ORF 474 HFT Final Report

Shreyas Joshi
sjoshi@princeton.edu

Abstract

High Frequency Trading commands the modern markets on daily basis. In this project we, in 48 hours, explore the market order-based strategies to meet a VWAP benchmark given a fixed participation rate. We find that machine learning can improve over baseline naive strategies in meeting this benchmark, with room for even greater improvement.

1 Introduction

For our final project, we explore machine learning methods for market-order timing for buying & selling shares at the high frequency level. We wish to maximize the following objective: min($VWAP^u - VWAP^m$, 0) when selling with the constraint that we have a 5% overall participation rate (i.e. if we are selling, then our sells must be 10% of the total sell volume that day). Our objective will be to minimize the objective max($VWAP^u - VWAP^m$, 0) when buying.

The core focus of our report will be feature engineering & machine learning: we will be using machine learning to predict ask (bid) movements 15 trades ahead given the past 15 trades that have occurred when we are buying (selling). We will attempt to create predictive features which will allow us to build robust predictive algorithms. Note that have chosen to look only 15 trades behind so as to control the feature size, and that we will be operating on the **trade clock**.

Once we have our features, we perform feature selection to reduce the dimensionality of the training set and make our data more parsimonious. Finally, we tune algorithms using 8-fold cross validation, and then pick the best one based on performance on a hold-out test set.

We will then build our execution algorithm for which we will try using a naive strategy, a book-pressure strategy, and a machine learning strategy that incorporates bid/ask move predictions. If we are buying, we will be placing market orders at the ask, and if we are selling, we will be placing market orders at the bid.

We are given 42 days of AMZN TAQ data, but we end up using approximately 30 due to the imbalance of buy and sell days. We split our data the following way, consecutively allocating buy and sell days:

- Machine Learning Buy Training / Cross Validation Set: Days 1, 2, and 4
- Machine Learning Buy Testing Set: Days 9, 11, 12, 13, and 20
- Execution Algorithm Buy Training: Days 21, 24, 31, and 35
- Execution Algorithm Buy Testing: Days 39, 40, and 42
- Machine Learning Sell Training / Cross Validation Set: Days 3, 5, and 6
- Machine Learning Sell Testing Set: 7, 8, 10, 14, and 15
- Execution Algorithm Sell Training: Days 16, 17, 18, and 19
- Execution Algorithm Sell Testing: Days 22, 23, and 25

2 Data Preparation

We used hw1 to prepare 42 days of Amazon Stock Data, and use the trade data frame.

We will use the first 3 buy days (i.e. days where we will need to buy) to build our buy machine learning algorithm (to use on days where we will be buying) from each of those days' trade data frames, which we will call $\mathbf{B_M L}^1$, $\mathbf{B_M L}^2$, and $\mathbf{B_M L}^3$. Post careful feature engineering, we will combine these datasets into the buy training set $\mathbf{TR}_{\mathbf{B_{ML}}}$. Likewise, we will call the first 3 sell data frames $\mathbf{S_M L}^1$, $\mathbf{S_M L}^2$, and $\mathbf{S_M L}^3$ and combine them (in a special way, as explained in the feature engineering section) to construct the sell training set $\mathbf{TR}_{\mathbf{S_{ML}}}$. We will then perform feature selection & cross validation of our buy machine learning algorithms on $\mathbf{TR}_{\mathbf{B_{ML}}}$, and then test on held out data $\mathbf{TS}_{\mathbf{B_{ML}}}$. We will do a similar step with the sell mahine learning algorithms (using $\mathbf{TR}_{\mathbf{S_{ML}}}$ and $\mathbf{TS}_{\mathbf{S_{ML}}}$).

2.1 Innovation 1: Feature Engineering & Feature Selection

To begin, we start with the trade data output from homework one. We will first explain how the feature space is constructed.

We cannot feed the trade data from homework one directly to a machine learning algorithm and perform cross validation in the traditional sense, because our data is a time series. Rollover cross validation, in which we train on *non-overlapping* samples from times $x-3, x-2, x-1$ and evaluate against x was considered, but was dismissed since it is difficult to code pragmatically.

Instead, we choose to map the data to a new feature space, where each feature vector would encodes local temporal historical data into the feature vector itself - the process is best explained visually. Suppose we wanted to use information from the past 2 trades to predict which direction the ask will move 2 trades later. Then, **Figure 1** captures the process of creating one training sample (feature vector). Please read this figure carefully.

