Limit order book analysis using Machine Learning

Naveen Mathew Nathan Sathiyanathan[1]
Department of Statistics, University of Illinois at Urbana-Champaign

(Dated: 9 May 2019)

Limit order data for a day usually exceeds 1 million rows after aggregation. Analyzing or visualizing across multiple days leads to large number of rows which cannot be stored in memory. The objective of this analysis is to visualize and preprocess the data at scale and to perform pattern analysis using machine learning. R Shiny application was built for dynamic visualization of order book data. Using performance enhancement methods, the time for visualization was reduced to approximately 2 minutes. The analysis was run on 200 days data and performance was tested. Finally, a machine learning models were built to predict direction of price based on image of volumes. The model outperformed majority guess.

I. INTRODUCTION

Aggregated limit order data is usually very large. Text file size is approximately 600 MB. The in-memory size of each data file exceeds 1.5 GB because of expansion of small numeric counts that are concisely represented as 1 digit in text format. Therefore, processing multiple files at a time is not possible even using multiprocessing.

For visualizing limit orders, 'obAnalytics' package was used. Trades and events data was also required for a complete picture, but this data was not available. Also, the input data format did not match the format required by the package for visualization. Thefeore, a scalable method was required to transform from the raw data format to the required format.

Finally, large scale feature extraction is not possible if all files are taken simultaneously. Also, the largest file for a particular ticker may not be the same across days. Therefore, one file was analyzed at a time and features were derived. The resulting features were concatenated and machine learning was applied to learn patterns that relate the image of volumes to the direction of movement of price.

II. LITERATURE SURVEY

Several attempts were made to apply machine learning for analyzing patterns in limit order books. Kercheval et al. (2013) applied machine learning to capture the dynamics of high frequency limit order books in financial equity markets and automate real-time prediction of metrics such as mid-price movement and price spread crossing. SVM is applied to perform multi-class classification based on vector of attributes such as price and volume at different levels. Experiments showed that the model was useful for short term price forecasts.

Dixon (2017) used RNN for sequence classification of limit order books. A short sequence of observations of limit order book depths and market orders was used to predict a next event price-flip. The paper claims that non-linear dynamics of near-term price flips was captured by RNN using spatio-temporal representation of limit order book. The resulting model compares favorably with linear Kalman filter on SP500 E-mini futures level II data over August 2016.

Nevmyvaka et al. (2006) applied reinforcement learning to optimized trade execution in financial markets. They used the model on 1.5 years of millisecond level limit order book data from NASDAQ. The learning algorithm exploits a natural low-impact factorization of the state space. Reinforcement learning showed promising results in learning market microstructures.

Ntakaris et al. (2018) set a benchmark for mid price prediction in high frequency limit orders. They extracted normalized data representations of time series data for five stocks from the Nasdaq Nordic stock market for a time period of 10 consecutive days. The resulting data set of 4 million rows were analyzed and experiments were performed using cross-validation.

Sirignano (2016) modeled the spatial distribution of limit order books using neural network architecture. It tries to model the joint distribution of the state of limit order book at a future time based on the current state of the limit order book. It was observed that deep learning outperformed logistic regression and exhibits good performance at the tail - which is important for risk management.

III. DATA DESCRIPTION

Data was purchased from proprietary source. It consists of 201 compressed (.zip) files, each of which has 10 categories of tickers, namely:

- ES (SP 500 E-Mini) - NQ (Nasdaq 100 E-Mini) - 6E (Euro FX) - 6J (JPY FX) - CL (Crude Oil WTI) - NG (Natural Gas) - ZN (10-Year T-Note) - GC (Gold) - ZC(Corn) - ZS (Soybeans)

The distribution of number of files in each ticker category is:

Ticker Category	Number of Tickers
6E	6
6J	6
CL	35
ES	5
GC	22
NG	45
NQ	5
ZC	17
ZN	2
ZS	23

Due to pausity of time, the scope is limited to analysis of 6E (Euro FX). The largest file is chosen in each day and the data is normalized to account for difference in scales of volumes. Each ticker file has the following structure:

A. Variables

- Date: Date string in the format of YYYYMMDD.
- Timestamp: Time of the day in HHMMSSsss (s = millisecond) format.
- Ticker: Name of the ticker.
- Side: BUY/SELL side of the aggregated row.
- Flags: .
- Depth: Total depth of market in given side.
- Level 1-10: 10 columns in "price x volume (orders)" format.

