High-Frequency Trade Sign Classification

Deep Learning - Final Project

Jared Hansen, Matt Isaac, Kegan Penovich

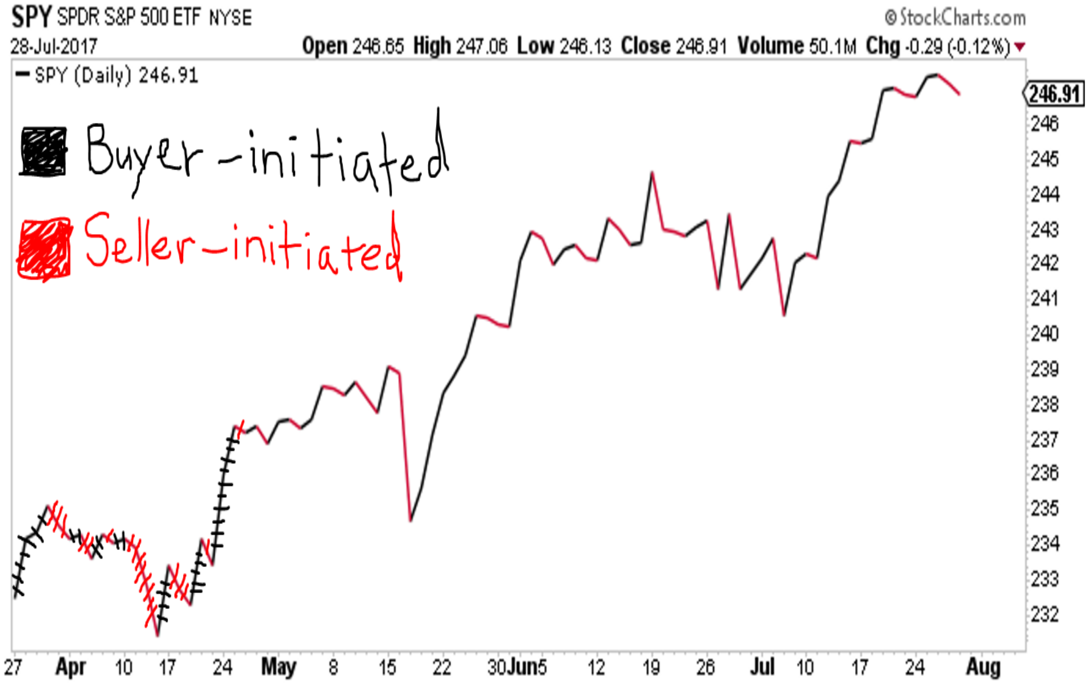
23 April, 2019

Outline

- **Problem & Motivation**
- **Data**
- **Baseline Models** (forests and k-NN)
- **Neural Network Models** (feedforward, LSTMs)
- **Conclusions & Future Work**

Problem and Motivation

- **PROBLEM**: classify high-frequency trade (HFT) initiation



Problem and Motivation

- **MOTIVATION**: Finance researchers often don't have ITCH access, but need to know trade signs (liquidity measures, price-estimation models)
- Target accuracy: surpass accuracies in Rosenthal's 2012 "Modeling Trade Direction"

		Percent of trades correctly classified				
Sector	*N*	Modeled	EMO	LR.new	LR.old	Tick
Capital goods	216,800	**74.7**	73.0	72.1	71.8	61.6
Conglomerates	33,863	**84.7**	83.4	79.5	78.9	63.7
Cons. cyclical	236,193	**73.4**	72.1	71.7	71.4	62.9
Energy	228,978	**77.3**	76.1	73.3	72.9	62.5
Financial	1,014,479	**74.2**	72.4	72.4	72.2	63.3
Healthcare	2,314,251	**72.2**	71.5	69.7	69.4	63.8
Industrial goods	4917	62.5	**63.8**	60.4	60.3	54.3

Data

- ITCH data feed (Nasdaq product)
- Dataset includes all 21 trading days in March 2018
- 10 Tickers (stocks)
    - **AAOI** (Applied Optoelectronics Inc.)
    - **BABY** (Natus Medical Inc.)
    - **CA** (CA Technologies)
    - **DAIO** (Data I/O Corporation)
    - **EA** (Electronic Arts Inc.)
    - **FANG** (Diamondback Energy Inc.)
    - **GABC** (German American Bancorp Inc.)
    - **HA** (Hawaiian Holding Inc.)
    - **IAC** (InterActiveCorp)
    - **JACK** (Jack in the Box Inc.)