As detailed in the caption of Figure 1, we run this process for each of $\mathbf{B_M L}^i$, $\mathbf{S_M L}^i$ for $i = 1, 2, 3$. We row bind each feature vector generated for all rows in $\mathbf{B_M L}^i$ for $i = 1, 2, 3$ to form $\mathbf{TR}_{\mathbf{B_{ML}}}$. We do the same with $\mathbf{S_M L}^i$ to form $\mathbf{TR}_{\mathbf{S_{ML}}}$ (except we predict bid).

We will now discuss the sequential and holistic features we create. We construct the following **sequential features**:

- Deltas (i.e. $Something_{t_n}$ - $Something_{t_1}$, where t_i refers to first row of Mini Data Frame (see Figure 1)).
  - Price
  - Bid
  - Ask
- Price - Bid
- Price - Ask
- AskSize - BidSize
- Spread
- Lots Traded
- Immediate Changes (i.e. $Something_{t_n} - Something_{t_{n-1}}$, where t_1 refers to first row of "Mini" Data Frame (see Figure 1)).
  - Price
  - Bid
  - Ask
- Crude Price Impact = $Immediate\ Change_{Price} \cdot Lots\ Traded \cdot Sign$ (Sign is a feature from the original trade data frame that was kept)

We believe that *Deltas* might be useful for providing recent price shifts and capturing a longer-term change, whereas *Immediate Changes* captures recent changes. The hope is that the machine learning