IV. PREPROCESSING

A. Tick Bar

A tick bar summarizes the open, close, low and high price over a fixed number of ticks. As an example, it was observed that a ticker file with 2.5 million rows has around 1.02 million distinct timestamps. We will consider only the last row for each timestamp due to lack of sub-millisecond level data. Therefore, the maximum number of ticks is 1.02 million for the chosen file.

For creating tick bars, a threshold of 5 ticks was used. This resulted in around 204,000 tick bars that were annotated with mid-price summaries.

B. CUSUM filter

CUSUM (cumulative sum control chart) is a sequential analysis technique used for detecting change. It can detect unexpected step changes in a time series. Therefore, it is ideal for filtering the time bars to come up with a series of potential signals of price change. The cumulative sum is calculated as:

$S_0 = 0; S_{t+1} = max(0, S_t + y_t - E_{t-1}(y_t))$

$S_t \geq h \implies \exists \tau \in [1,t] | \sum_{i=\tau}^{t} (y_i - E_{i-1}(y_t)) \geq h$

The idea of CUSUM is extended to symmetric CUSUM filter:

- $S_t^+ = max(0, S_{t-1}^+ + y_t - E_{t-1}(y_t)); S_0^+ = 0$
- $S_t^- = max(0, S_{t-1}^- + y_t - E_{t-1}(y_t)); S_0^- = 0$
- $S_t = max(S_t^+, S_t^-)$

A feature bar is sampled where the symmetric cumulative sum exceeds a threshold. Using the symmetric CUSUM filter, different number of feature bars are sampled per day based on the amount of variability in the price compared to the expected distribution of price change.

C. Triple Barrier Labeling

CUSUM filter was used for sampling relevant ticks from the whole data set. Triple barrier method was used for labeling features that were sampled. Triple barrier method uses three barriers: top, bottom and right. The right barrier is characterized by a pre-defined number of ticks from the sampled feature bar. The top and bottom barrier are characterized by return threshold. The label is based on the first bar that is touched by the series. The labeling is done as follows:

$$label = \begin{cases} -1 & \text{if bottom barrier is hit first} \\ 0 & \text{if right barrier is hit first} \\ 1 & \text{if top barrier is hit first} \end{cases} \quad (1)$$

The ‘0‘ labels were removed because it causes discrepancies in classification and the distinction between ‘-1‘ and ‘1‘ labels is expected to be clearer. This makes the decision boundary less flexible, leading to a data set that can fit appropriately.

D. Feature Scaling

Volume features for different days will be of different scales. The hypothesis is that the relative amount of trading within a *day* will affect the movement of mid-price. Therefore, the individual volume columns were 1) scaled to have mean = 0 and variance = 1 within each column, 2) normalized to have minimum 0 and maximum 1 across all 10 columns. The difference in preprocessing is expected to produce different results in the convolution - therefore the preprocessing step acts like a hyperparameter.

V. EXPLORATORY ANALYSIS

A. Visualizing Prices and Volumes

The data set is structured to have one row per side. The row contains the summary of the side - with highest to lowest price from level 1 to level 10 for buy side and lowest to highest price from level 1 to level 10 for sell side.

The required format for visualization using ‘obAnalytics‘ package has 1 row per side-price level combination. This leads to approximately 20.4 million rows for visualization. This number exceeds the possibile visualization limit in R using the package.

It is important to note that creating repeated rows for the same price-volume combination leads to overlapping colors

on the plot. Therefore, it is beneficial to have only 1 row of price-volume combination across multiple times if the volume remains unchanged for a time period. This led to the following visualization:

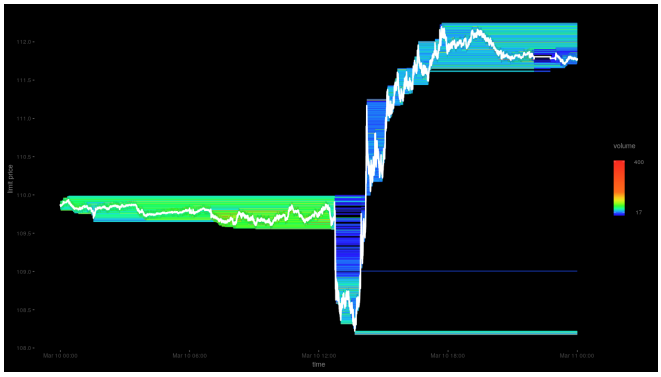

FIG. 1. Mid price-time plot colored by volume.

