Predictive Analysis

Aamodini Gupta,[1] Bill Zheng,[1] Marc Lee,[1] and Naveen Sathiyanathan[1]
STAT 430 (Fall 2018) - Group 4

(Dated: 20 December 2018)

The objective of this project was to explore the application of deep learning in the analysis of limit order books. Specifically, the target was to predict the direction of movement of VWAP smoothed over 60 seconds. Deep learning algorithms including fully connected network, CNN, RNN and hierarchical RNN were applied. Randomized grid search was used for hyper parameter tuning and the generalization of each model was measured on a hold-out set and the best set of hyper parameters were identified. It was found that different methods perform better for different tickers. Among the algorithms that were tested, hierarchical recurrent neural network model performed the best on ticker 1 in predicting the direction of movement of VWAP. The results were compared to that of shallow machine learning models such as random forest, naive Bayes, conditional inference trees, XGBoost, etc. It was observed that shallow machine learning algorithms perform worse than random guess.

I. INTRODUCTION

Since the stock market was introduced to the world, predicting the Stock Market has been a one of the most famous goals of investors. In the market, more than billions of dollars are traded with the hope of investors to make profits from their decisions. Similar to the bitcoins discussed in the previous project, the stock trading is run by investors order made in less than a second period. As the market fluctuates everyday, entire companies rise and fall daily based on the different behaviors of the market.

It is no wonder that the Stock Market and its associated challenges are now publicly open and throw the questions. The 2008 financial crisis was an example, as evidenced by the media based on the financial market crash. If there was a common theme among those media sources, it was that few people knew how the market worked or reacted. So, it is highly believed that a better understanding of stock market prediction might help in the case of similar events in the future.

II. DATA DESCRIPTION

The dataset contained information directly from the Exchange feed on the 10 levels of buy/sell prices of 4 different futures contracts (denoted as Tickers 1-4) in a 24 hour period. Each level gave three pieces of information - the price, contract, and the number of orders. Buy/Sell is also specified in the dataset. The timestamp was in the format "YYYYMMDDHHMMSSMMM" where the last 3 digits were milliseconds, the next 2 were seconds, the next 2 were minutes and the 2 after that were hours.

The data sets for each ticker had different number of rows within the same day. Therefore, it can be concluded that the liquidities are different. Number of rows of raw data for each stock's ticker can be found below:

Ticker	Number of Rows	Number of Labels
1	3355436	71192
2	2018898	59818
3	3286516	67389
4	1071070	58333

There were no missing values within each of the datasets, but as the dataset was merged column-wise based on the timestamp, the unique timestamps did not match. In this case missing values were imputed using time series interpolation.

III. EXPLORATORY ANALYSIS

Tick data was very dense, which made visualization very difficult. Also there could be multiple values of price for buy and sell side within the same second. A simple visualization of the first minute of tick data for ticker 1 on the given trading day could be found below.

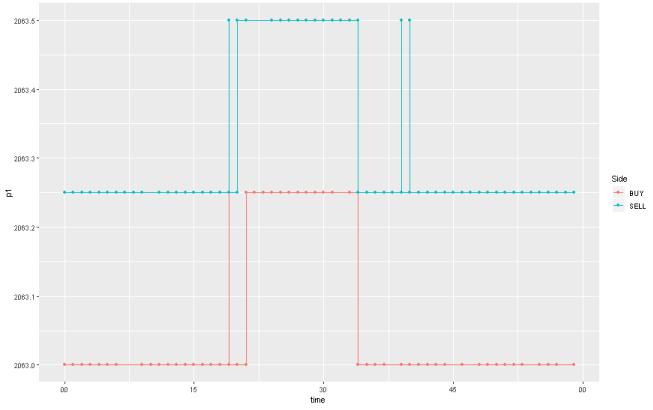


FIG. 1. Buy and Sell.

We observed that despite the richness of the data set, ticks were recorded between discrete price points with an interval of $0.25.