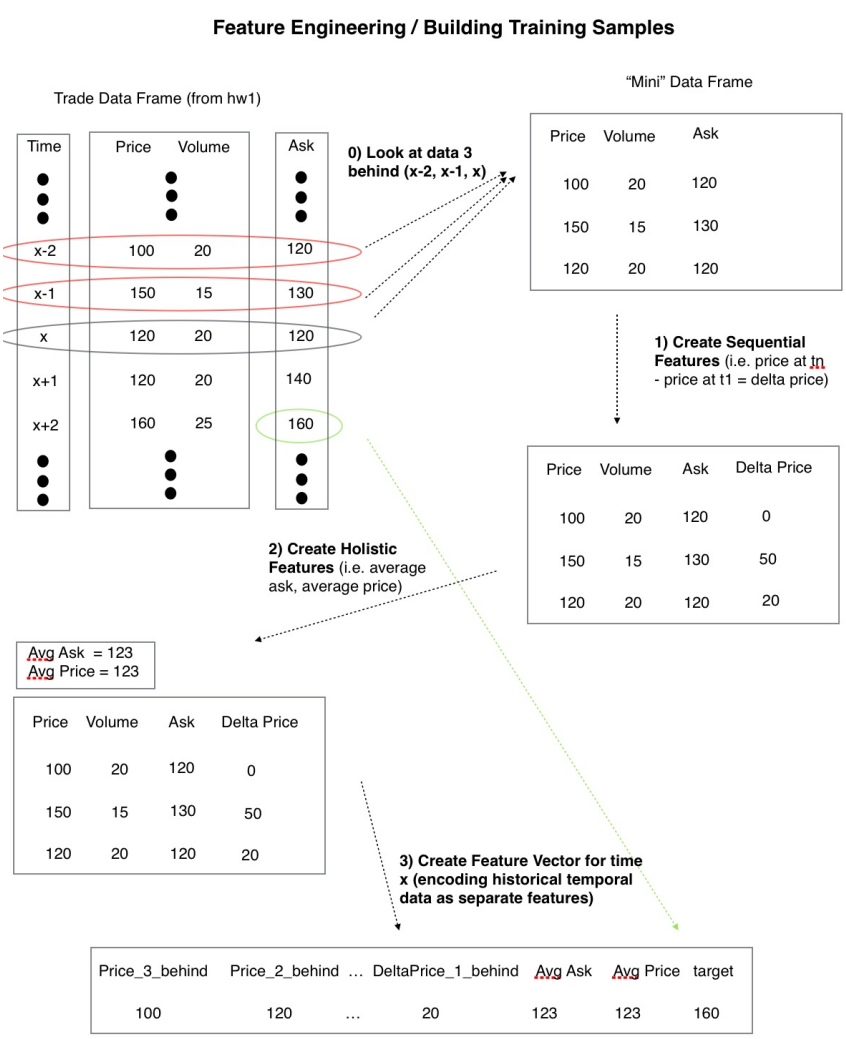

Figure 1: *Visual Representation of the feature mapping process if we were to use data from the two previous trades. We will repeat this process on on approximately each row in the trade data frames* $\mathbf{B_M L}^1$, $\mathbf{B_M L}^2$, *and* $\mathbf{B_M L}^3$, *except we look behind 15 trades and aim to predict the ask (bid) if we are buying (selling). Since the temporal data is being encoded into the sample/feature vector itself, we will now be able to perform traditional cross-validation on a set of feature vectors; no leakage can occur, since the features were built using only information from the "Mini" data frame.* ***Sequential*** *features represent features evolving over time, whereas* ***holistic*** *features capture some property in just one number.*

algorithm, given these features would be able to project a momentum representation, which could potentially be informative for predicting future bid/ask moves. The *Deltas* **replace** the actual values of Prices, Bids, and Asks; a value of 810 (Price) is less informative to the the machine learning algorithm than the change in price.

We use Price - Bid and Price - Ask in hopes of capturing directional changes of the bid/ask legs. For example, if the Price - Ask is negative, then the feature could be suggesting that many trades are happening below the ask, and that the future ask could be lower, if the price moves down. Congruent logic applies to Price - Bid.

AskSize - BidSize captures the balance of the order book. It tends to be zero most of the time, but an imbalance could potentially be a very strong signal; it can show whether there is heavy buying/selling on one side or the other.

Note that the original feature Shares Traded was also kept, but reshaped to Lots Traded (i.e. we use $Shares/100$). We do this so that a neural network we train later will not have large numbers as input (neural networks can converge faster with inputs on roughly the same scale), however scaling & and centering could also be used. Finally, the sequential features Sign, BidSize, and AskSize from the original training set were also kept.

Next, we also have our **holistic features**:

- Standard Deviation
  - Price
  - Bid
  - Ask
  - Sign
  - Lots Traded
- # of trades that occurred at a price $>=$ Ask
- # of trades that occurred at a price $<=$ Bid
- # of trades that occurred at a price Bid $<$ price $<$ Ask

The standard deviations of each feature capture recent volatility - higher standard deviations could suggest that a move in the bid/ask is upcoming. Each of the "# of trades ..." feature captures the environment around and near the bid-ask spread. If, suppose, the # of trades that occurred at a price $>=$ Ask is high, then it is quite possible that the # of trades that occur at a price $>=$ Ask will continue to be high (likewise with $<=$ Bid and Bid $<$ price $<$ Ask).

Upon completion of the feature engineering process, we are left with **233 predictors**; large, because we have 15 * (15 sequential features + original sequential features total) + (8 holistic features).

Quite a few of these features might not be very predictive of the ask/bid moves, so we use feature selection to reduce our feature space. Furthermore, a reduced feature space has the advantage in that it decreases time complexities of machine learning algorithms, and is a more parsimonious representation of the data.

To perform feature selection, we used R's Bagged Regression Trees in conjunction with the the recursive feature elimination function rfe(). rfe() begins with all the features, then reduces features in user-specified increments in greedy fashion by looking at predictor importance, and then reports the 4-fold cross validated Root Mean Square Error of bagged tree algorithm. We first ran the elimination in increments of 5 predictors over all the predictions; we noticed that the optimal number of predictors chosen was less than 50 on both the buy and sell data sets. Then, to get better resolution, we ran feature elimination with an increment of 1 predictor, with 50 being the maximum amount of predictors allowed. The results of this higher-resolution run are shown in Figure 2.

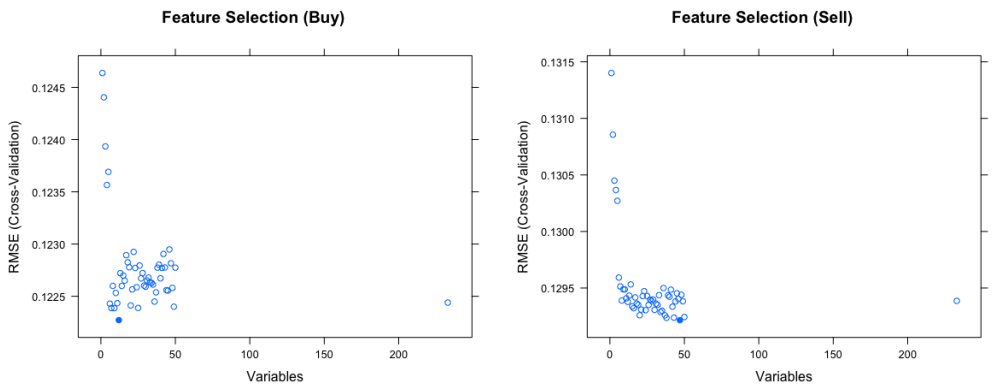

Figure 2: *Plots depicting the higher-resolution feature selection runs on the buy and sell training sets* $\mathbf{TR_{B_{ML}}}$ *and* $\mathbf{TR_{S_{ML}}}$*, respectively.*