We note that the figure is inaccurate when higher depths are censored. This occurs when the depth of market exceeds 10 and the volumes in those price levels exceeds reduces to 0. The color scale of the image can be modified based on transformed values of the volumes at different levels.

B. Tick Bar - Example

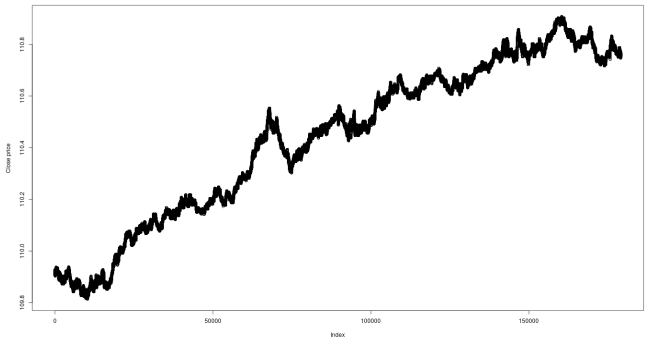

FIG. 2. Tick Bar

Note: This example shows sampled ticks, but it is not for the same ticker as the price-volume plot shown above.

C. CUSUM Filter - Example

Note: This example shows CUSUM filter on the sampled ticks shown above, but it is not for the same ticker as the price-volume plot shown above.

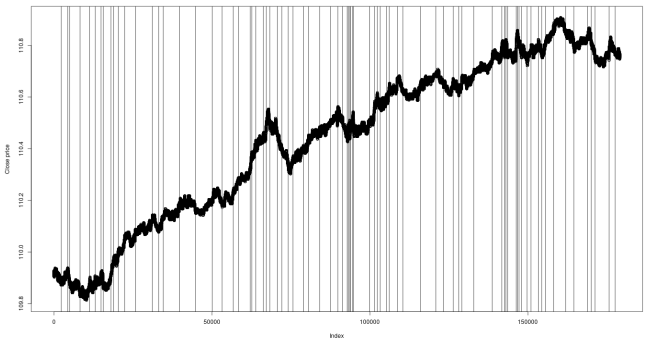

FIG. 3. CUSUM Filter

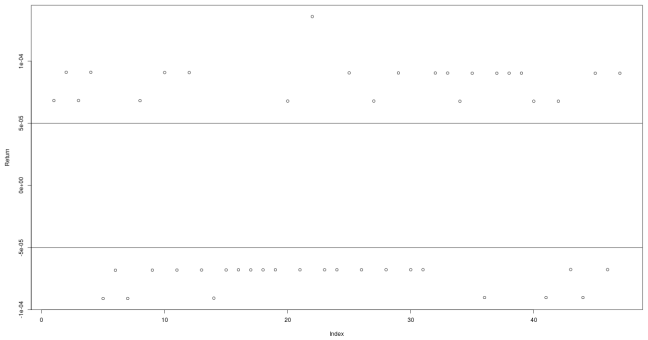

FIG. 4. Triple Barrier Labeling - After Subsetting

D. Triple Barrier Labeling

Note: This example shows CUSUM filter on the sampled ticks shown above, but it is not for the same ticker as the price-volume plot shown above.

E. Visualizing Unscaled Features

Features were created by considering the volumes at different levels for 3 seconds before the sampled feature bar. These features are useful in predicting the direction of movement of the mid price.

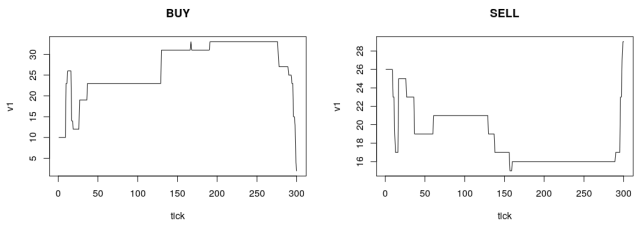

FIG. 5. Buy-Sell feature example 1

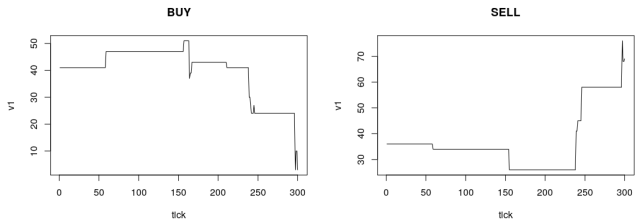

FIG. 6. Buy-Sell feature example 2

VI. MODELING

Candlestick diagram is very useful in understanding the movement of prices in limit order book. This inspired the idea of using image understanding for relating patterns in the image to the movement in price. The visualization by 'obAnalytics' package gives an idea about the importance of scales of volumes in different levels to the movement of mid price.

