Market Making with Machine Learning Methods

Kapil Kanagal Yu Wu Kevin Chen
{kkanagal,wuyu8,kchen42}@stanford.edu

June 10, 2017

Contents

1 Introduction **2**

2 Description of Strategy **2**
2.1 Literature Review 2
2.2 Data 3
2.3 Instruments Traded and Holding Periods 4

3 Signal Modeling **5**
3.1 Machine Learning Methods 5
3.1.1 Support Vector Machine 5
3.1.2 Random Forest 7
3.1.3 Stochastic Gradient Descent 8
3.2 Alpha Model 10
3.3 Risk Management Strategy 10

4 Results **11**
4.1 Difference in markets 11
4.2 Comparison of machine learning models 11
4.3 Analysis of the Random Forest version 12

5 Discussion **13**

1 Introduction

Our project seeks to use novel machine learning techniques to predict directional market movement for market makers. Specifically, our project seeks to find signals in the market order book – the book which contains order, pricing, and volume data – to predict the direction of asset price movement. Once our machine learning methods predict the directional movement of an asset price (either up, neutral, or down), it then reprices the orders we make. To better understand how this works, we must first examine the role of a market maker.

Market makers provide liquidity in a market – they ensure that buyers and sellers have the ability to convert their assets to cash and vice-versa. This is essential to the modern stock market, as without liquidity, it would be difficult to execute orders in a timely manner. To better understand the role of a market maker, let us explore the example in figure 1:

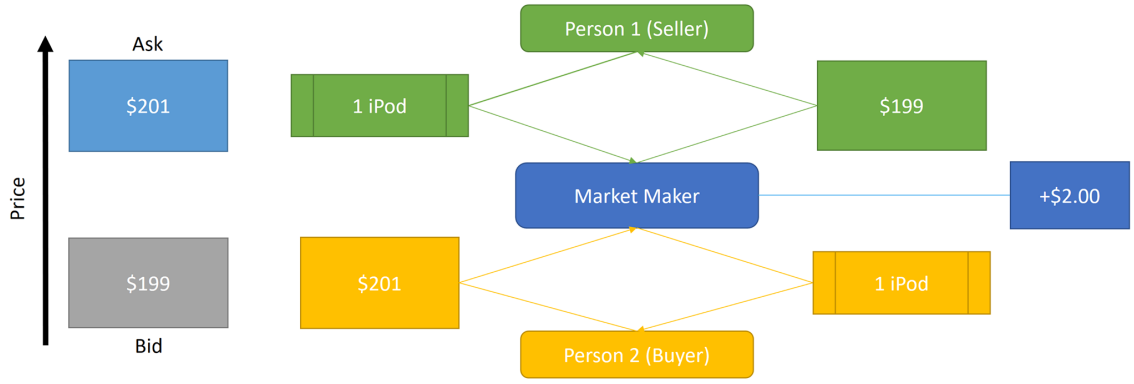


Figure 1: Market making example.

From this example, we see the following:

1. Person 1 is a seller who owns an iPod and wants cash for it
2. Person 2 is a buyer who has cash and wants an iPod
3. The market marker buys Person 1's iPod for $199 and then sells the iPod to Person 2 for $201.
4. This allows the market maker to make $2 on the bid-ask spread, where the bid price is $199 and the ask price is $201.

In practice, the market maker is able to do this very quickly (within a few seconds) and make a small profit on each trade. Thus, to make money market makers execute a high-frequency of trades. Our strategy combines machine learning methods that predict directional price movements in the market with market making to try and earn money off of the bid-ask spread. With this in mind, the sections below outline our trading model through a literature review, examine the data and assets we trade, evaluate the efficacy of our different models, and explore our backtested results, in order to assess our strategy.

2 Description of Strategy

2.1 Literature Review

Our strategy uses machine learning models to predict the evolution of the price of the stocks we're looking at based off of [LDZ+14]. We periodically sample the state of the market and use these models to output a signal indicating whether the price is expected to increase, decrease, or remain globally unchanged. Taking this evolution into account, we post ask and bid orders at a given price (which can for example be the best ask and bid quotes or second best bid and ask as we do in our strategy), adjusted for the evolution, in order to better capture the trend of the market.

While [LDZ+14] also uses a news aggregator signal for the assets they are looking at, we do not do so, as our trading platform, Thesys, does not allow us to efficiently incorporate web scraped news into signal generation. Furthermore, as discussed in [LDZ+14] the order book consists of two priority queues with each element of the queue representing a different level of the order book as shown in figure 2.

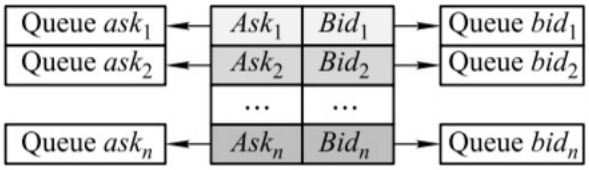


Figure 2: Order Book Structure

[LDZ+14] also introduces the concept of Order Book Pressure (OBP) to summarize the dynamic shape of the order book. If the sell side (ask_1) is bigger than the buy side (bid_1) at a threshold at time t, we expect the mid quote ($mid = \frac{bid_1 + ask_1}{2}$) to go down in the short period, and vice versa. If the queue size is within the upper and lower threshold, we expect the mid quote not to change. This is summarized in figure 3