Top 6 Selected Features on the buy training set $\mathbf{TR_{B_{ML}}}$:

1. PriceAskDiff1
2. PriceAskDiff2
3. PriceBidDiff1
4. PriceAskDiff3
5. PriceBidDiff2
6. PriceBidDiff15

PriceAskDiff$n$ refers to the nth most recent value of Price - Ask. Note that the the feature selection algorithm discovered that the 3 most recent values of Price - Ask and the two most recent values of Price - Bid were very important. This not surprising, since intuitively, the most recent events are likely to have the most predictive power. If the value of Price - Ask or Price - Bid starts to change, it could mean that a move on the ask/bid is coming. We were however surprised to see to see that the value of Price - Bid from 15 trades ago was an important feature. We speculate that this may be because the treebag algorithm attempted to capture a long run moving average representation of sorts so that it could have a comparison point to determine whether more recent price shifts (i.e. PriceAskDiff1) were significant.

Top 6 Selected Features on the sell training set $\mathbf{TR_{B_{ML}}}$:

1. PriceBidDiff1
2. PriceAskDiff1
3. PriceBidDiff2
4. PriceAskDiff2
5. PriceBidDiff3
6. PriceBidDiff15

The selection of the features on the sell training set parallels those chosen on the buy training set, except with the 3 most recent values of Price - Bid and the two most recent values of Price - Ask being very important. Once again, however, the Price - Bid differential from 15 trades ago was important as well.

3 Innovation 2: Various Machine Learning Methods

Having selected our features, we now try multiple machine learning methods on this data set. We select the following algorithms, because they are **diverse** & **relatively fast:**

- GBM (Stochastic Gradient Boosting)
- glmnet
- knn (K Nearest Neighbhors)
- mlp (Multi layer Perceptron)

GBM is a tree-based regression algorithm which uses boosting to produce a final results; it is usually a well performing, general purpose regression algorithm. glmnet is similar to a generalized linear model, but uses a mixture penalty between ridge and lasso regression. K Nearest Neighbhors looks at the neighborhood of points closest to the sample we wish to make a prediction for, and then averages these outcomes. Finally, the multi-layer perceptron is a neural network with one hidden layer.

We use grid search on $\mathbf{TR_{B_{ML}}}$ and $\mathbf{TR_{S_{ML}}}$ with 8-fold cross validation to tune each algorithm's parameter to minimize rmse. We tune each algorithm on the buy & sell sets. Figure 3 shows a sample tuning procedure for gbm on the buy training set.

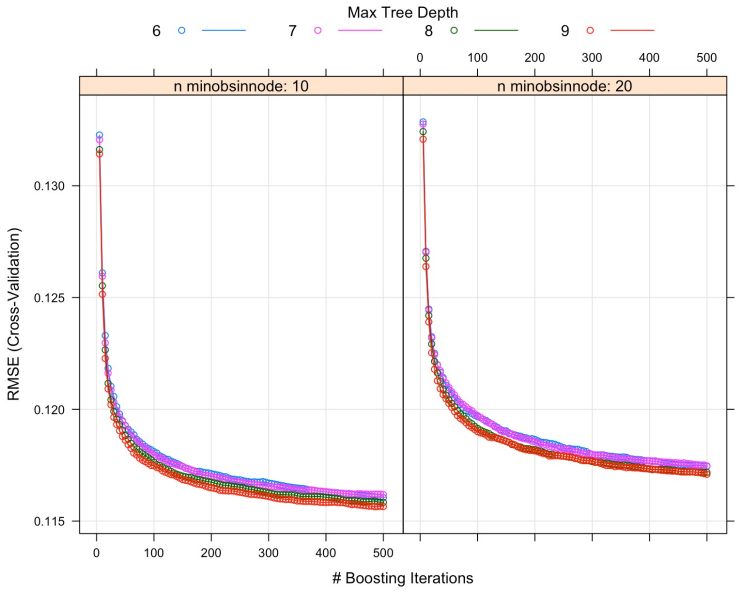

Figure 3: *Tuning GBM's four parameters tree depth, number of trees, shrinkage, and minobsinnode w/ 8-fold cross validation. As the number of trees/iterations increases, the cross validated rmse begins to fall.*

Table 1 shows a table of the cross validation results after tuning for each algorithm. Note that GBM appears to outperform the other algorithms. Since we used the cross validation folds to tune the parameters, however, it would be innapropriate to use them to choose our algorithm; we would end up over-fitting to our training set.