The data set was then aggregated to 1000 milliseconds and relevant features were created (discussed in feature engineer-

ing). A simple visualization of features from the first 1 minute of ticker 1 can be found below:

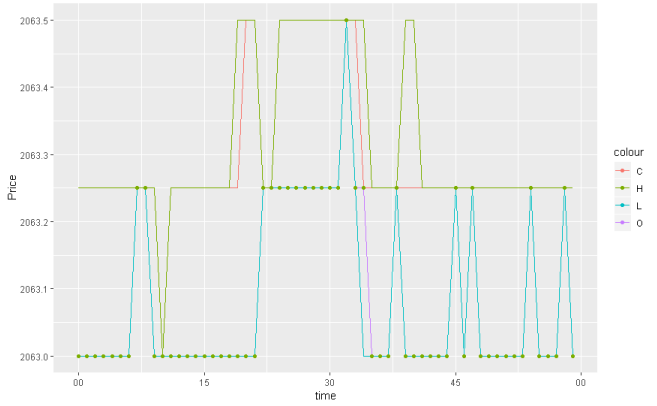


FIG. 2. Close, High, Low and Open Prices for Ticker 1 against Time.

It was also observed that the tickers are in different scales. In most cases the variation in close price was not comparable to the scale of the close price. Therefore, scaling of variables was found to be necessary for faster learning and when regularization was used.

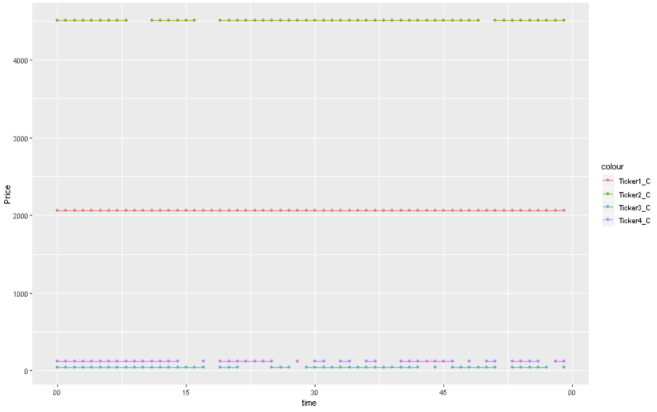


FIG. 3. Unscaled Close Prices against Time.

The close prices are brought to approximately the same scale after scaling. The plot of scaled features subsetted for first minute can be found below.

IV. FEATURE ENGINEERING

In order to get the useful features from the limit order book raw data, the format of timestamp was preprocessed from millisecond to second. Level 1 was expected to contain most of the information about the behavior of the ticker in market. Also, usage of levels 2-10 might add noise to the analysis. Therefore, only level 1 was processed and used for downstream analysis for each ticker.

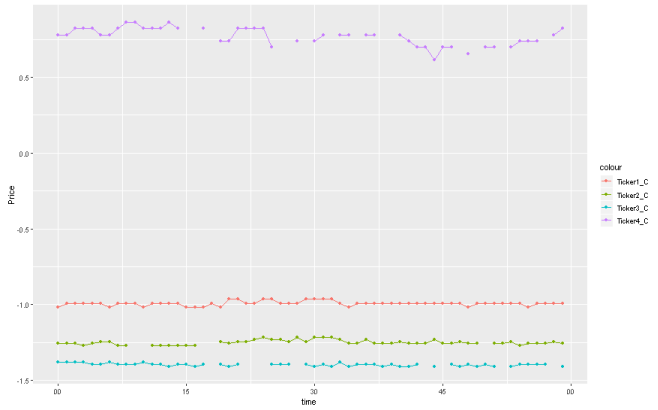


FIG. 4. Scaled Close Prices against Time.

It was observed that trading occurred throughout the day. There was no missing data in the data set. However, multiple timestamps were missing in the aggregated data. Also, there were multiple timestamps that were available in one ticker data but not in the others'. Time series interpolation was used to impute these missing timestamps.

A. Time Bars

All 4 tickers, were aggregated to 1000 milliseconds to get time bars with OHLCV. Missing values were filled for all tickers because parametric models like neural networks cannot handle missing data. Data from level 1 which had the smallest price for sell side and largest price for buy side was used to calculate OHLCV.

V. METHODOLOGY AND ANALYSIS

Discriminative machine learning and deep learning models capture patterns that differentiate between the output classes. No universal theory or guideline exists for the analysis of limit order book data using statistical or machine learning models. The structure of data generating process in limit order book is unknown. Therefore, the true underlying model is assumed to be unknown. Therefore, errors on validation and test sets were estimated to understand the generalization of models. Hyper parameter tuning was performed and empirical error (bias + variance) was compared for model selection.

$$TotalError = Error_{bias} + Error_{variance}$$

A. Train-Validation-Test based on Time

The data set was preprocessed by capturing information from level 1 only. After creating features such as open, close, high, low and volume, the data set was split into train, test and validation sets based on time as:

- Train: Data from first 60% of the day
  Validation: Data from 60-80% of the day
  Test: Data from last 20% of the day

It was assumed that the test set will be a representative of the next day's data. However, this assumption was relaxed in ticker 2 where the model accuracy on test set was worse than that of a random guess (33%). In such cases it was assumed that the validation set acts as a representative sample for the next day.