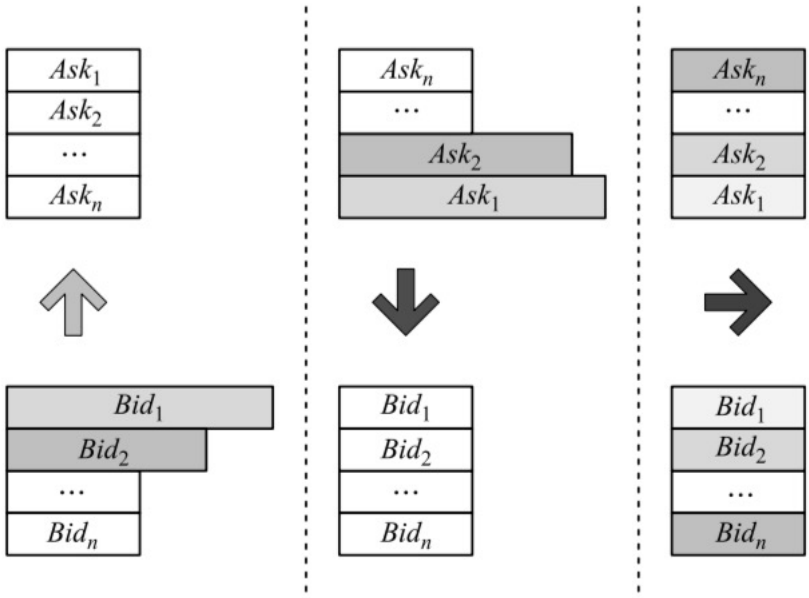


Figure 3: Order Book Pressure – the size of the box indicates the queue size at that level

As market makers, we make money when both the ask and bid orders we post are hit. In our strategy, in the event that only one of the two sides is hit, we decide to wait a given time for the other side to be hit (typically half the sampling period for our signal generation). If that wait isn't sufficient, we withdraw the one outstanding order and post new quotes, keeping our inventory.

Another important point of our trading algorithm is the notion of stop loss. In order to avoid extreme loss if we're trading against the market, we implement a stop loss function which will regularly check that we're within our boundaries, and will liquidate our inventory to start anew otherwise. [LDZ+14] uses a simple predefined stop loss trigger and does not unfold the discussion of other risk management facilities. Unlike [LDZ+14] we implement our own stop loss function in our algorithm below. This is discussed in section 3.3.

2.2 Data

We sample market book data from Thesys every minute, and get both the first L book levels queue sizes on both ask and bid sides, and the best ask and bid prices. This data is then processed to

obtain both the features and the signal used to train the different machine learning methods. Data is downloaded beforehand from Thesys, as getting it on the same day would give us Thesys-induced delays which are unrealistic in reality where hardware would prevent such a latency. We number each minute by the index t for $t \in \mathbb{N}$.

- Given a threshold θ, the signal is

$$Sig_t = \begin{cases} 1 & \text{for} \quad Midprice_t > \theta \\ 0 & \text{for} \quad -\theta < Midprice_t < \theta \\ -1 & \text{for} \quad Midprice_t < \theta \end{cases}$$

$$Midprice_t = \frac{BestBid_t + BestAsk_t}{2}$$

  The threshold has been chosen to be \$0.03 or \$0.04, yielding roughly equal proportions of increasing, decreasing, and constant signals.

- The features are a representation of the relative queue sizes of the bid and ask sides. The underlying idea is that if the number of bid orders is greater than the number of ask orders, supply and demand will naturally raise the price of the traded financial instrument, same goes in the other direction. On the other hand, if there is no clear difference in the number of ask and bid quotes, then the market is simply fluctuating and no signal will be generated from the noise. We generate $6L$ order book pressure features which are

$$OBP_t(n,l) = \frac{\sum_{\tau=0}^{n} \sum_{j=1}^{l} BidSize_{j,t-\tau}}{\sum_{\tau=0}^{n} \sum_{j=1}^{l} AskSize_{j,t-\tau}}$$
$$n \in \{1,2,3,5,10,15\}$$
$$l \in \{1,\dots,L\}$$

  where $BidSize_{j,t-\tau}$ denotes the bid queue size of level j obtained τ minutes before (same for $AskSize_{j,t-\tau}$). We chose $L = 5$ leading to 30 features, as described in the litterature.

Since queue size data is needed up to 15 minutes before actually trading, the trading strategy can't be used for the very beginning of the trading day, since we wouldn't want to train on possibly wrongful data from the previous day. This problem doesn't arise since we're only starting our strategy at 9:30 am, leaving us enough margin to get the data.

Other possibilities for the signal generation which we didn't use in our final algorithm would have been choosing the last executed order price instead of the midprice, or using a weighted average of the prices over the L first book levels.

The time scale for our algorithm is the following

- Parameters fitting for the machine learning models is done on 3 months of data.
- Training the algorithm, once the correct parameters have been found, is done on 15 days of data preceding the trading day.
- We tested our algorithm over the course of three months.

2.3 Instruments Traded and Holding Periods

The instruments we traded in simulation over 3 month period are AAPL, FB, ORCL, MSFT, and GOOG. The choice is more or less arbitrary since theoretically the market making strategy can be extended to all symbols. We also gathered data and run 1-month simulations from the top 5 companies ranked by market capitalization from the basic, health care, energy, and finance industry, and during the short interval most stocks from all industries have similar performance. Thus to keep the project with in the alloted time frame, we chose to focus on the 5 stocks listed for more in-depth analysis.