Thus, we evaluate our classifiers performance on untouched testing data $\mathbf{TS}_{\mathbf{B_{ML}}}$ and $\mathbf{TS}_{\mathbf{S_{ML}}}$, and get the results shown in Table 2.

Table 1: Cross Validation Performance

Algorithm	GBM	glmnet	knn	mlp
CV RMSE on Buy Days	**0.1156453**	0.1269655	0.1219115	0.1230227
CV RMSE on Sell Days	**0.1192709**	0.1298087	0.1328139	0.1344122

Table 2: Out of Sample Test Performance

Algorithm	GBM	glmnet	knn	mlp
Mean RMSE on Buy Days	0.1359031	**0.1274332**	0.1322206	0.1303746
Mean RMSE on Sell Days	0.1313279	**0.128946**	0.1404795	0.1344122

Much to our surprise, GBM performs much worse on the test set, even more so than the other algorithms; we suspect that this may be because GBM over-fitted during the cross validation process. On the other hand, glmnet performed quite well - its testing results nearly matched its cross validation performance. It may be that because glmnet is a linear predictive model, it is able to generalize better than its peers, and hence avoid over-fitting. Thus, we select glmnet to be our final model for predicting both bid & ask price moves.

Finally, we would also like to point out that glmnet was the fastest algorithm, both in training and making predictions. This is important to note, because in the live trading environment, prediction speed is a limiting factor of execution speed. We cannot simulate this easily in our back-testing framework, but it is something to bear in mind.

We are now ready to deploy our machine learning model into the execution algorithm.

4 Execution & Trading Engine

4.1 Constructing the execution framework

Our execution algorithm will simulate trading and measure how well we perform on our objective functions (as described in the introduction section). Because we are performing market orders, we can following our objectives trade-by-trade, and report our performance at the end. We will be using 5% PoV rate, meaning that if we are buying, we must buy 10% of the buy volume, in shares, that occured that day.

Our trading engine will work as follows:

1. Suppose we are buying (selling). At each timestep x, we make a decision as to whether or not we would like to trade at the ask (bid). We will discuss a naive decision function, a book pressure decision function, and a machine learning decision function in the upcoming sub sections.
2. If we wish to trade, then we place a market order immediately, and then assume that this market order will execute the next time a trade takes place (i.e. trade time x + 1). In reality, there would likely be a delay, but for the sake of keeping our model simple, we will assume that our order gets executed the next time a trade occurs.
3. Note that we somehow need to penalize our simulation for the fact that we cannot incorporate price impact into our model. We will do this by setting an under-participation limit to prevent us from having a situation arise where we have to trade a large amounts of shares at once; this will also help us stay on track with the PoV Rate in close to real time. For example, suppose we are buying. Then if $.1 * Total\ Buy\ Volume\ Observed$ - $Our\ Pov\ Rate > threshold_{up}$, where $threshold_{up}$ is a parameter, we will force ourselves to buy n_b shares, where n_b is limited to 500 shares. We choose this value of n_b having looked at distributions of shares traded on our stock (most of the trades were of size 5 lots or less). We place a similar limit when selling.
4. Similarly, we will also place a limit on over-participation (to avoid a scenario where we end up having to trade a large amount of shares at the end of the day). Put simply, we will not allow trades to occur if $Our\ Pov\ Rate$ - $.1 * Total\ Buy\ Volume\ Observed > threshold_{op}$, where $threshold_{op}$ is a parameter (for buys). We place a similar limit when selling.
5. At the end of the day, we will sell off any additional we may have had by over participating or buy shares as needed if we were under-participating. Note that these quantities will be limited by our choice of $threshold_{up}$ and $threshold_{op}$.