B. Model Training Scheme

The following set of hyper parameters were tuned during the analysis:

- Fully connected layers:
  - Number of neurons
  - Dropout
- CNN:
  - Number of filters in each 2D convolution layers
  - Number of neurons in dense layer
- RNN:
  - LSTM units
  - Number of neurons in dense layer
- HNN:
  - Column encoding units
  - Row encoding units

C. Methods Used

1. *Fully Connected Layers*

The fully connected layers learn features from each of the previous layers. There is typically a linear operation, like weighted averages, that connects the input to the output. This is then followed by a nonlinear activation.

2. *CNN*

Convolution is a window-based filtering operation which leads to aggregated features from a tensor with 1 or more dimensions. Convolution layers act as feature extractors for the downstream tasks such as classification. Different types of filters can be applied and stacked together for downstream layers to learn from.

3. *RNN*

It is called Recurrent because the way it perform is recursive, having a backward connection, unlike other feed-forward neural networks. In other word, it has a "memory" which can trace and keep information about the computations done so far.

4. *Hierarchical RNN*

Hierarchical recurrent neural networks are similar to recurrent neural networks - they perform encoding. However, they perform encoding hierarchically between columns and rows. Structures learnt by the network can be quite different from that of recurrent neural networks.

5. *Shallow Machine Learning Algorithms*

Algorithms such as naive Bayes, linear discriminant analysis, SVM, conditional inference tree, random forest and XGBoost were used to predict the outcome based on given predictors. However, their architectures considered each observation independently for training and prediction, which was not robust for financial machine learning.

D. Hyper-Parameter Tuning

Neural networks are susceptible to variation in performance based on the hyper parameters. It was observed that different model hyper parameters led to significantly different performance and generalization. The following sets of hyper parameters were tuned:

	Train Acc	Validation Acc	Test Acc	Parameter		
				Number of Hidden Layers	Hidden Layer Structure	Dropout
TICKER 1	.3972	.3358	.3076	3	(8)	0.2
TICKER 2	.3253	.3142	.3002	3	(8)	0.2
TICKER 3	.3762	.3559	.3293	3	(8)	0.2
TICKER 4	.3302	.3018	.2974	3	(8)	0.2

	Train Acc	Validation Acc	Test Acc	Parameter		
				2D Convolution Layers	2D Convolution Structure	Dense Hidden Layer Structure
TICKER 1	0.4077	0.3648	0.337844	3	(16, 16, 16)	(8)
TICKER 2	0.4147	0.3523	0.3215798	3	(16, 16, 16)	(8)
TICKER 3	0.4341	0.3882	0.3012985	3	(16, 16, 16)	(8)
TICKER 4	0.4128	0.3264	0.3399483	3	(16, 16, 16)	(8)

	Train Acc	Validation Acc	Test Acc	Parameter		
				Window size (w)	Hidden Neurons	LSTM Units
TICKER 1	0.3917	0.3130	0.3155532	60	4	12
TICKER 2	0.4180	0.3486	0.3291597	60	4	20
TICKER 3	0.4672	0.3983	0.306806	60	24	8
TICKER 4	0.3850	0.3066	0.3400517	60	4	8

	Train Acc	Validation Acc	Test Acc	Parameter		
				Window size (w)	Column Encoding Units	Row Encoding Units
TICKER 1	0.3988	0.3678	0.3804965	60	20	10
TICKER 2	0.4041	0.3573	0.3143025	60	20	15
TICKER 3	0.4452	0.3837	0.4117015	60	15	8
TICKER 4	0.3900	0.3112	0.3433448	60	15	20