A. Convolutional Neural Network

Convolutional neural network is generally used for image analysis. Convolution operation creates feature maps that extract useful information from the image. An optional pooling layer samples from the output of the convolution layer. The output of one or more convolutions is fed to a dense layer for classification.

Convolutional neural network can be used to understand spatio-temporal patterns in volume in limit orders. The raw volumes can be used instead of using the image plotted from the raw data as it will have more predictive power. An image of last 3000 milliseconds is created for each feature bar which is labelled either -1 or 1. This was also downsampled to 300 observations spaced at 10 milliseconds to understand the granularity of time required for prediction.

B. Hyperparameters

There are several tunable hyperparameters in the model. The set of hyperparameters extends to the preprocessing steps and is not limited to the model. There is a tradeoff between the following hyperparameters that were used in preprocessing: 1) number of ticks between the feature bar and right barrier in triple barrier method, 2) number of labels created per day, 3) return threshold for labeling as -1 or 1. Ideally, we require sufficientnumber of labels per day to build a good model for prediction - therefore, the right barrier should be closer to the feature bar. However, if the number of ticks is too low, then there is no scope for change in mid-price before the right barrier. Finding the right setting is difficult because each iteration of preprocessing consumes more than 3.5 day (84 hours). Therefore, the following values were chosen for the hyperparameters used in preprocessing:

- nTic (tick bar): 5.
- h: 0.2 * daily standard deviation of mid price (dynamically calculated).
- right barrier: 50, 5000 ticks.
- trgt: 0.002 (symmetric), dynamic based on minimum price change for each ticker.

C. Logistic Regression

The chosen hyperparameters led to average forecast period of 150 seconds with features lasting for a duration of 3 seconds. This is unrealistic in financial markets. Therefore, the data set was downsampled to test the hypothesis. The downsampling was done to reduce the forecast period to 60 seconds - features with T_{up} or T_{lo} that correspond to 1 minute on average were used. The resulting data set has 2930 rows. A train-test ratio of 3:1 was used for splitting the data. Logistic regression model was built with volumes from last 1 second as features - $n_{features} = 100 \times 10 \times 2 = 2000$. The model led to few probabilities of 0 or 1. This was a clear case of overfit.

D. Forward Selection

Logistic regression is likely to overfit because addition of any new independent variable with finite VIF will lead to strictly lower residual deviance. In cases where p > n, statistical significance usually leads to underfitting for prediction. AIC adjusts the residual deviance for residual degrees of freedom. It usually leads to a reduced feature set with good bias-variance tradeoff. However, step-wise forward selection using AIC did not converge in 84 hours. Therefore, a reduced feature set of 227 variables was used for modeling.

E. Elasticnet Regularization

It was observed that stepwise model did not produce a good bias-variance tradeoff because of the low signal-noise ratio. Therefore, regularization was used to improve the bias-variance characteristic of the model. Simple L1 or L2 regularization usually leads to under and over fits respectively. Therefore, elasticnet (linear combination of L1 and L2) regularization was used on logistic regression.

$Penalty = \lambda \times (\alpha \times ||\beta||_1 + \frac{1-\alpha}{2} \times ||\beta||_2^2)$

Hyperparameters λ and α were tuned using 3-fold cross-validation.

F. Training Scheme

Training deep learning algorithms such as convolutional neural networks involves several heuristics. Most of the heuristics are experimental and may go wrong consistently based on the loss curve for the data. Therefore, it is essential

to use experimental checks and corrective measures on few neural network hyperparameters. The commonly used methods are:

- Adam optimizer (gradient + momentum)
- Early stopping (stop if validation accuracy gets worse in 3 iterations)
- Learning rate decay (decrease learning rate by a factor of 0.1 on loss curve plateau)
- Save best iteration (based on validation accuracy)

VII. RESULTS AND CONCLUSIONS

A. Traditional Machine Learning

It was noted that the traditional machine learning algorithms outperformed deep learning at high thresholds and very low sample sizes (7000 examples). This is because of overfitting. However, deep learning algorithms clearly outperformed traditional machine learning algoriths at larger sample sizes even though the signal to noise ratio was lower than those used in cases where the sample size was small. Also, the training time for forward selection using AIC was too high. The forward selection was stopped after 24 hours and the resulting variables were chosen for analysis.