As outlined in the introduction section, we will tune parameters needed for the decision functions/strategies using the *execution training* set of days and then evaluate our performance on held out *execution testing* set of days. Note neither the *execution training* nor the *execution testing* were used at all in evaluating/tuning the machine learning algorithms; these days are completely different for both the buy and sell. This way, we ensure that we are preventing information leakage.

Finally, we will fix $threshold_{up}$ and $threshold_{op}$ to be 10 lots (1000 shares) each. These thresholds were chosen due to our model's inability to incorporate price impact (i.e. this is because at the end of the day we have to buy/sell shares as needed to meet our participation rate; hence, we assume that we can only buy/sell 1000 shares that one time at the end of the day).

We will now explain the three strategies we chose to try in the following subsections.

4.2 Naive Strategy

Buying: Our naive strategy in the case that we are buying will be to follow trades that occur at the ask or higher (i.e. if we see a trade happen at a price $>=$ ask, then we place a a market order to buy at the ask). Our order size will be fixed to one lot.

Selling: Our naive strategy in the case that we are selling will be to follow trades that occur at the bid or lower (i.e. if we see a trade happen at a price $<=$ bid, then we place a a market order to sell at the bid). Our order size will be fixed to one lot.

Tuning: N/A

4.3 Book Pressure Strategy

For our book pressure strategy, we calculate book pressure as $BP = (BidSize - AskSize)$ / $(BidSize + AskSize)$.

Buying: Our book pressure strategy in the case that we are buying will be to execute a trade on the ask if $BP >= threshBP_{Buy}$, where $threshBP_{Buy}$ will be a parameter we can tune. We will let the trade size ($TradeSize$) be a tunable parameter as well.

Selling: Our book pressure strategy in the case that we are selling will be to execute a trade on the bid if $BP <= threshBP_{Sell}$, where $threshBP_{Sell}$ will be a parameter we can tune. We will let $TradeSize$ be a tunable parameter as well.

Tuning: For buys, tune over $threshBP_{Buy} \in \{-0.5, -0.3, -0.1, 0.1, 0.3, 0.5\}$ and $TradeSize \in \{200, 300, 400, 500\}$.

For sells, tune over $threshBP_{Sell} \in \{-0.5, -0.3, -0.1, 0.1, 0.3, 0.5\}$ and $TradeSize \in \{200, 300, 400, 500\}$.

4.4 Machine Learning Strategy

We use our trained glmnet buy algorithm from the machine learning section to predict the ask move on the buy days, and the glmnet sell algorithm to predict the bid move on the sell days. Note that we will not be able to make predictions for first 14 trades that occur (since the the algorithm uses the 15 most recent trades for prediction).

Buying: Our machine learning strategy in the case that we are buying will be to execute a trade on the ask if $PredictedAskMove >= threshML_{Buy}$, where $threshML_{Buy}$ will be a parameter we can tune. We will let the trade size be a tunable parameter as well.

Selling: Our machine learning strategy in the case that we are selling will be to execute a trade on the bid if $PredictedBidMove <= threshML_{Sell}$, where $threshML_{Sell}$ will be a parameter we can tune. We will let the trade size be a tunable parameter as well.

Tuning: For buys, tune over $threshML_{Buy} \in \{0.00, 0.04, 0.08, 0.12, 0.16, 0.20\}$ and $TradeSize \in \{200, 300, 400, 500\}$.

For sells, tune over $threshML_{Sell} \in \{0.00, -0.04, -0.08, -0.12, -0.16, -0.20\}$ and $TradeSize \in \{200, 300, 400, 500\}$.

5 Results of the execution

The results on the training/tuning set are shown in Table 3.

Table 3: Training/Tuning Results of the Various Strategies. The mean objective for the buys is the average of the objective over buy training days 21, 24, 31, and 35. The mean objective for the buys is the average of the objective over sell training days 16, 17, 18, and 19.

Strategy	Naive	BP	ML
Buy Mean Objective (lower is better)	0.05481149	0.06432751	**0.03692328**
Sell Mean Objective (higher is better)	-0.1596762	-0.1489058	**-0.1367947**

BP Buy Optimal Params: $threshBP_{Buy}$ = -0.5, $TradeSize$ = 400.

BP Sell Optimal Params: $threshBP_{Sell}$ = -0.1, $TradeSize$ = 400.

ML Buy Optimal Params: $threshML_{Buy}$ = 0.16, $TradeSize$ = 400.

ML Sell Optimal Params: $threshML_{Sell}$ = -0.20, $TradeSize$ = 500.