Our strategy runs from market open to market close every trading day. During the trading period, we post a pair of orders or post market orders to reduce inventory every minute for each stock we trade. The ideal scenario is that both the bid and ask orders are executed during each interval so we hold no inventory, but if only one side is executed we will cancel the unfilled orders. We will hold on to the inventory until our risk management module tells the strategy to reduce it. At the end of each trading period we will liquidate all inventory to avoid taking overnight risks.

3 Signal Modeling

3.1 Machine Learning Methods

In order to evaluate our model and ensure we were optimizing our results, we decided to use three different machine learning methods. Support Vector Machines (SVM) was the optimal method cited in [LDZ+14]. However, the authors in this paper did not explicity verify this result, instead citing other papers. Accordingly, we decided to use a more liberal machine learning method (Random Forest) and a more conservative machine learning method (Linear Stochastic Gradient Descent) to evaluate which model was most successful in market making.

For all three algorithms, the parameters fitting process was done with a sample of data collected over 3 months, with the signal and feature generation aforementioned. 80% of the sample was used to fit the models while the other 20% was used as a cross-validation set to test for overfitting. Algorithms are trained separately for each stock, so that the shape of the evolution of one doesn't impact the others.

3.1.1 Support Vector Machine

Support vector machine (henceforth referred to as SVM) is a machine learning technique that classifies the data samples into categories by finding the optimal separating hyperplanes. More specifically, support vector machine finds the hypoerplane that maximizes the margin, where margin is defined as the distance from the closest point of any class to the separating hyperplane. Intuitively speaking, the bigger the margin, the more room the classifier has for errors, and thus the classification is conceivably more reliable. For this project we used the `SVC` library from `scikit-learn` package. In the implementation of SVC, the problem of finding the optimal hyperplane formulated as the following optimization problem [PVG+11]:

$$\begin{aligned} \min\ & \frac{1}{2}w^T w + C\sum_{i=0}^{n}\zeta_i \\ \text{s.t. }\ & y_i(w^t\phi(x_i)+b) \geq 1-\zeta_i, i=1,\cdots,n \\ & \zeta_i \geq 0, i=1,\cdots,n \end{aligned}$$

The objective function takes into account not only the margin but also the weights of the features for normalization purposes. How important the margins are relative to the weights can be adjusted with the hyper parameter C. The dual of the above problem is

$$\begin{aligned} \min\ & \frac{1}{2}a^T Q a - e^T a \\ \text{s.t. }\ & y^T a = 0 \\ & 0 \leq a \leq C, \end{aligned}$$

where $Q(i,j) = y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel function for the inputs. The most common kernel used in SVM is the rbf kernel:

$$K(x_i, x_j) = \exp(-\gamma|x_i - x_j|^2)$$

And this is also used by our primary reference [LDZ+14]. Thus we will also use this kernel in our model. There are two hyperparameters we need to set: C and γ in the rbf function. To find the optimal choice of these parameters, we will also use the same methodology outlined in [LDZ+14]: we will train SVM with a variety of combination of these parameters, and select the best combination using the validation data. To evaluate the performance of SVM on the validation data, we take into accound the following three aspects:

1. accuracy of prediction (i.e. probability of prediction agreeing with actual signal)
2. probability of adverse prediction (i.e. probability of predicting up when the signal is down of vice versa)
3. conditional accuracy of bold predictions (i.e. the probability of signal being up when the prediction is up, and the same for down)

4. probability of having bold prediction

The latter two aspects were taken into account because we found that in practice just prediction accuracy is not enough to generate a practical model: the SVM could "cheat" to obtain high accuracy by almost always predicting neutral. So, in addition to accuracy, we want to encourage the classifier to make up or down predictions by rewarding correctly predicted ups or downs more than the neutrals. However, we also want to penalize wrong trend predictions, since in practice these would tend to lead to more losses than neutral predictions. Even with these adjustments, the model could still "cheat" by having only a few predictions in ups or downs when it is really confident to boost the conditional probability. Thus we will penalize this kind of behaviour by rewarding having more up or down predictions. Our performance function is as follows:

$$\begin{aligned}\text{performance} &= \text{accuracy of prediction} - \text{probability of adverse prediction} + \\ &\quad \text{conditional probability of bold prediction} + \text{probability of having bold prediction}\end{aligned}$$

Technically the optimal C and γ could be different for each stock, so finding the optimal parameters for each stock could be a time consuming process. However, we found that for many of the stocks we examined, $C = 10$ and $\gamma = 0.005$ works fairly well, so this will be the parameters we use for all of the stocks we look at. The confusion matrix for AAPL is shown in figure 4:

	pred - 1	pred 0	pred 1
actual -1	398	493	435
actual 0	338	939	759
actual 1	251	483	734

Figure 4: Confusion matrix for AAPL validation data using threshold of 3 cent and $C = 10$, $\gamma = 0.005$ in terms of frequency

The histogram for the distribution of predictions and actual signals for the validation data is shown in figure 5

