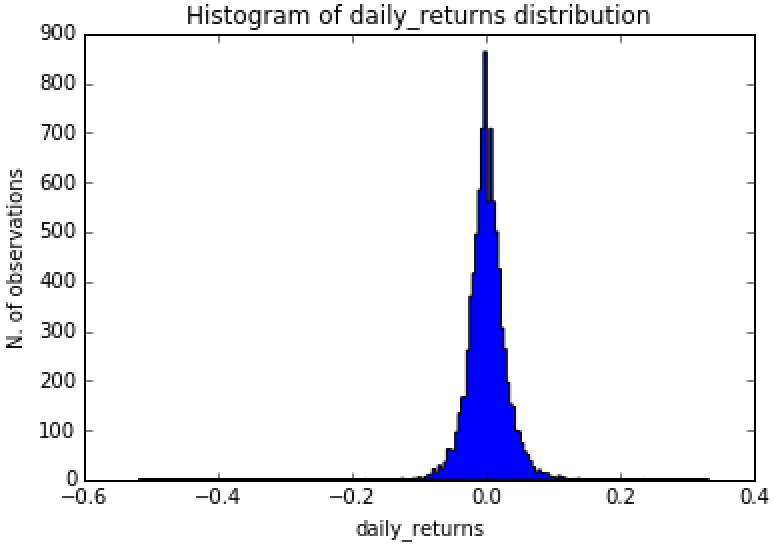
Data Preprocessing

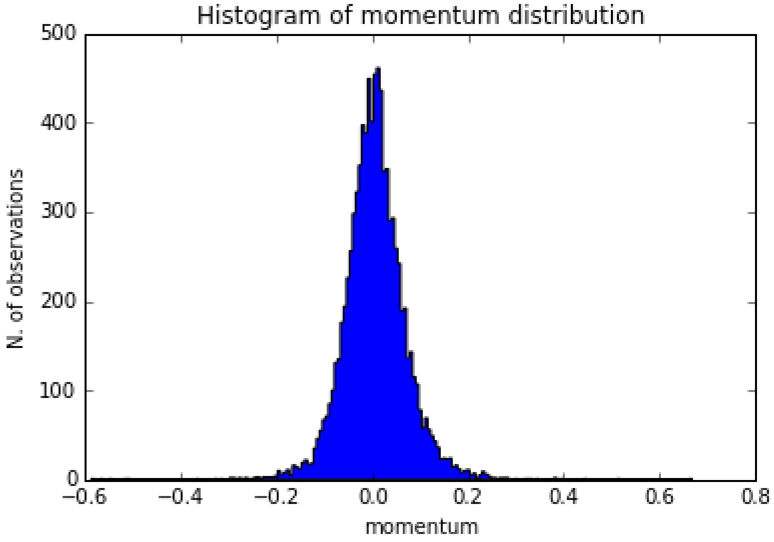
As we anticipated in earlier sections, our data require a few preprocessing steps in that the raw data per se contains very little. However, there is a lot more information we can extract by applying some feature transformation to our raw data. We shall list the preprocessing steps we put in place in order to ease the process of handling our data and squeeze relevant information out of it:

- Keep only adj_close and adj_volume, remove all other useless columns. All the information needed is contained by the selected columns.
- Reverse order from-most-recent-to-least-recent (Z -> A) to from-least-recent-to-most-recent (A -> Z).This will ease data handling due to the roll forward cross validation we chose and discussed about in our Analysis section, in the Algorithms and Techniques subsection.
- Compute daily returns. How much the stock price changed with respect to the previous day since the variation of the price tells a lot more than the price per se.
- Compute cumulative returns. How much the stock price changed with respect to the beginning of the period. This defines how much we would have made or lost (in percentage) by owning the stock at time t given we bought it at the beginning of the period.
- Compute momentum. How much the stock price has changed over some number of days. This measures the steepness of the cumulative returns line over some number of days.
- Compute momentum * volume. Scale momentum by volume so that trading days with more transactions acquire more weight than trading days with less transactions.
- Normalize all features. Scale all values so that all observations are between zero and one for learning algorithms requiring features scaling such as support vector machines.
- Remove outliers. Keep only data points that could be considered 'normal' market conditions.

This last point is directly related with the abnormalities we observed in our Data Exploration section regarding the high number of outliers. Now that we have our new human-engineered features, we shall perform some additional data exploration on these in order to verify whether we can get some insight on the data by looking at their distribution. In particular, we aim at spotting outliers in order to consider removing abnormalities from our dataset.