B. Deep Learning

1. *Vulnerability - Initialization*

The train-validation-test splitting was made repeatable by setting a seed. However, it was observed that the process of learning was not repeatable because of lack of seed in Keras GPU based training APIs. It was also observed that the convergence and estimates were affected by the initialization of the parameters in the learning algorithm. This occurred despite the use of training heuristics, use of momentum and Adam optimization, regularization, etc. due to the low signal to noise ratio. Therefore, the model was run at least 3 times for each combination of hyperparameters and the best results were noted. It should be noted that the best iteration of each run (based on validation accuracy) is used to identify the best set of hyperparameters.

2. *CNN - Learning Curve*

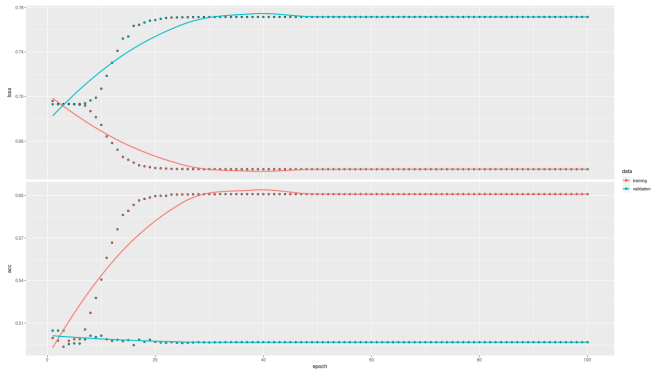

FIG. 7. Learning

3. *Depth-wise Separable CNN - Learning Curve*

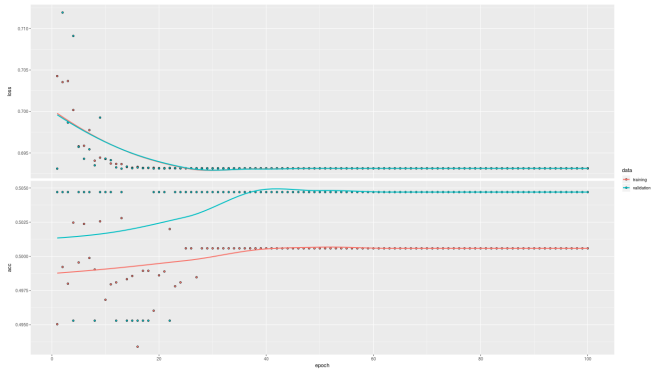

FIG. 8. Learning

C. Models - Summary

Case	Preprocessing	Modeling	Train accuracy | NIR | p-value	Test accuracy (p-value)
1	h = 0.1, trgt = 0.0001, t_{right} = 50	Logistic regression	0.517 | 0.517 | 0.500	0.529 | 0.529 | 0.500
2	h = 0.1, trgt = 0.0001, t_{right} = 50	Depth-wise separable CNN input (3000, 10, 2) >> filter: 32 (3 x 3) >> ReLU >> filter: 8 (3 x 3) >> ReLU >> Dense (8 units) >> output(2 units)	0.517 | 0.517 | 0.500	0.529 | 0.529 | 0.500
3	h = 0.1, trgt = 0.0002, t_{right} = 50	Depth-wise separable CNN input (300, 10, 2) >> filter: 32 (3 x 3) >> ReLU >> filter: 8 (3 x 3) >> ReLU >> Dense (8 units) >> output(2 units)	0.517 | 0.517 | 0.500	0.529 | 0.529 | 0.500
4	h = 0.2 x stdev_{day}, trgt = 0.0002, t_{right} = 5000	Depth-wise separable CNN input (300, 10, 2) >> filter: 2 (3 x 3) >> ReLU >> filter: 1 (3 x 3) >> ReLU >> Dense (8 units) >> output(2 units)	0.517 | 0.517 | 0.500	0.529 | 0.529 | 0.500
5	Subset of case 4 with max(t_{up}, t_{lo}) ≤ 2000	Regularized logistic regression Cross-validated alpha = 0.296	0.545 | 0.519 | 0.009)	0.537 | 0.519 | 0.109
6	h = 0.2 x stdev_{day}, trgt = dynamic, t_{right} = 50	Depth-wise separable CNN input (100, 10, 2) >> filter: 2 (3 x 3) >> ReLU >> filter: 1 (3 x 3) >> ReLU >> Dense (8 units) >> output(2 units)	0.5026 | 0.5014 | 0.341	0.5014 | 0.5014 | 0.5
7	h = 0.2 x stdev_{day}, trgt = dynamic, t_{right} = 50	Case 6 without depth-wise separation	0.5026 | 0.5014 | 0.341	0.5014 | 0.5014 | 0.5
8	h = 0.2 x stdev_{day}, trgt = dynamic, t_{right} = 50	Case 7 with rearranged volume levels	0.5275 | 0.5014 | 0	0.5071 | 0.5014 | 0.065