We were surprised to see that the book pressure strategy performed worse than the naive strategy, but it is quite possible; we saw in homework 2 that book pressure was not always keen on predicting price moves.

On the other hand, the machine learning algorithm performed well against the naive and book pressure strategies. Figure 4 shows the value of the objective function as we tune the ML Buy strategy.

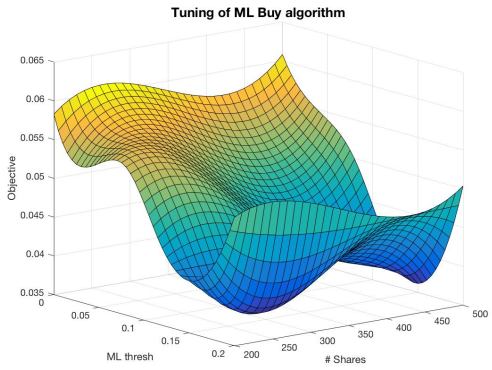

Figure 4: *Smoothed topography of tuning the ML Buy Strategy. Lower objective value is better.*

Notice how the dip at ML Thresh = 0.16. This threshold represents the balance between having the ML Algorithm participate often enough to make a difference in the objective while also being confident (higher value of ML Thresh corresponds to the ML algorithm making fewer, but more more confident predictions). The number of shares traded appears not have made as big of an impact on the objective as ML thresh.

Finally, we run these strategies on out-of-sample days. For buys, we tested on Days 39, 40, and 42 (see Table 4). For sells, we tested on Days 22, 23, and 25 (see Table 5).

Table 4: Testing performance of the buy strategies: ML trading algorithm outperforms both the naive and book pressure trading algorithms. Lower objective is better.

Strategy (Buy)	Naive	BP	ML
Day 39 Objective Value	0	0.001497115	0
Day 40 Objective Value	0.05734603	0.110501461	0.05559902
Day 42 Objective Value	0	0.008662126	0
Average Objective Value	0.01911534	0.04022023	**0.01853301**

Table 5: Testing performance of the sell strategies: ML trading algorithm outperforms both the naive and book pressure trading algorithms. Higher objective is better.

Strategy (Sell)	Naive	BP	ML
Day 22 Objective Value	-0.123298525	-0.12391691	-0.069166941
Day 23 Objective Value	-0.003866017	-0.00660113	0
Day 25 Objective Value	-0.036700354	0	-0.002399167
Average Objective Value	-0.05462163	-0.04350601	**-0.02385537**

Unsurprisingly, out results show that machine learning can help decrease the cost we incur in VWAP terms when trading at the high frequency level. Ideally, more days would be needed to ensure that the results are statistically sound, but this is quite exciting to see!

6 Drawbacks, Assumptions & Extensions

Our machine learning model, although appears to be great, needs much more rigorous testing before it can be deployed. For example, one assumption we make is that we know ahead of time whether or not a day was a buy day or a sell day; not knowing the type of day would impact our machine learning training process (since we had seperate algorithms for buy and sell days). Although this impact is likely minor, since bid and ask moves are highly correlated, it is something to consider.

Furthermore, one additional major assumption we make is regarding speed and price impact. We were able to trade up to 500 shares at once and make executions on trades that immediately follow our observed ones, when in reality, we would be slower, since we would have to compute both the prediction from glmnet and account for latency. We would also end up paying more for large trades, since we would eat up the order book. Potential extensions to our project could involve adding a time delay on the execution. Furthermore, we could potentially simulate a worst-case price impact scenario by assuming we would have to trade at a price m cents higher for every n lots in our block.

We also had only a few days we were able to use for our final testing set; ideally, we would use months of data to get a more confident estimate of our strategies.

Finally, ensemble strategies using machine learning could be explored to give even stronger predictions. It is likely that glmnet, mlp, and knn make somewhat uncorrelated predictions due to the nature of their designs. Thus, a scheme like stacked generalization could be used, with glmnet as a meta-classifier to improve predictive capabilities even further.

7 Acknowledgements

My machine would like to thank Amazon's TAQ data for not being too strenuous on its cores.

I would like to thank the TAs and Prof. Kevin Webster for helping me throughout this semester and for strengthening my knowledge of High Frequency Trading Systems.

I began the exam Wednesday, 5/11/2016 at 2:45pm, and finished Friday, 5/13/2016 at 2:26pm, using 47 hours and 41 minutes of the 48 hour time limit.