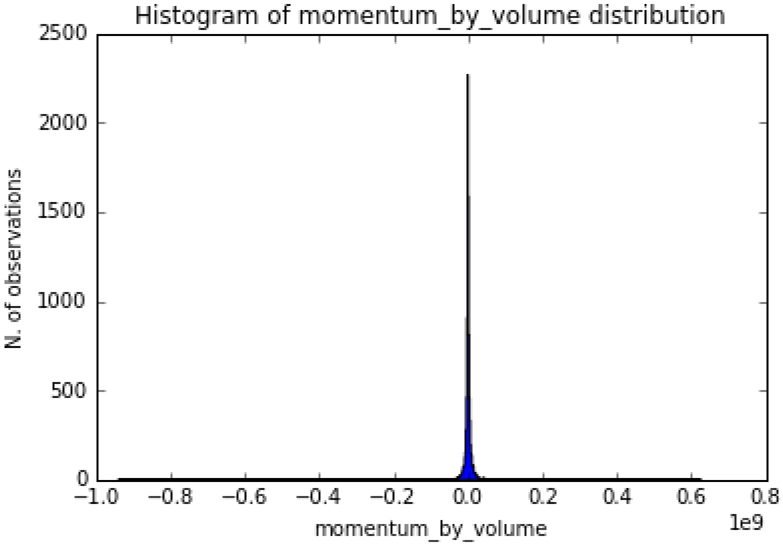
It can be observed that daily returns and momentum distributions share a similar distribution. Both have their mean centered around 0, both have relatively long tails due to the distance of major outliers from the mean and both have a quite symmetrical shape, except that the daily returns is slightly negatively skewed whereas the momentum distribution is very slightly positively skewed.



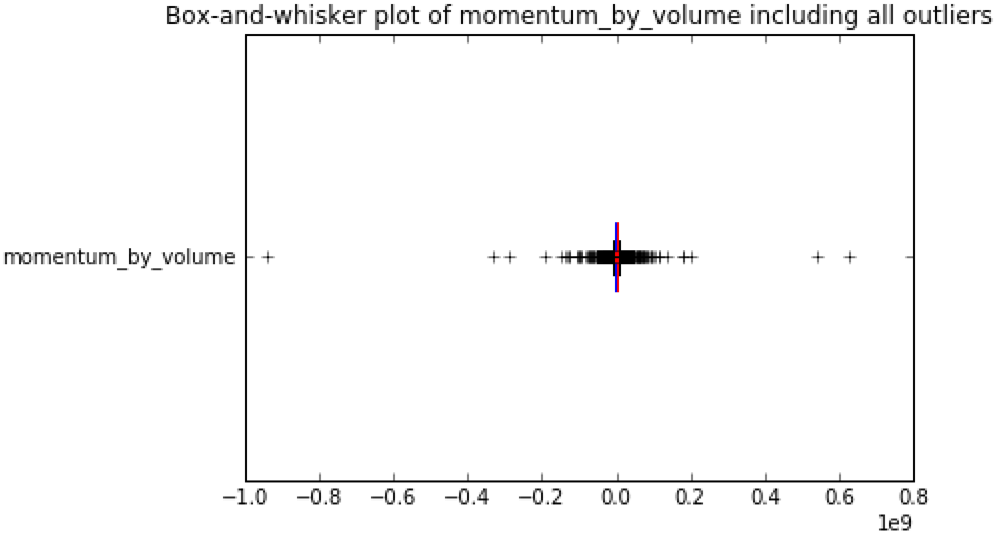


The similarities should not surprise us in that both features share a similar nature: daily returns tell us the percent change in the stock price from the previous day, while momentum tells us the percent change in the stock price over the last 5 trading days. What we can observe by the difference in skewness is that some violent negative returns make the left tail longer and consequently the skewness negative, but such violent negative daily returns tend to be recovered over a slightly longer 5-day term.

In the case of the momentum by volume distribution we can observe the number of outliers is very high, thus making the tail of our distribution very long. Interestingly, while the momentum skewness is slightly positive and the volume skewness is clearly positive, the momentum by volume skewness is very negative. This shows how violent price declines are a lot more frequent than violent price increases, confirming the saying according to which the market goes up by the stairs and down by the elevator.



It should also be noted that the number of outliers is a lot higher for the momentum by volume feature than for other features. This might be explained by the fact that the multiplication of two existing features results in larger values at the extremes, thus increasing the number of outliers. We will keep this in mind when we decide which data points to keep in our training sets based on whether we decide to consider them an outlier or not.