Note: NIR = No Information Rate (accuracy of majority guess). nTic = 5 for each case

FIG. 9. Results

D. Confusion Matrix

Based on the full summary provided in the previous section, CNN with levels arranged in order of prices was chosen for prediction. The ordering of columns was: $sell_{level10}, sell_{level9}, ..., sell_{level1}, buy_{level1}, buy_{level2}, ..., buy_{level10}$

		Reference	
		0	1
Prediction	0	1981	593
	1	24549	26087

FIG. 10. Confusion Matrix - Train

		Reference	
		0	1
Prediction	0	4027	3926
	1	4816	4967

FIG. 11. Confusion Matrix - Test

E. Conclusion

Based on the results obtained above, deep learning CNN model using only the image of last 3 seconds to predict the label of next 2 seconds (approx.) was the best. It was able to beat the majority guess by a margin that was statistically significant at $\alpha = 0.1$, but not statistically significant at $\alpha = 0.05$ where traditional machine learning algorithms failed to beat the majority guess ($p - value > 0.5$). The model shows some signs of overfitting because the train set metrics are much better than test set metrics, but this could not be controlled due to the low signal to noise ratio.

However, this method of using the volumes at different levels is not exactly the same as using the image. This is because the level of the price was used as proxy for absolute position of the price in the image, but this idea does not always generalize - when the gap between level1 prices of buy and sell is high or when the volume is 0 at few levels.

VIII. SCOPE FOR IMPROVEMENT

- Use large computational facility with large RAM to enable multiprocessing to handle preprocessing step. This was a bottleneck in the project because preprocessing took 3.5 days.
- Use other tickers as covariates.
- Tune preprocessing parameters such as nTic, h, trgt along with model hyperparameters: this is very time consuming, hence it could not be explored during the duration of this project.
- Arrange the prices of different sides according to their absolute prices instead of the levels. This is expected to increase the postprocessing time.
- Increasing training data: Create anti-patterns with the inverted image as features and inverted labels as outcome.
- Label using VWAP (volume weigted average) instead of mid-price.
- Explore other labeling methods other than triple barrier method.

APPENDIX

A. Code Samples

- Visualizing price-volume patterns

```
def process_buys_row(row):
  ret = []
  row["Depth"] = int(row["Depth"])
  row["shifted_Depth"] = int(row["shifted_Depth"])
  lst = [row['Timestamp'], 'bid']
  if row["Depth"] > 0:
    i = j = 1
    if row["shifted_Depth"] > 0:
      while i <= row["Depth"]:
        del_check = False
        while row['p' + str(i)] < row['shifted_p' +
     str(j)] and j < row["shifted_Depth"]:
          if i > 1:
            if row['p' + str(i - 1)] != row['
    shifted_p' + str(j)]:
              ret.append(lst + [row['shifted_p' +
    str(j)], 0, 0])
              del_check = True
          else:
            ret.append(lst + [row['shifted_p' + str
    (j)], 0, 0])
            del_check = True
          j += 1
        if row['p'+str(i)] == row['shifted_p'+str(j
    )]:
          if del_check and i > 1:
            if row['p' + str(i - 1)] == ret[-1][2]:
              ret = ret[0:-1]
          if row['v' + str(i)] != row['shifted_v' +
     str(j)]:
            ret.append(lst + [row['p'+str(i)], row[
    'v'+str(i)], row['o'+str(i)]])
        else:
          ret.append(lst + [row['p'+str(i)], row['v
    '+str(i)], row['o'+str(i)]])
        i += 1
      if j < row["shifted_Depth"]:
        missing = row[shifted_cols[(3*j):(3*row["
    shifted_Depth"])]].values.reshape((row["
    shifted_Depth"] - j, 3)).tolist()
        if type(missing[0]) == list:
          ret = ret + [lst + x for x in missing]
        else:
          ret = ret + [lst + missing]
    else:
      ret = row[all_level_cols[0:(3*row["Depth"])
    ]].values.reshape((row["Depth"], 3)).tolist()
```

```
      if type(ret[0]) == list:
        ret = [lst + x for x in ret]
      else:
        ret = [lst + ret]
  ret = np.array(ret)
  if ret.shape[0] == 0:
    ret = np.array([lst + [0.0, 0.0, 0.0]])
  return ret


def process_buys_df(df):
    return df.apply(process_buys_row, axis = 1)
```

```
splits = mp.cpu_count() - 1
p = mp.Pool(processes = splits)
split_dfs = np.array_split(buys, splits)
pool_results = p.map(process_buys_df, split_dfs)
p.close()
p.join()
bys = pd.concat(pool_results, axis = 0)
```

```
plotPriceLevels(depth, spread, volume.scale = 1,
    col.bias = 0.25, show.mp = T,
                start.time = as.POSIXct("2016-03-10
     00:00:00.000"),
                end.time = as.POSIXct("2016-03-11
    00:00:00.000"))
```