Data - Preparation and Cleaning

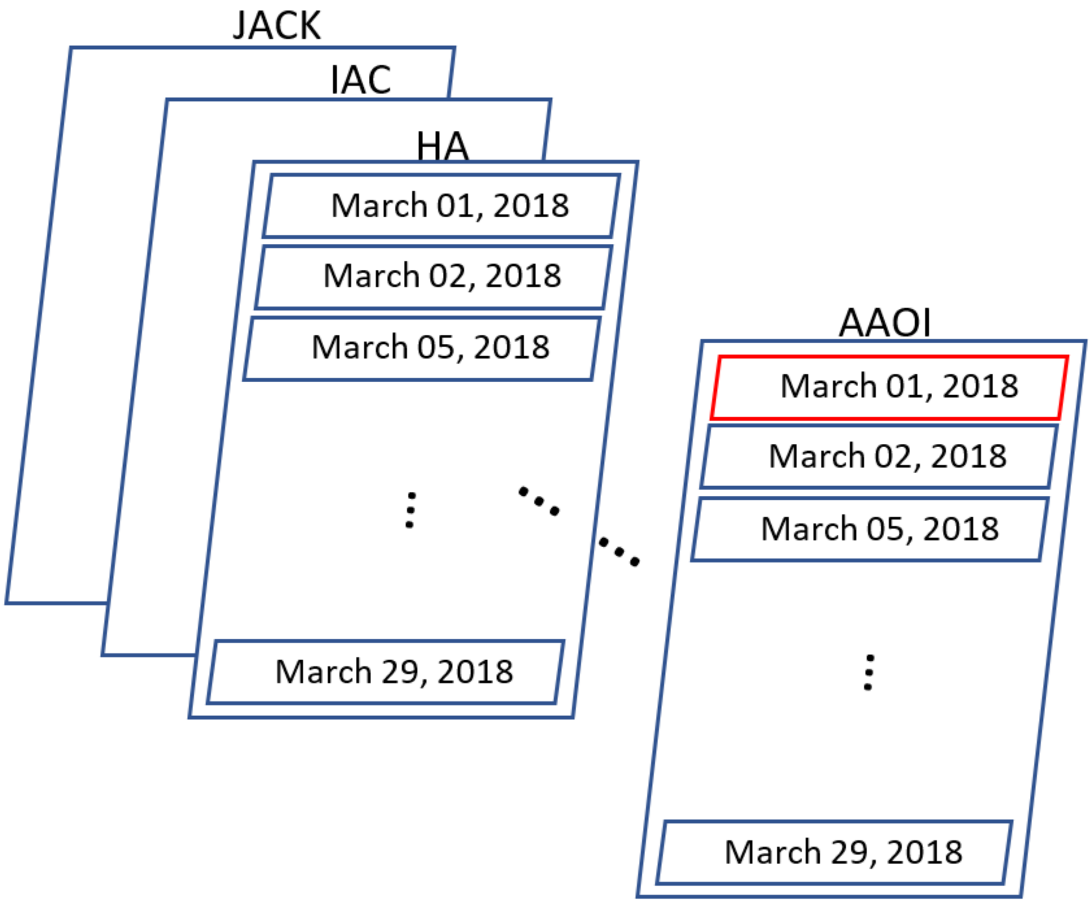
- 10 stocks x 21 days = 210 text files
- For each text file:
  - Remove all bids, asks, cancels, etc., leaving only executed trades
  - Convert to .csv file
- Result: total of about 700,000 observations
- Standardized within each variable (except the binary msg_type features) across each dataset, e.g.

$$standardized(timestamp_i) = \frac{timestamp_i - mean_{timestamp}}{\sigma_{timestamp}} \; \forall i, i \in \{1, 2, ...len(dataframe)\}$$

Data Aggregation

Two ways we aggregated the data for training:

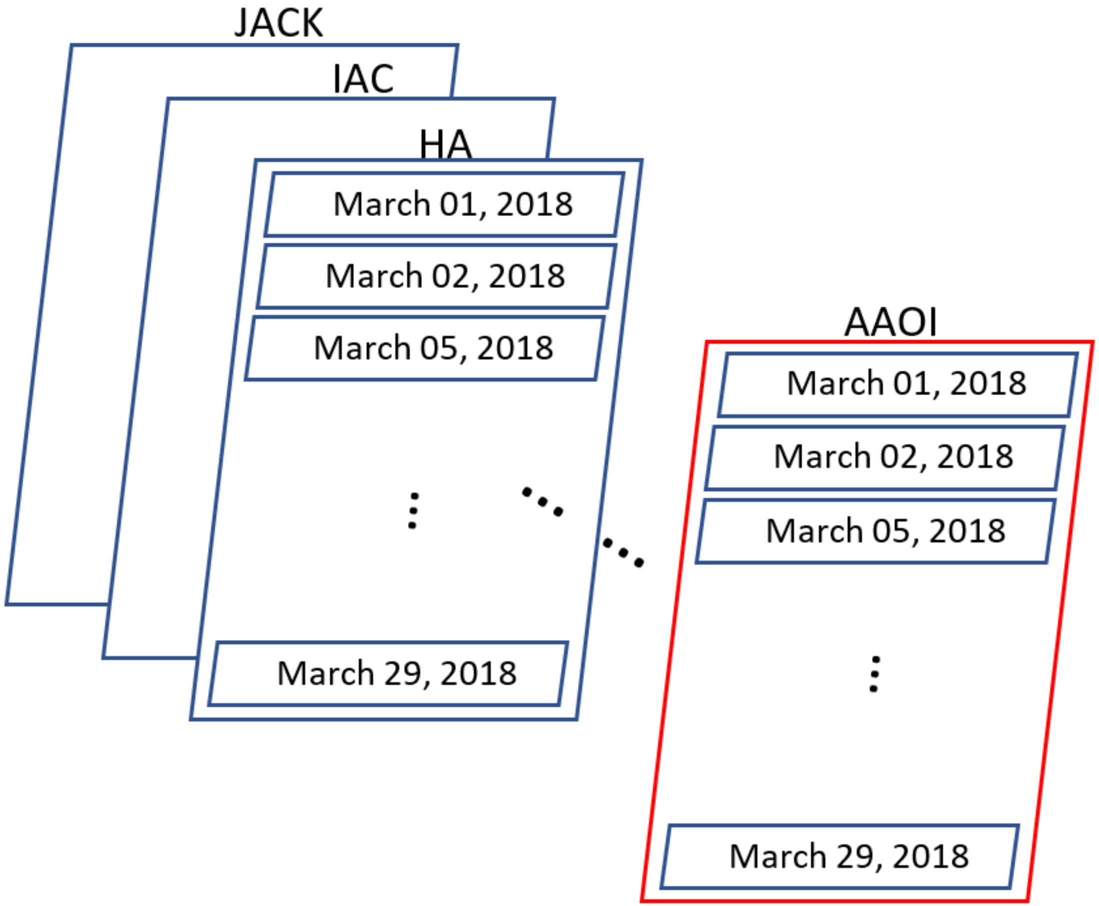
1. **One ticker, one day**
2. One ticker, all days