Training set and test set generation

The nature of our problem is such that it is crucial to pay very close attention to how we create our training and test sets as this choice alone could make all our models prone to overfitting. In particular - as Georgia Tech's professor Tucker Balch points out in the Machine Learning for Trading course - we do not want to make the mistake of looking into the future. Therefore, when doing cross-validation - i.e. partitioning the dataset into sub datasets - it is of utmost importance to make sure that the training set always precedes the test set. This cross-validation method is called roll-forward cross-validation.

The way we have implemented this is through our generate_sets function, which takes as arguments:

- a dataframe of features,
- the size of the training sample (window_size),
- the frequency of re-training the model(step_size).

For example, with window_size=90 and step_size=6, on day 91 we train the model on the subset of our features from day 1 to day 90 and use it to predict days 91 to 96. Then, on day 97 we retrain the model from day 7 to 96 and use to predict days 96 to 101, and so on so forth.

Dataset labeling

The way we choose to label our data is to create 3 classes: -1, 0 and 1:

- -1 when the daily return of the following day is negative
- 0 when the daily return of the following day is zero
- 1 when the daily return of the following day is positive

This was done by computing the sign of daily returns and shifting them by -1.

Dealing with outliers

When it comes to the choice of how to deal with outliers, we considered several options:

- take them out of the dataset and behave as they did not exist,
- keep them into the dataset but do not trust the model's predicted label if there are too many of them,
- keep them into the dataset but do not trust the model's predicted label even if there is just one them.

Taking them away from the dataset does not seem wise to use since they are part of the dataset and arguably they mean something. Not trusting the model's predicted labels if there is a certain number of outliers in the dataset does not look like a good alternative either in that there are many of them in the dataset, which would decrease greatly the number of trading opportunities and prevent us from receiving potentially useful insight.

For these reasons, we opted for keeping them into the dataset but pruning them by setting them equal to the outlier upper bound when they were on the right side of the distribution and to the lower bound when they were on the left side of distribution. As a result, after scaling all left outliers take the value of 0 and all right outliers take the value of 1. The benefit of such move is that outliers stay in the dataset but do not have the effect of squeezing the bulk of the distribution into an artificially dense area.

Feature scaling

Features are scaled using the minmax scaler so that all values are between 0 and 1. The reason for this is that some algorithms - like SVM - would give excessive weight to some features just because of the range of their values is wider than others.

It is interesting to notice that we could not scale the test sets the same way we used with the training sets. With the training sets, we could simply prune the outliers with the function we created. However, if we had done the same with the test sets, the outliers would have been identified based on the data contained in the test set only. This would have been a problem for two reasons:

- the training set and test set would have two different outlier lower and upper bounds - thus two different ranges - which is not logical since both should have the same lower and upper bounds and range, in order to compare comparables
- computing the lower and upper bound of the test sets would imply that all its values are known at the beginning of the test set period, but this is not the case as only training set values are known at decision making time

For these reasons, it appeared a lot more logical to compute the scaled values of the test set using the lower and upper bound and range of the training set as follows:

scaled value = (test set value - training set min) / (training set max - training set min)

Implementation

We created a run_model function that takes as input:

- one model
- a list of training sets
- a list of test sets
- a dataframe of labels

This function:

- iterates over the couples of training and test set,
- fits the model on each training window,
- tests it on the test window,
- compares the predicted outcome versus the actual outcome within the test window
- computes the profit or loss made on each test example within the test window
- returns a dataframe of accuracy values and a dataframe of profit and loss values over all test windows

The models from sklearn we tested are:

- linear_model.LogisticRegression()
- svm.SVC()
- neighbors.KNeighborsClassifier()

Refinement

The initial solutions computed by our run_model function produced the following results:

Logistic Regression
Please see the table in box n. 29 in the notebook.

Support Vector Machines
Please see the table in box n. 30 in the notebook.

Nearest Neighbors
Please see the table in box n. 31 in the notebook.

In order to try to improve the performance of our model we used grid search cross validation. In the case of logistic regression and SVM we used several different regularization parameters (C), while in the case of KNN we used several different numbers of neighbors to pass to the classifier in order to fit the model.

We can see from the tables below that the performance on the test set substantially increase as the regularization parameter C is decreased, i.e. as the classification boundaries are a closer fit to their respective training set.

Please see the table in box n. 33 and 34 in the notebook.