- Extracting volume features for last 3 seconds

```
      all_features_df <- lapply(cusum_df$Timestamp,
     function(timestamp) {
        df <- data.frame(Timestamp = seq(timestamp
    -3, timestamp-0.001, by = 0.001),
                         stringsAsFactors = F)
        df$Side <- "BUY"
        df1 <- df
        df1$Side <- "SELL"
        df <- rbind(df, df1)
        df <- merge(df, last_df, all.x = T, all.y =
     F, by = c("Timestamp", "Side"))
        cols <- setdiff(colnames(df), "Timestamp")
        if(is.na(df$p1[1])) {
          df1 <- last_df[last_df$Timestamp < df$
    Timestamp[1] & last_df$Side == "BUY",]
          df1 <- df1[nrow(df1),]
          set(df, 1, cols, as.list(df1[1, cols,
    with = F]))
        }
        if(is.na(df$p1[2])) {
          df2 <- last_df[last_df$Timestamp < df$
    Timestamp[1] & last_df$Side == "SELL",]
          df2 <- df2[nrow(df2),]
          set(df, 2, cols, as.list(df2[1, cols,
    with = F]))
        }
        df <- data.frame(df)
        df <- fill_na_df(df)
        df <- df[seq(10, nrow(df), by = 10), ]
        return(df)
      })
```

- Labeling and subsetting features with labels in {-1, 1}

```
label_meta <- function (x, events, ptSl, ex_vert =
    T, n_ex = 0, df_list = NULL)
{
  nBar <- length(x)
  t0 <- events$t0
  t1 <- sapply(events$t1, function(tt) {
    min(tt, nBar)
  })
  trgt <- events$trgt
  side <- events$side
  u <- ptSl[1]
  l <- ptSl[2]
  T_up <- T_lo <- label <- NULL
  out <- sapply(1:length(t0), function(i) {
    i_t0 <- t0[i]
    i_t1 <- min(t1[i], length(x))
    i_x <- x[i_t0:i_t1]
    i_nx <- length(i_x)
    i_trgt <- trgt[i]
    i_side <- side[i]
    if (i_side == 0) {
      up <- i_trgt * u
      lo <- i_trgt * l
      isup <- (i_x/i_x[1] - 1) >= up
      islo <- -(i_x/i_x[1] - 1) >= lo
      T_up <- ifelse(sum(isup) > 0, min(which(isup)
    ),
                     Inf)
      T_lo <- ifelse(sum(islo) > 0, min(which(islo)
    ),
                     Inf)
    }
    else if (i_side == 1) {
      up <- i_trgt * u
      isup <- (i_x/i_x[1] - 1) >= up
      T_up <- ifelse(sum(isup) > 0, min(which(isup)
    ),
                     Inf)
      T_lo <- Inf
    }
    else {
      lo <- i_trgt * l
      islo <- -(i_x/i_x[1] - 1) >= lo
      T_up <- Inf
      T_lo <- ifelse(sum(islo) > 0, min(which(islo)
    ),
                     Inf)
    }
    ret <- i_x[min(T_up, T_lo, i_nx)]/i_x[1] - 1
    label <- which.min(c(T_lo, i_nx + 1e-06, T_up))
     - 2
    rst <- c(T_up, T_lo, length(i_x), ret, label)
    return(rst)
  })
  out <- data.frame(t(out))
  names(out) <- c("T_up", "T_lo", "t1", "ret", "
    label")
  out$t0Fea <- c(n_ex + 1, t0[-length(t0)] - 1)
  out$t1Fea <- t0 - 1
  out$tLabel <- t0 - 1 + apply(out[, c("T_up", "T_
    lo", "t1")],
                               1, min)
  if (ex_vert == T) {
    df_list <- subset(df_list, !(is.infinite(out$T_
    up) & is.infinite(out$T_lo)))
    out <- subset(out, !(is.infinite(T_up) & is.
    infinite(T_lo)))
  }
  return(list(out = out, df_list = df_list))
}
```