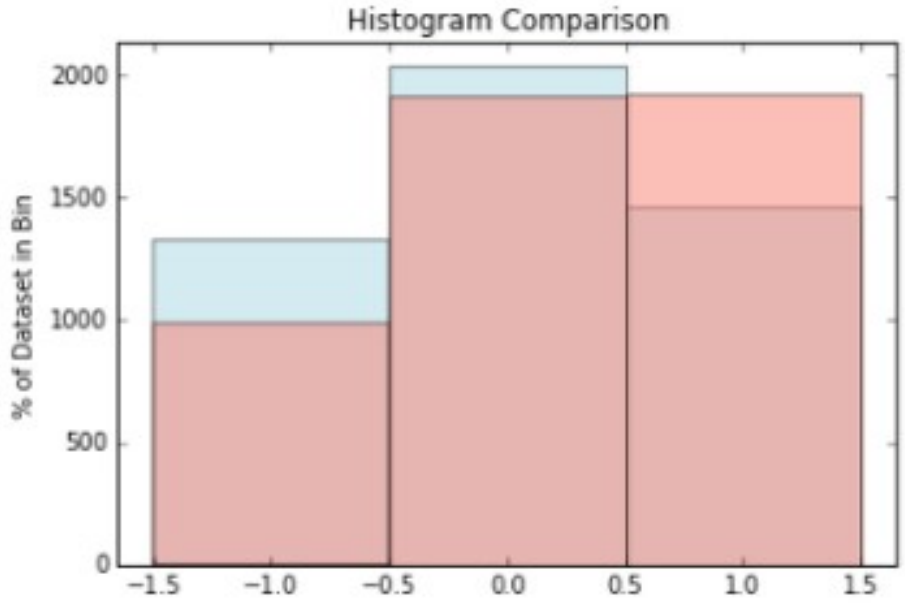


Figure 5: Histogram of prediction and actual signal distributions with SVM

We can see that the distribution of the prediction and actual signal matches fairly well, but the accuracy of prediction is no where as good as the histogram suggests. From the confusion

matrix table we can see that the conditional accuracy of up or down signal is only about 40%, while the number of adverse predictions are quite significant; in fact, there are more up than down predictions when the actual signal is down. This unsatisfactory result of our signals shaped how we post orders during actual trading and how we approached problem of inventory and risk management, which will be discussed in more detail in section 3.2 and section 3.3.

3.1.2 Random Forest

Random forest is based on decision trees, where each node considers a certain feature chosen according to its improvement on a given loss function. Since decision trees tend to overfit their training set, random trees add a random part in that several decision trees are generated from bootstrap samples of data and then averaged. In the process, the variance is greatly reduced at the price of a slight increase in bias, producing a more robust model.

We used the `RandomForestClassifier` for the Python `scikit learn` library. Some of the parameters of importance are

- `num_estimators`: the number of trees to be generated. Since the decrease in variance varies slower than the increase in computational power, an average number of 100 is enough.
- `criterion`: the loss function to estimate to quality of the splits when selecting features. Both the Gini criterion and the entropy have been tested with no real change.
- `max_features`: the maximum number of features to consider. We chose it be equal to `n_features` to explore a maximum of possibilities.
- `max_depth`: the maximum depth of the tree, chosen to be default so that the tree is expanded until there aren't enough samples to fill the leaves.
- `min_samples_split`: the minimum amount of samples required to split an internal node. We chose to be twice the following parameter `min_samples_leaf` for the leaves to be possible.
- `min_samples_leaf`: the minimum amount of samples required to create a leaf of the tree. The default value is 1, however a value of 3 is preferred. Indeed, a smaller value will make the classifier more liberal, which already is the case. Choosing a slightly higher value for this parameter allows to counteract this fact and avoid excessive overfit.

When looking at the results, we first look at the contribution of each feature, as shown on the figure 6 where $Feature_{nL+l} = OBP(n, l)$.

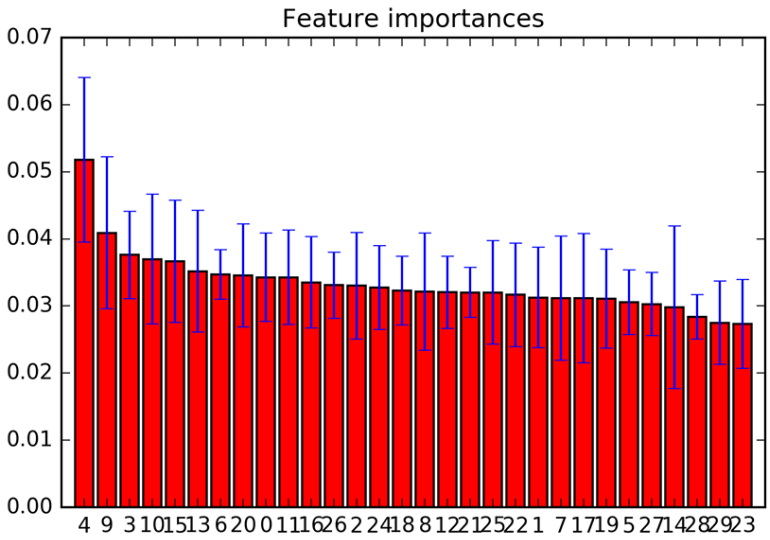


Figure 6: Contribution of random forest features.