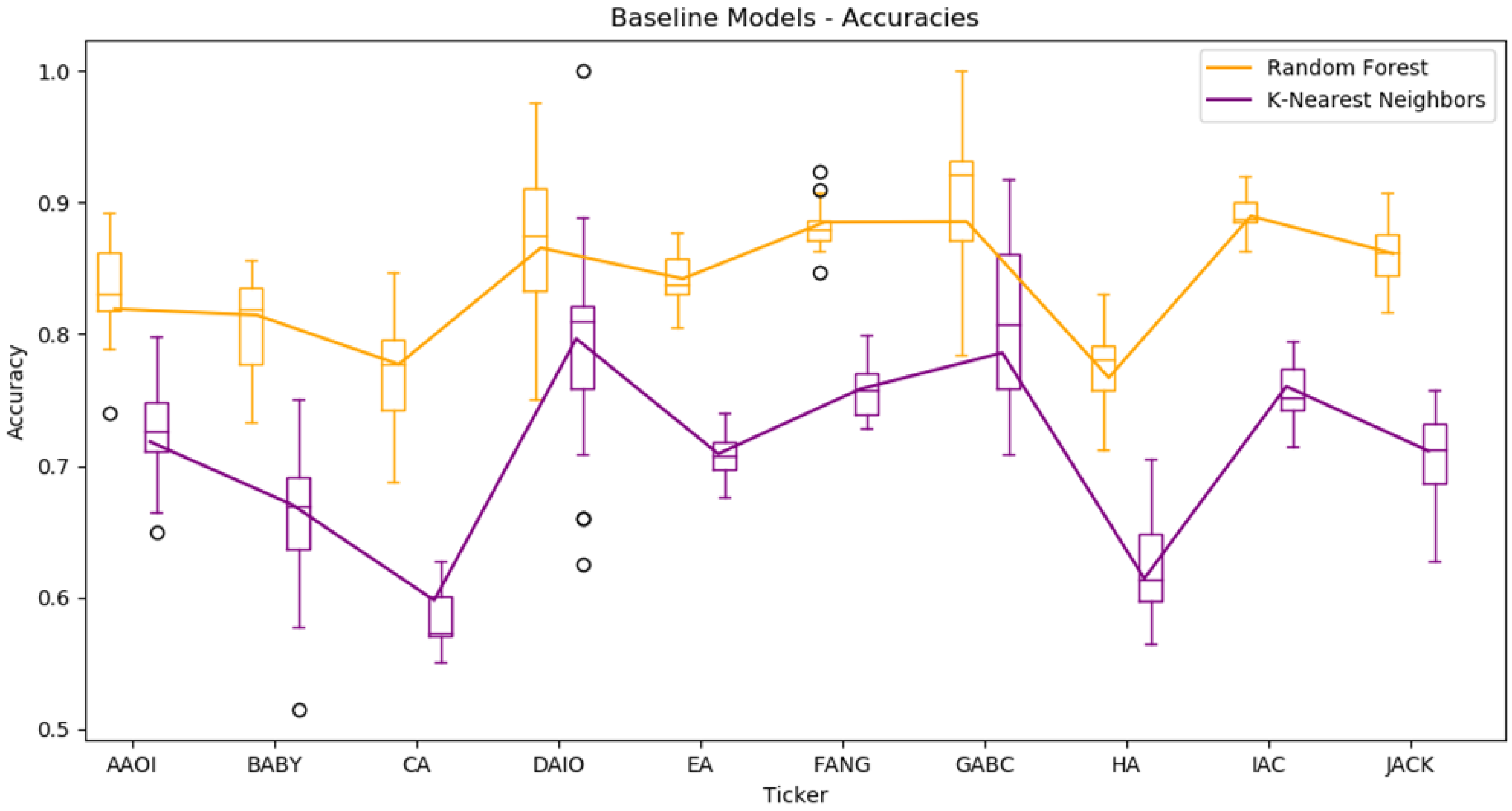
Data Aggregation

Two ways we aggregated the data for training:

1. One ticker, one day
2. **One ticker, all days**

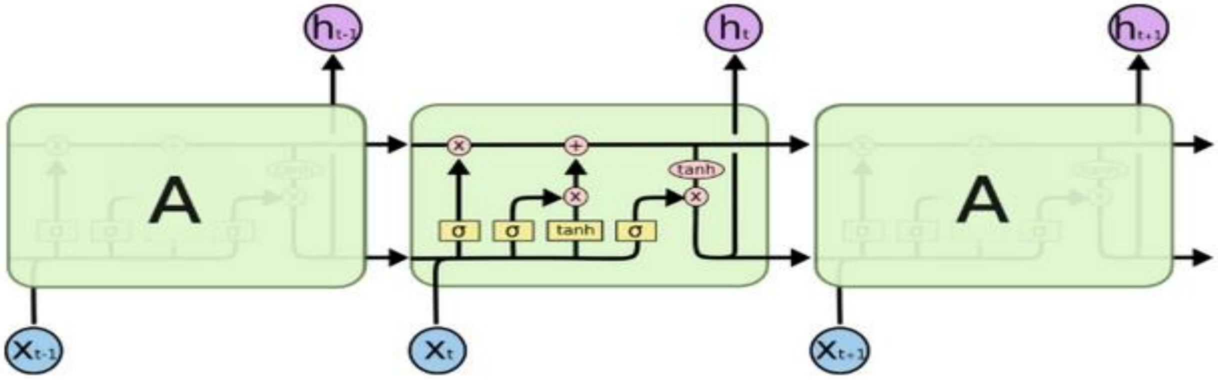


Baseline Models: k-NN & Random Forests



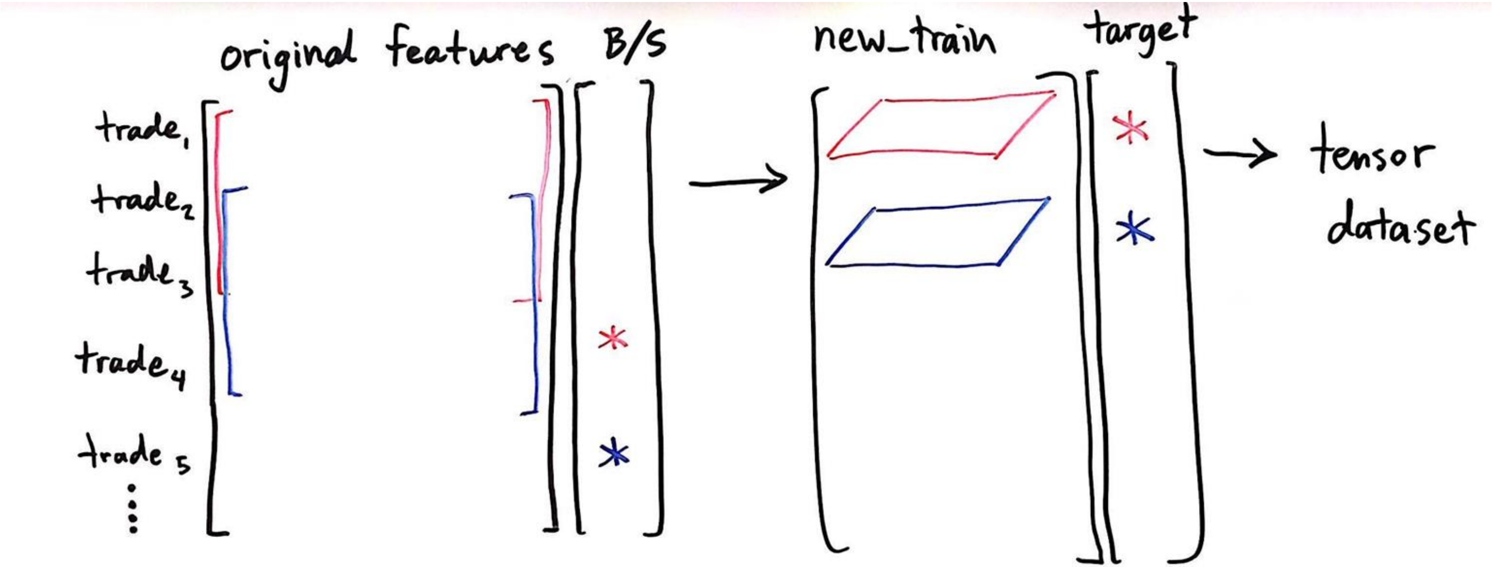
Bidirectional LSTM: Keras

- Since there is a time series aspect to the data, we could consider a RNN. LSTM's are the canonical choice for this kind of problem.
- We do not need to take this into account for our stated purpose, but we can try it to see if it can assist.