As far as the KNN performance is concerned, we can see that the performance on the test set is substantially better with the highest value of n_neighbors.

Please see the table in box n. 35 in the notebook.

Results

Model Evaluation and Validation

The final model is not reasonable and does not align with solution expectations. The graph of cumulative profits alone is enough to give an intuition of why. A robust model should return some smooth upward trending line. On the contrary, all models we examined have a graph of cumulative profits that looks more like a roller-coaster, which clearly does not indicate the robustness we would like to see. All models yield lower returns - often even negative - with same or higher standard deviation, which implies lower risk-adjusted returns. All risk-adjusted returns metrics are lower than the benchmark.

The model was trained on several different training set and tested on several different test sets as a result of the method we used to generate training sets and test sets, which we explained in the Methodology section.

The model is clearly not robust enough. Therefore, we cannot say it can be trusted.

Justification

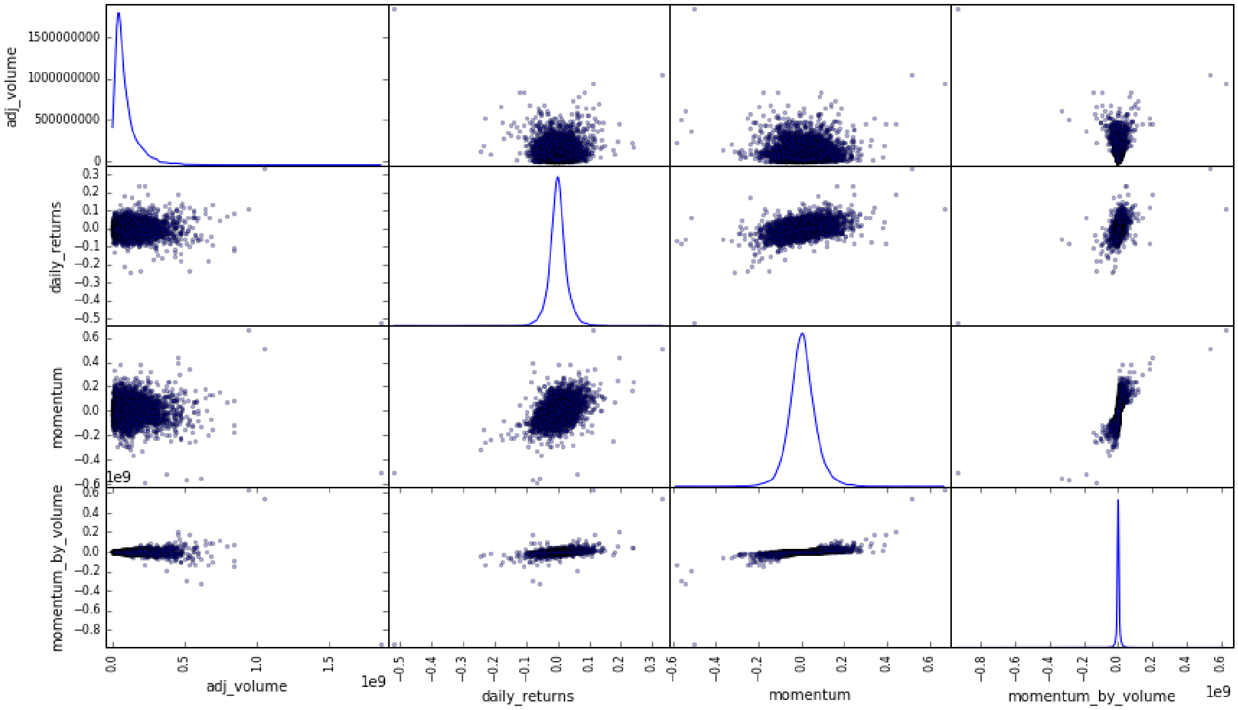
The final results are much weaker than the benchmark results reported earlier. The model yielding the best results is logistic regression with regularization parameter C = 0.01. Over the whole duration of the dataset, this final model yields an average annual return of 19.2% with an annual standard deviation of 47.1%, which results in an information ratio of 0.42, an adjusted sterling ratio of 0.60 and an omega ratio of 1.08, whereas our benchmark yields an average annual return of 26.6% with an annual standard deviation of 46.9%, which results in an information ratio of 0.57, an adjusted sterling ratio of 0.60 and an omega ratio of 1.11.

The effect of compound interest is such that the final profit our performance is a staggering 24929% versus a less impressive 620% over a 26-year frame. For these reasons, we do not consider the final solution to be significant enough to have solved the problem.