As we can see, all features have roughly the same importance. Furthermore, there is no clear explanation as to why some would be more important than others, as seemingly similar features (e.g. features 4 and 29) don't share similar positions. This confirms the choice of considering all `n_features`.

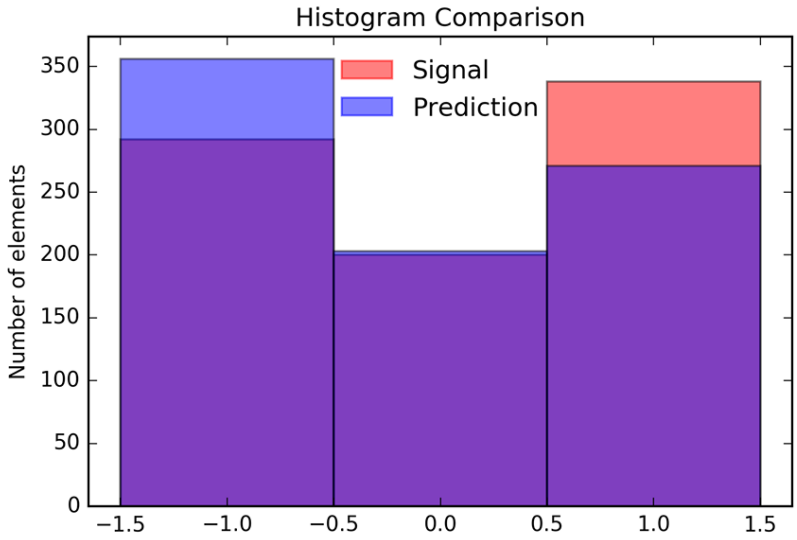


Figure 7: Prediction histogram with random forest.

Figure 7 shows the histogram of the predictions obtained with the random forest compared to the original signal used to train the model. We obtain a correctness of

$$\mathbb{P}(\text{signal} = \text{prediction}) = 52\%$$

which is better than the third that literature mentions, the details of which is

- $\mathbb{P}(\text{signal} = 1 \,|\, \text{prediction} = 1) = 58\%$
- $\mathbb{P}(\text{signal} = 0 \,|\, \text{prediction} = 0) = 37\%$
- $\mathbb{P}(\text{signal} = -1 \,|\, \text{prediction} = -1) = 55\%$

The random forest model is more liberal and gives better prediction when there actually is a trend in the evolution of the price, while having poorer results when the market is only fluctuating. However, predicting the correct evolution is more important since this is what generates profits in our algorithm.

3.1.3 Stochastic Gradient Descent

The third machine learning algorithm we decided to test on our data was Linear Stochastic Gradient Descent (SGD). To train our linear model on our data, we iteratively fit one hundred linear models on our 4830 data points in our 15 day rolling window. The first 4000 of these points were part of our training set and the other 830 points were part of our validation set. SGD uses gradient descent to find the minimum or maximum of a given function. In our case, we sought to minimize the log loss function on our data. The log loss function is a classification loss function used as an evaluation metric. Since we were trying to classify the signals as +1 (price going up), 0 (price staying neutral), or -1 (price going down) using our 4 cent threshold, the log loss function quantifies the accuracy of our classifier by penalizing the false classifications our linear model makes. Using SGD allows us to select the linear model that minimizes the number of incorrect predictions we make via the log loss function. The algorithm works as shown in figure 8.

- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta \nabla Q_i(w)$.

Figure 8: SGD Algorithm

In our code, we imported the linear SGD algorithm from the scikitlearn library in python. We note the following:

- w is a vector of zeros of size n
- n is the length of our training set (in our example n=4000)
- The stochastic gradient descent process is repeated 500 times to find the minimum of the log loss objective function
- $Q_i(w)$ is the log loss objective function (the gradient of this function is taken in the formula)
- η the learning rate is internally optimized by the scikitlearn library

Next, we evaluate the efficacy of our linear SGD model on the data and draw conclusions from these observations. To do so, we use data from the ticker IVV, which represents the S&P500, and simulate our linear model by running SGD. We then use the 830 points in our validation set to judge the accuracy of our model. We then make a histogram of our classifications versus the actual values to evaluate our model. One sample of this graph is shown in figure 9. The classification metrics are shown below:

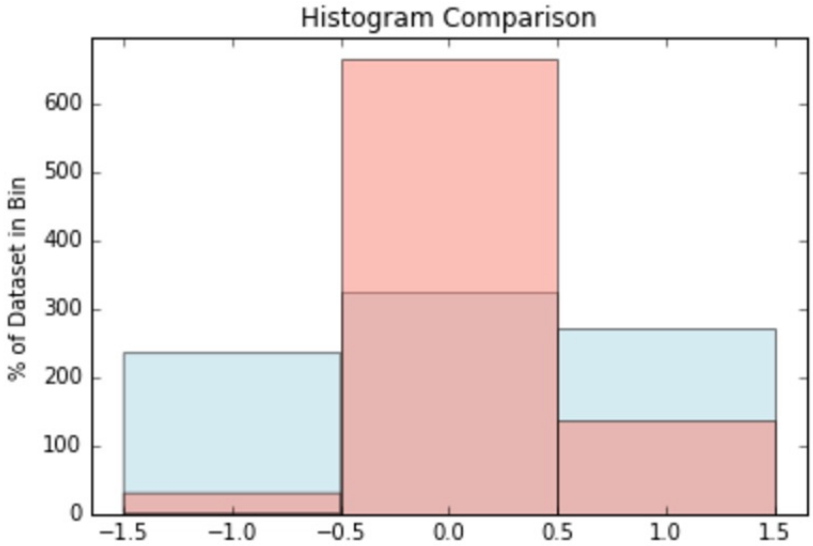


Figure 9: Prediction Histogram with linear SGD

- $\mathbb{P}(\text{signal} = \text{prediction}) = 38.733\%$
- $\mathbb{P}(\text{signal} = 1 \,|\, \text{prediction} = 1) = 14.033\%$
- $\mathbb{P}(\text{signal} = 0 \,|\, \text{prediction} = 0) = 82.486\%$
- $\mathbb{P}(\text{signal} = -1 \,|\, \text{prediction} = -1) = 7.183\%$

From the histogram above we note that our linear SGD model is quite conservative in its guesses. It is overpredicting the number of 0 (neutral) signals, while drastically underpredicting the number of +1 (up) and -1 (down) signals. This may be due to the conservative nature of a

linear model and how restrictive a linear model is compared to SVM or Random Forests. A linear model can be really limiting in trying to classify the data and we observe that through our results. It would be difficult to combine this machine learning method with SVMs or Random Forests, since we would not know when the linear SGD model is correct in classifying as a neutral signal and when it is incorrect in classifying a signal as neutral. This would make it difficult to decide which classification result (the result from the SVM, the result from the Random Forest, or the result from linear SGD) to use. Thus, though this model does an excellent job of classifying neutral signals, it does a very poor job of classifying up or down signals and accordingly, is not nearly as effective as the other two machine learning techniques.

3.2 Alpha Model

The market maker's profit comes from the bid-ask spread. When one pair of buy/sell orders gets executed, the market maker will make the difference between the ask and bid price posted. This will hold true as long as both orders get executed, so this strategy theoretically can be applied to any stocks. The frequency and bid-ask spread at which the orders can be posted is is up to the decision of the market maker as well, so one could post as frequently as possible and at as large of a spread as possible to maximize profit. However, one has to keep in mind that high posting frequencies and large spread would decrease the probability of the pair getting executed, so these two are also important parameters to adjust for a market making strategy. Since all of our data are gathered at one minute intervals, we will also post one pair of buy/sell orders per minute. Theoretically if we are able to obtain data at more granular levels we could post more frequently, but gathering and analyzing data on Thesys at finer intervals would be another very time consuming process and we feel it is beyond time frame we have for the project. As for the spread, originally we followed our reference's set up and used the following price for posting at period i:

$$p^i_{bid} = b_i + \mu * sign_{OBP}$$
$$p^i_{ask} = a_i + \mu * sign_{OBP}$$

where b_i and a_i are the best-bid and best-ask price at interval i, $sign_{OBP} = \{-1, 0, 1\}$ and μ is another parameter of the model we can tune. In practice we founds that this posting strategy would always incur losses. Since our prediction is not very reliable, posting aggressively will frequently result in us only executing one side of the trade and hold an inventory against the market. Instead we used the following posting scheme:

$$p^i_{bid} = b_i + \mu * \min(sign_{OBP}, 0)$$
$$p^i_{ask} = a_i + \mu * \max(sign_{OBP}, 0)$$

With this scheme, we will never post ask price below the best ask and we will never post the best bid above the best bid, which is less aggressive than the previous scheme. This will allow us to increase our profit when the trade goes through and decrease the risk we take if we guessed wrong at the expense of lowering the chance that we will get the pair executed. We found in experimentations that this scheme works better than the previous one, and so this is used for all the strategies in our simulation. In addition, we found in experiments that setting a_i and b_i to be the second best ask and bid price does not significantly impact the probability of execution, but the profit increased significantly when the trades go through. Thus for all of our strategies the reference price a_i and b_i is always the second best ask and bid. A dynamic posting strategy, for instance, one based on a stochastic model of order arrival [AS07], may significantly improve the performance of our strategy and would serve well as a future extension of this project.

A critical part of market making strategy is the management of inventory risk. The market maker could only make the spread when the market is stable and is oscillating regularly. If the market is trending, only one side of the pair posted will be executed, and the market maker will always hold inventory against the market, which may create huge losses if the inventory is not managed carefully. The risk management strategy we used will be discussed in the next section.

3.3 Risk Management Strategy

The inventory management strategy we used is very simple:

- If the current best bid/ask is better than the book price for the inventory, we will reduce it as much as we can
- If the unrealized loss calculated from marking our inventory to the best available price exceeds a certain threshold and we predict that it will get worse, then we will reduce it as much as we can