1. *Fully Connected Layers*

2. *CNN*

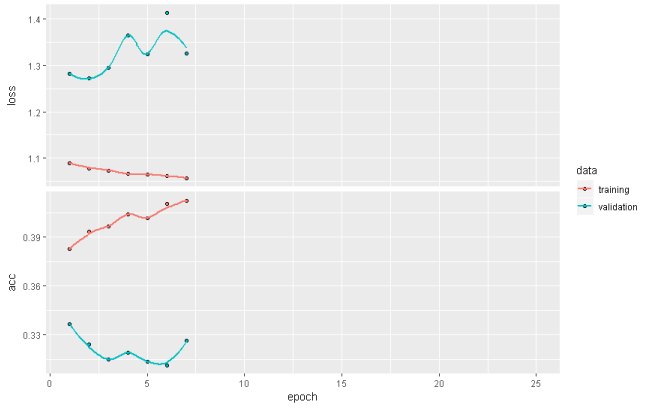
3. *RNN*

4. *Hierarchical RNN*

Example of Training

Models weights were initialized randomly. Model training is expected improve the weights in order to decrease the loss. The performance of model training was tracked on training and validation sets. A plot of training errors vs epochs (number of time the data was used for learning) can be found below:

This model showed signs of overfitting, although there were other models which showed stable behavior across epochs.



VI. RESULTS

	Final Model	Train Acc	Val Acc	Test Acc
TICKER 1	HRNN	0.3988	0.3678	0.3804965
TICKER 2	RNN (LSTM)	0.4180	0.3573	0.3291597
TICKER 3	HRNN	0.4452	0.3837	0.4117015
TICKER 4	HRNN	0.3900	0.3112	0.3433448

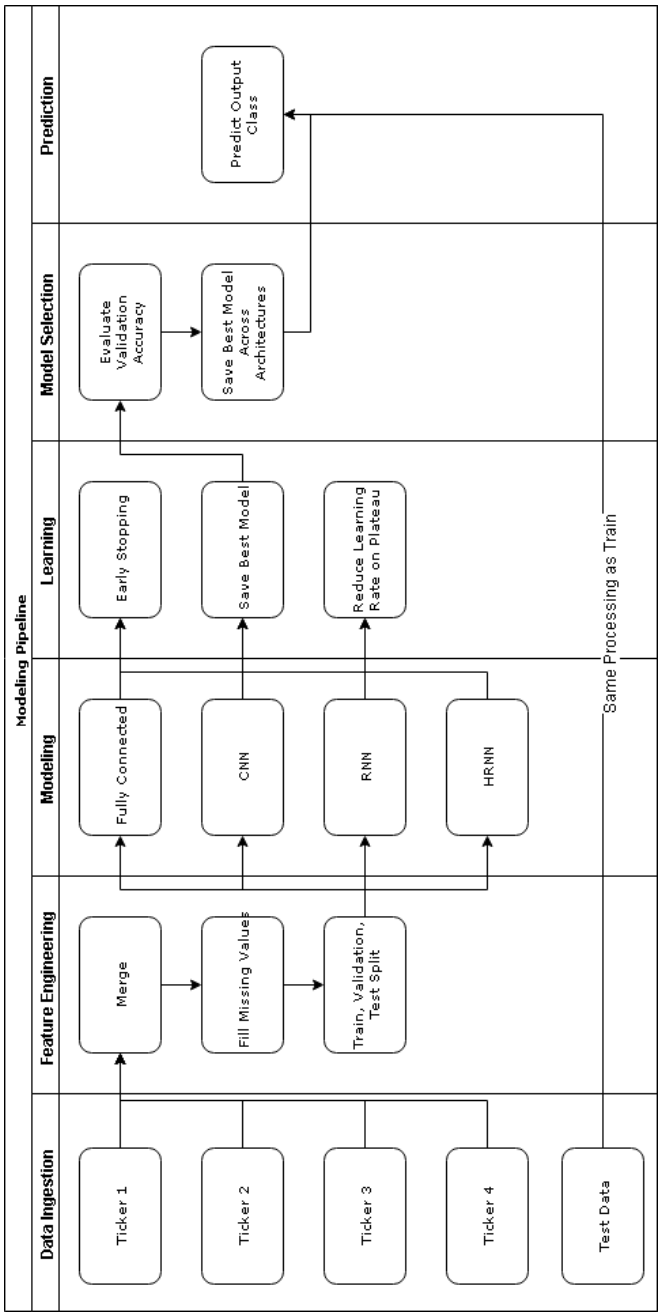
Please refer Appendix for results of shallow machine learning algorithms.