Better Leveraging B-LSTM

- Data pre-processing: more predictive sequence inputs



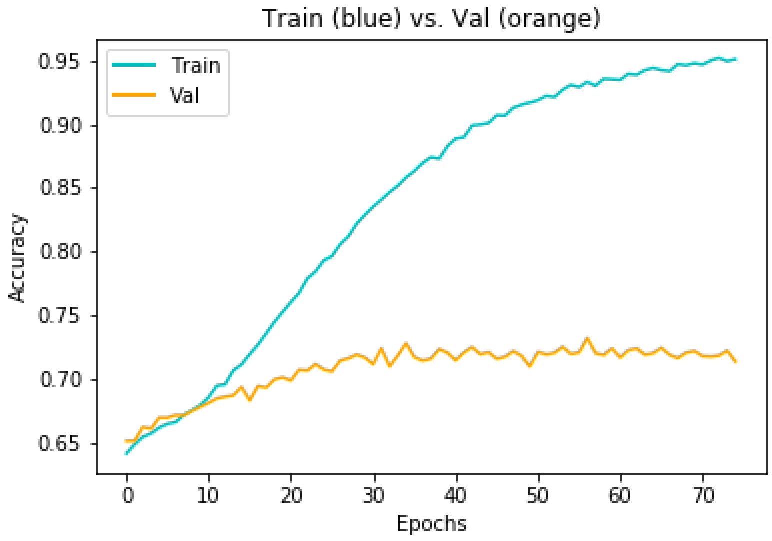
Bidirectional LSTM Architecture

- We kept this very basic.
- Single LSTM layer
- Dropout = 0.2
- Single dense layer = 100 neurons
- Sigmoid output
- BCE loss function
- Adam optimizer (defaults)

```
epochs = 75
batch_size = 64
keras.optimizers.Adam(lr=0.01, beta_1=0.9,
                      beta_2=0.999, epsilon=None,
                      decay=0.0, amsgrad=False)
ourmodel = Sequential()
ourmodel.add(Bidirectional(LSTM(100, input_shape =(33,11),
                                bias_regularizer = regularizers.l2(0.03),
                                dropout = 0.1, recurrent_dropout = 0.1)))
ourmodel.add(Dropout(0.2))
ourmodel.add(Dense(1, activation='sigmoid'))
ourmodel.compile(loss='binary_crossentropy',
                 optimizer='adam', metrics=['accuracy'])
ourmodel.fit(X_train, y_train, validation_data=(X_val, y_val),
             verbose = 2, shuffle = True,
             epochs=epochs, batch_size=batch_size)
print(ourmodel.summary())
```

B-LSTM Results (w/out regularization)

- A typical result (regardless of parameters) is in the low to mid 70’s.
- There was a serious issue of over fitting that would happen.
- Initial efforts to address this did reduce overfitting, but did not increase validation accuracy.

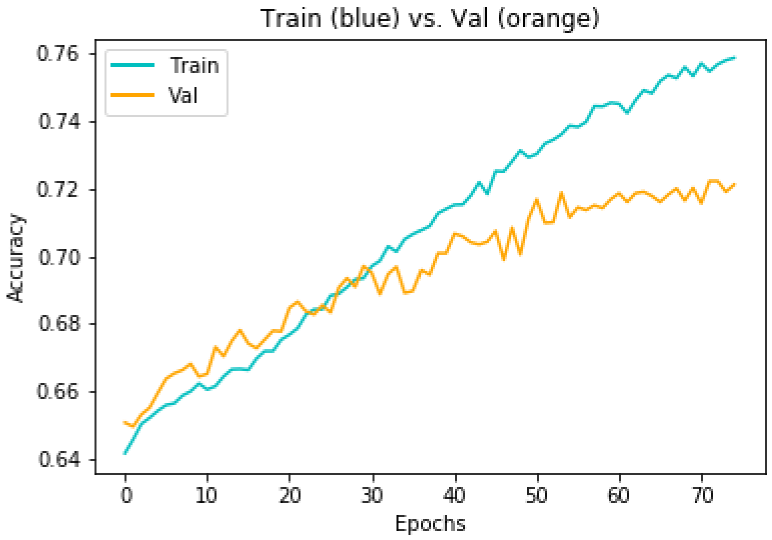


B-LSTM Results (w/regularization)

To deal with the overfitting we implemented some regularization techniques.

- L2 Regularization (0.03)
- Drop Out (0.1, 0.1, 0.2)

These drastically reduced overfitting, though it did not improve the validation accuracy.