Conclusion

Free-form Visualization

The scatter matrix below enables us to look for potential correlations among features:



The correlations between momentum and momentum_by_volume and between adj_volume and momentum_by_volume should be ignored since momentum_by_volume is a combination of momentum and adj_volume.

There appears to be some correlation between daily_returns and momentum, but it should be noted that - as we pointed out above - daily_returns and momentum share a similar nature: daily returns tell us the percent change in the stock price from the previous day, while momentum tells us the percent change in the stock price over the last 5 trading days. Hence, daily_returns values contribute to determine momentum values.

Moreover, the nature of our problem is such that it is of no use to know that daily_returns and some other feature are correlated at time t. Instead, what we need to know is whether the information we have at time t is enough to predict if the daily_returns value will be positive or negative at time t + delta. For example, we would like to know if there is some correlation between momentum_by_volume and daily_returns at time t + 1:

From the scatter above, we can notice than it it becomes harder to see some immediately recognizable correlation when we look at the daily_returns at time t + 1 instead of those at time t.

Reflection

We started by defining the problem we aimed at solving. Then, we declared the metrics we would look at in order to evaluate the quality of our solution. After that, we did some exploratory data analysis and visualization. Next, we chose the algorithms and techniques we would test, specified their characteristics and explained our choice of using roll forward cross validation based on the nature of the problem. Subsequently, we declared which benchmark we would use to measure the performance of our models and explained why. Then, we preprocessed our data, generated some features out of our dataset, generated our training sets and test sets, labeled our data, dealt with the outliers and normalized our features. After that, we ran our models on all couples of training sets and test sets with their default argument. The next step was to run the same models but using grid search cross validation in order to fine tune their parameters. Eventually, we evaluated our results using our initially declared metrics versus the benchmark we chose in the beginning.

One of the most interesting parts we found in this project was defining the metrics to base our evaluations on. What made this particular aspect interesting is the fact that it is where machine learning meets the application domain, which is different for any domain. We found it very interesting to apply our domain knowledge to this machine learning problem.

One difficult aspect of the project was to decide how to deal with outliers. It took a considerable amount of reflection to think of some reasonable way to deal with those. Although outliers are often taken out of the dataset, our understanding of the domain suggested us that it would not be wise to ignore them because we understand outliers seen at time t do have an impact on the data at time t + delta. At the same time, they have the power to distort the data, especially when normalized. Hence, finding the good balance between not ignoring them but not giving them to much weight either was a little challenging.

The final model and solution do not fit my expectations for the problem, hence it should not be used to solve the problem. However, this project gives us a sound research framework to find other models. As Rishi Narang points out in “Inside the Black Box”, the field of quantitative finance is a field where a researcher should confident enough in his skills to believe there exist robust models to find and humble enough to know that out of many explored models only a few will actually be robust.

Improvement

Certainly there are countless improvement that could be tested. All sorts of other classifiers and regressors might be tested, as well as ensemble learners. Another option would be reinforcement learning, in particular Q-learning: we could create an agent which learns the rewards and penalties associated with each state and acts on the test set based on the Q-table it learned from the training set. The metaphor we may use is that of a self-driving car which gets rewarded for driving safely and penalized for crashing, causing damages to people or things, and breaking traffic rules. Accordingly, we could reward our agent when it makes money and penalize it when it loses money.

Another method we might use is ensemble learning. The reason for using a ensemble learning would be to combine the predictions of all other learning algorithm in order to improve generalizability / robustness over a single estimator. Since all learning algorithms have advantages and disadvantages that often differ from each other, by building several estimators independently and averaging their predictions we aim at having a combined estimator such that its variance is lower than that of any other single base estimator.

The way we could apply this averaging method to our problem is to build one portfolio made up of all models. Each model will have equal weight and will perform the action suggested by the outcome of its respective training: buy, sell or do nothing.

Example: Given a portfolio of 100$ and 4 models A, B, C and D, if model A and B say 'buy', model C says 'sell' and model D says 'do nothing', our ensemble model will suggest to buy 50$ of Apple stock due to A and B's recommendation, sell 25$ of Apple stock due to C's recommendation and do nothing with the remaining 25$ due to D's recommendation. Eventually, the 50$ of buying and the 25$ of selling will partially offset each other, resulting in 25$ of buying only. The clear consequence is that by listening to the opinion of four different learning algorithms the variability of the returns is reduced.

We will certainly keep all this in mind for further development.