A. Scope for Improvement

- Shallow machine learning models such as random forest and XGBoost were tried, but their accuracies were worse than that of random guess. Therefore, their predictions were considered unreliable and they were removed from the analysis. However, this gave us a clue that the features are not very predictive.
- The current set of features have low predictive power. Additional features such as news (text), forecasts that might capture the relationship to the stock's moving direction would fit a better model.
- Applying 'voting ensemble' method to a classes predicted by different models may improve accuracy.
- Trying out other neural network architectures, traditional machine learning methods to have a heterogeneous set of predictions and stacking their predictions may improve accuracy.
- Tuning of hyperparameters took lot of time. Scope for experimentation was low.

APPENDIX

1. Project Workflow



2. Sample R code for the overall flow

```
me_train <- apply(trainDat[, -which(colnames(
  trainDat) == "direction")], 2, mean)
sd_train <- apply(trainDat[, -which(colnames(
  trainDat) == "direction")], 2, sd)
for(name in names(me_train)) {
  trainDat[, name] <- (trainDat[, name] - me_
  train[name])/sd_train[name]
  testDat[, name] <- (testDat[, name] - me_train[
  name])/sd_train[name]
  valDat[, name] <- (valDat[, name] - me_train[
  name])/sd_train[name]
}

trainDat$direction <- as.factor(trainDat$
  direction)
testDat$direction <- as.factor(testDat$direction)
valDat$direction <- as.factor(valDat$direction)

X_data_train <- as.matrix(trainDat[, -which(
  colnames(trainDat) == "direction")])
Y_data_train <- as.vector(trainDat$direction)
X_data_val <- as.matrix(valDat[, -which(colnames(
  valDat) == "direction")])
Y_data_val <- as.vector(valDat$direction)
X_data_test <- as.matrix(testDat[, -which(
  colnames(testDat) == "direction")])
Y_data_test <- as.vector(testDat$direction)

source("runs_cnn.R")
source("runs_hrnn.R")
saveRDS(me_train, paste0("mean_", ticker, ".Rds")
  )
saveRDS(sd_train, paste0("sd_", ticker, ".Rds"))
```

3. Sample R code for building maintaining Tensorflow runs of hierarchical recurrent neural network

```
runs <- tuning_run("modeling_hrnn.R", flags = list(
  row_hidden = rev(c(8, 10, 15, 20)), col_hidden =
    rev(c(8, 10, 15, 20))
), sample = 0.1, confirm = F)
```

4. Sample R code for building hierarchical recurrent neural network

```
model <- keras_model_sequential() %>%

  model %>% compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(lr = 1e-4),
    metrics = c("acc")
  )

##############
# run model #
##############
batch_size <- 100
print(w)
his <- model %>%
  fit_generator(
    sampling_generator0(
      X_data_train, Y_data_train, batch_size =
    batch_size, w=w),
    steps_per_epoch = floor((nrow(X_data_train)-w
    +1) / batch_size),
    epochs = 10,
    callbacks = list(checkPoint, reduceLr, logger,
    earlyStopping),
    validation_data = sampling_generator0(
      X_data_val, Y_data_val, batch_size = batch_
    size, w=w),
    validation_steps = floor((nrow(X_data_val)-w+1)
     / batch_size))
plot(his)
```

5. Sample R code for prediction on test set

```r
me_train <- readRDS(file.path(train_mean_sd_dir, paste0("mean_", ticker, ".Rds")))
sd_train <- readRDS(file.path(train_mean_sd_dir, paste0("sd_", ticker, ".Rds")))
X_data_test <- as.matrix(reqd_ticker_df[, 4:ncol(reqd_ticker_df)])
for(col in names(me_train)) {
  X_data_test[, col] <- (X_data_test[, col] - me_train[col])/sd_train[col]
}



# Loading the model and predicting
model <- load_model_hdf5(file.path(h5_models_dir, models[ticker]))
batch_size <- 100
w <- 60
pred_lis[[ticker]] <- get_predictions(X_data_test, w, batch_size, ticker)
```

6. References

- The Data Source
  AlgoSeek
- Reference Book
  Deep Learning with R by Francois Chollet, J.J. Allaire

7. Shallow machine learning on Ticker 1

Model	Train Acc	Val Acc
Linear discriminant analysis	0.287	0.277
SVM	No result (high run time)	No result (high run time)
C-tree	0.291	0.279
XGBoost	0.285	(Not measured)
Random forest	0.285	0.278