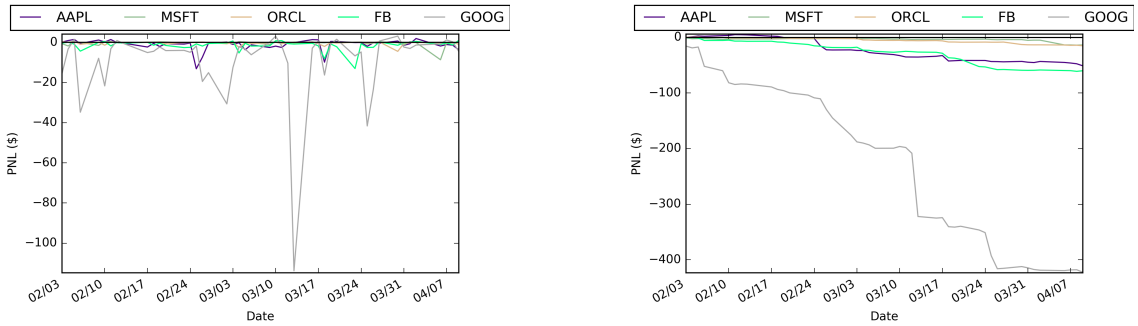
In practice we found that this stop loss strategy performs worse than even just holding on all inventory until the end of day to liquidate it. This is because our prediction is unreliable; very often we would reduce our inventory but in reality it would have been better to just hold on. Doing this seem to improve our average performance by quite a bit, but we are also more susceptible to large losses if market is trending for the major part of the day. We may get better performance with more dynamic stop loss mechanisms, such as the VPIN strategy suggested in [ELdPO10]. This is also another good future extension of this project.

4 Results

In this section, we explore the results from the backtests of our strategy on our selected data.

4.1 Difference in markets

The algorithm mentioned in the literature was tested on the Tokyo NIKKEI. One first thing we notice is the discrepancies between that market and the NASDAQ we're trading on. An example is the case of the Google stock (ticker GOOG). It performs exceptionally worse than the other stocks in the same IT industry we picked. As can be seen on figure 10, the Google stock has more frequent dips in PNL which has a lasting negative impact on the cumulative PNL over the 3-month trading period.



(a) Daily PNL.

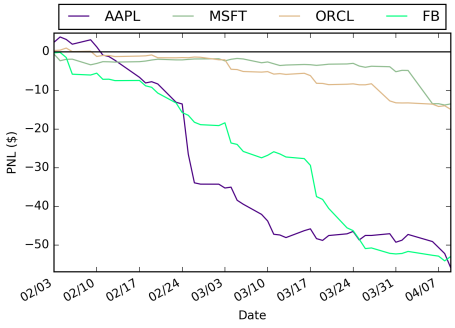
(b) Cumulative PNL.

Figure 10: PNL for 5 IT stocks obtained with our trading strategy using SVM.

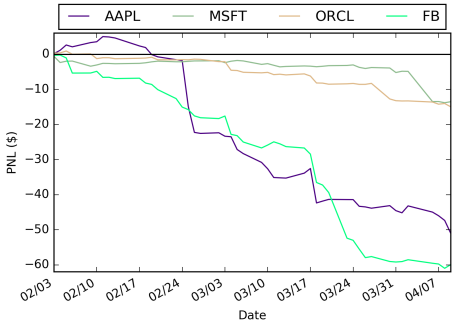
In all the following graphs, we removed Google from the pool of stocks we're trading on.

4.2 Comparison of machine learning models

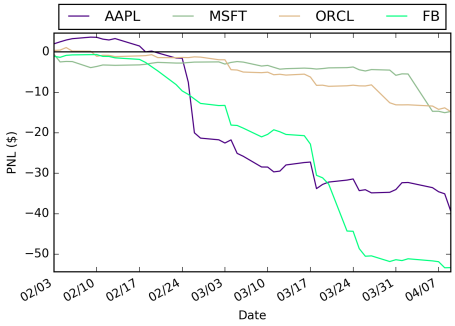
We compared four strategies, the three obtained from the different machine learning models plus the default strategy where there is no signal generation. The cumulative PNLs obtained are seen on figure 11.



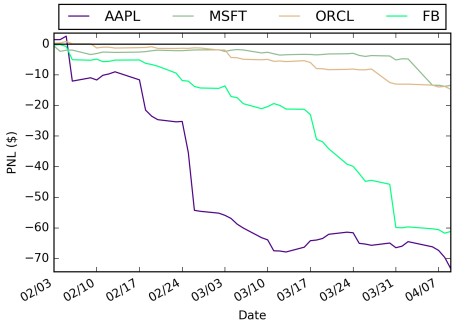
(a) No signal generation.



(b) Support Vector Machine.



(c) Random Forest.



(d) Stochastic Gradient Descent.

Figure 11: Cumulative PNLs obtained for 4 IT stocks with our trading strategy using different machine learning models.

We can first notice that all graphs have roughly the same pattern, implying that signal generation helps generate PNL but can't fundamentally change the profitability of a trading algorithm. The default version with no signal generation, which just posts quotes at the second best ask and bid prices, actually performs better than the Support Vector Machine and Stochastic Gradient Descent versions. Only the Random Forest prediction yields a slightly higher cumulative PNL.

SGD performs the worst out of all algorithms as a logical consequence of it being too conservative. Generating wrong predictions when the trend is either going up or down hurts more than having no signal at all since going against the market makes us lose more money. On the other hand, the RF version which boasts more than 50% correct prediction rate for changing trends naturally gives us more profits. The SGD version produces a slightly worse PNL than the default version, which infirms the saying that it's the best signal generating method as found in [LDZ+14]. That can be explained by different markets not exhibiting the same behavior.

4.3 Analysis of the Random Forest version

Looking further intot the version which gives the best results, we can look at the inventory, which may be a concern if it carries over for too long in our strategy.