- Scaling daily volumes based on highest and lowest volume of the day

```
max_arr <- rep(-1, length(files))
min_arr <- rep(100000, length(files))
```

```
for(i in 1:length(files)) {
  all_features_df <- readRDS(paste0("data/modeling/all_features_df/", files[i]))
  all_features_df[["ticker_file"]] <- NULL
  rows <- rows + length(all_features_df)
  if(length(all_features_df) > 0) {
    max_arr[i] <- max(max_arr[i], max(sapply(all_features_df, function(df) max(df[, volume_cols]))))
    min_arr[i] <- min(min_arr[i], min(sapply(all_features_df, function(df) min(df[, volume_cols]))))
  }
}
clusterExport(cl, c("volume_cols", "min_arr", "max_arr"))
mat <- matrix(c(-1, 1)*pi, nrow = rows, ncol = 6001)
# mat <- matrix(c(-1, 1)*pi, nrow = rows, ncol = 60001)
finished <- 0
for(i in 1:length(files)) {
  all_features_df <- readRDS(paste0("data/modeling/all_features_df/", files[i]))
  # BUY v1 to v10, SELL v1 to v10, BUY v1 to v10, ...
  clusterExport(cl, "i")
  all_features_df1 <- parLapply(cl, all_features_df, function(df) {
    if(class(df) == "data.frame") {
      # Order step: BUY v1 to v10, SELL v1 to v10, BUY v1 to v10, ...
      df <- df[order(df$Timestamp, df$Side),]
      df[df$Side == "BUY", volume_cols] <- (df[df$Side == "BUY", volume_cols] - min_arr[i])/(max_arr[i] - min_arr[i])
      df[df$Side == "SELL", volume_cols] <- (df[df$Side == "SELL", volume_cols] - min_arr[i])/(max_arr[i] - min_arr[i])
      # df[df$Side == "BUY", volume_cols] <- scale(df[df$Side == "BUY", volume_cols])
      # df[df$Side == "SELL", volume_cols] <- scale(df[df$Side == "SELL", volume_cols])
      df$split <- 1:nrow(df)
      df_list <- split(df, df$split)
      df_final <- lapply(df_list, function(row) row[, volume_cols])
      df_final <- do.call(cbind, df_final)
      return(df_final)
    }
  })
  all_features_df1[["ticker_file"]] <- NULL
  all_features_df1 <- do.call(rbind, all_features_df1)
  file <- filenames[[i]][1]
  labels <- readRDS(paste0("data/modeling/labels/", file, "_labels.Rds"))
  all_features_df1 <- data.frame(all_features_df1)
  all_features_df1$label <- labels$label
  all_features_df1 <- as.matrix(data.frame(sapply(all_features_df1, unlist)))
  if(ncol(all_features_df1) != 60001 & nrow(all_features_df1) == 60001) {
    all_features_df1 <- t(all_features_df1)
  } else if(ncol(all_features_df1) != 6001 & nrow(all_features_df1) == 6001) {
    all_features_df1 <- t(all_features_df1)
  }
  if(nrow(all_features_df1) > 0) {
    mat[(finished+1):(finished + nrow(all_features_df1)), ] <- all_features_df1
    finished <- finished + nrow(all_features_df1)
    if(length(all_features_df) != (nrow(labels)+1)) {
      print(file)
    }
  }
  print(dim(all_features_df1))
  # full_df <- rbind(full_df, all_features_df1)
}
```

B. References

- Deep learning for limit order books (Sirignano, 2016)
- Modeling high-frequency limit order book dynamics with support vector machines (Kercheval et al., 2013)
- Sequence classification of the limit order book using Recurrent Neural Networks (Dixon, 2017)
- Reinforcement learning for optimized trade execution (Nevmyvaka et al., 2017)
- Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods (Ntakaris et al., 2018)