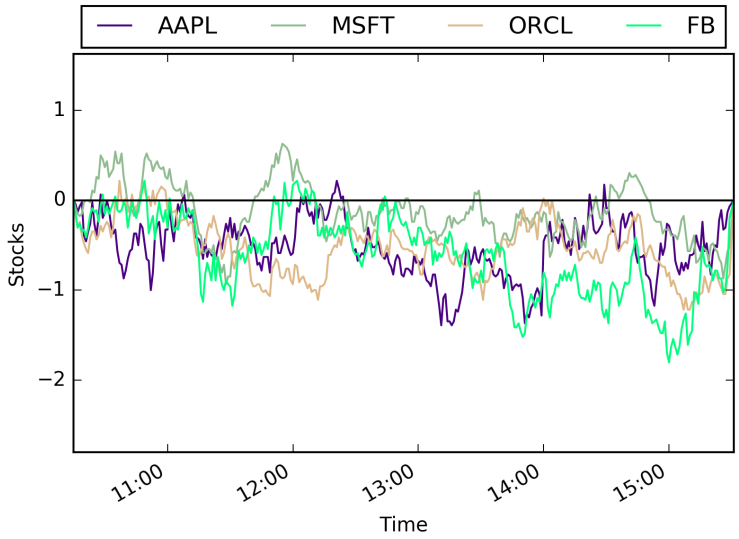
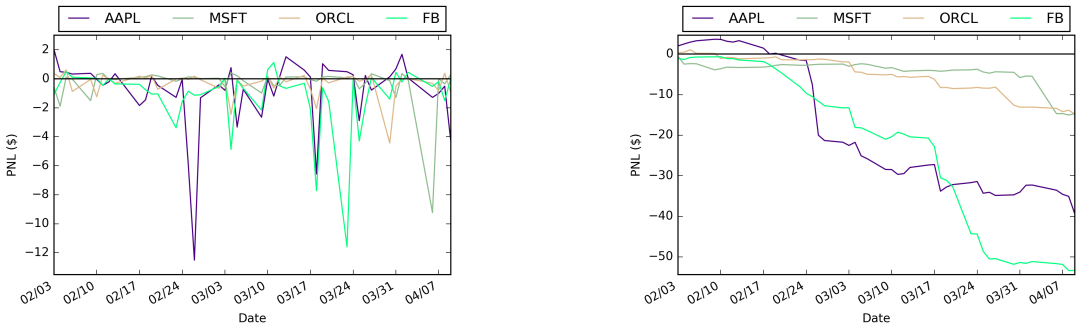


Figure 12: Average inventory obtained for 4 IT stocks with our trading strategy RF.

As shown on figure 12, the inventory turns out not to be a cause of concern in our algorithm, as it stays within one stock on either sell or buy side. This is due to our implementation of the stop loss function, which will liquidate our inventory whenever profitable. This in turn helps limit it as noise fluctuations help meet this case if there is no larger-scale trend in the market. Figure 13 shows the PNL obtained with our strategy.



(a) Daily PNL.

(b) Cumulative PNL.

Figure 13: PNL for 4 IT stocks obtained with our trading strategy using RF.

We can observe our daily PNL is globally average, except for some days when it tanks and tanks our cumulative PNL. Our algorithm seems to be able to capture small changes in trend but can't render the larger-scale trends due to real world events. This would be prevented by implementing the other part of the algorithm described in [LDZ+14], which is the signal generation from real world financial news. In the litterature, that is the part which is supposed to prevent really big losses in case of unexpected event. However, we weren't able to implement this idea since Thesys lacks the option to import it.

5 Discussion

Though our strategy did not generate results as promising as we hoped for, it still provided a novel machine learning strategy to mine signals and make predictions based on the order book. In reality, since we are market makers, we would receive compensation for providing liquidity in the market – our model does not factor in that compensation.

In our future work, we would like to gather trading statistics on the number of trades we execute per day over a long investment horizon (about 1 year). This would allow us to calculate our potential expected compensation for providing liquidity in the market. Calculating these statistics could potentially make our strategy more profitable on certain stocks.

We encountered myriad problems with the Thesys platform, which limited the assets we could trade. While we tried to trade large and midcap stocks from almost every major sector, our memory limits in Thesys (1GB) prevented us from simulating our strategies on all of these assets. Another major problem we faced with Thesys dealt with the constant timeouts we would get from their client website. While running code or backtesting our strategies, we would continually encounter 502 Bad Gateway Errors or 504 Gateway Timeout Errors, which would prevent us from accessing our strategies and backtests on the trading platform. These errors would often lock us out of the system for multiple hours at a time, derailing our productivity.

For examination of our codebase, feel free to visit our github.

References

[AS07] MARCO AVELLANEDA and SASHA STOIKOV. High-frequency trading in a limit order book. *Quantitative Finance*, 8(3):217–224, 2007.

[ELdPO10] David Easley, Marcos Lopez de Prado, and Maureen O'Hara. The microstructure of the 'flash crash': Flow toxicity, liquidity crashes and the probability of informed trading. *The Journal of Portfolio Management*, 37(2):118–128, 2010.

[LDZ+14] Xiaodong Li, Xiaotie Deng, Shanfeng Zhu, Feng Wang, and Haoran Xie. An intelligent market making strategy in algorithmic trading. *Frontiers of Computer Science*, 8(4):596–608, 2014.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.