Bidirectional LSTM (PyTorch)

- Amazing results! So good that they broke mathematical law

```
   ...:         correct += (predicted == trade_signs).sum()
   ...:
   ...:     print('Test Accuracy of the model on the test t
total))
Test Accuracy of the model on the test trades: 1278.0881 %
```

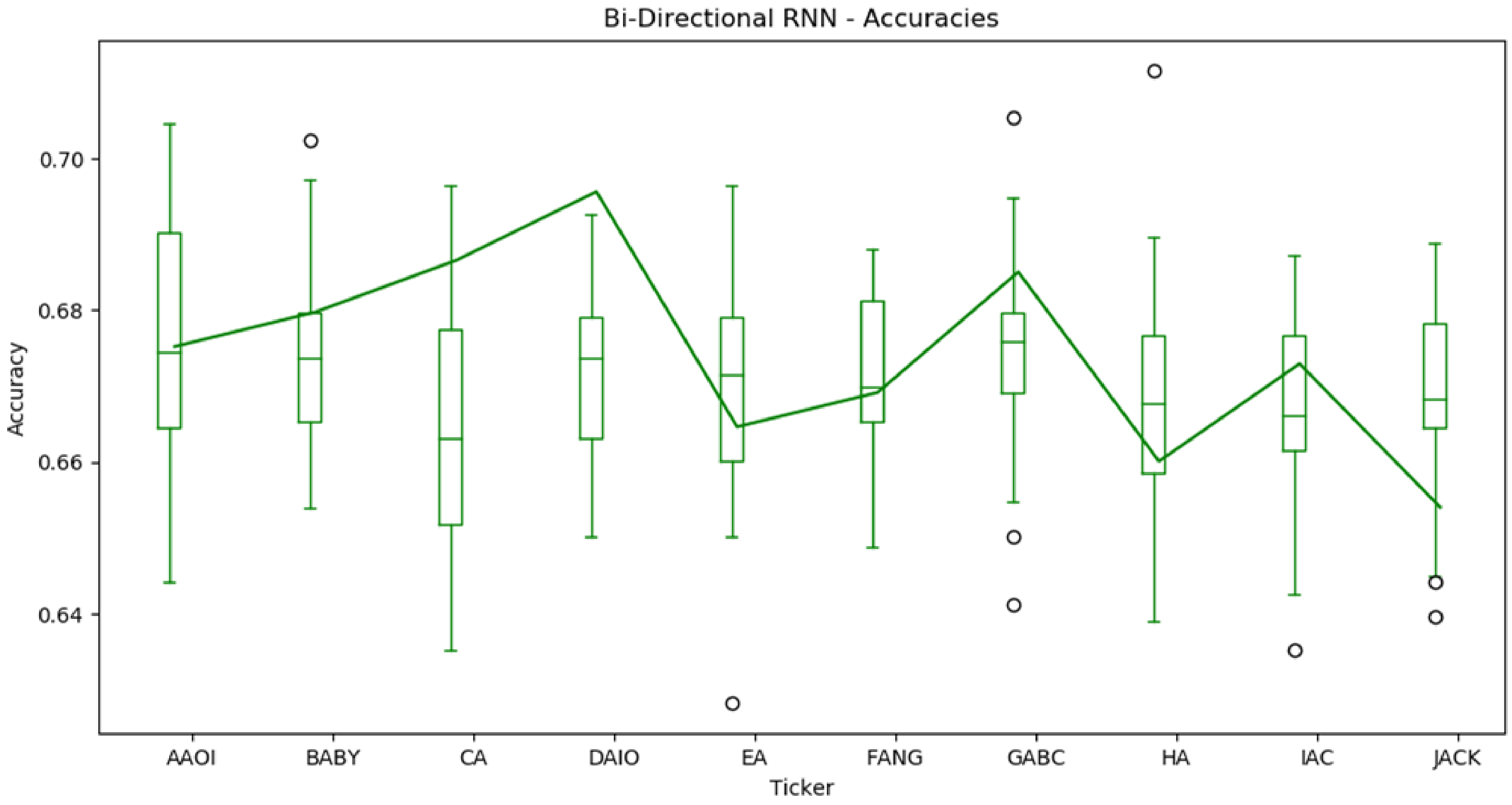
- Unfortunately this wasn't the case

```
Test Accuracy of the model on the test trades: 66.5320 %
```

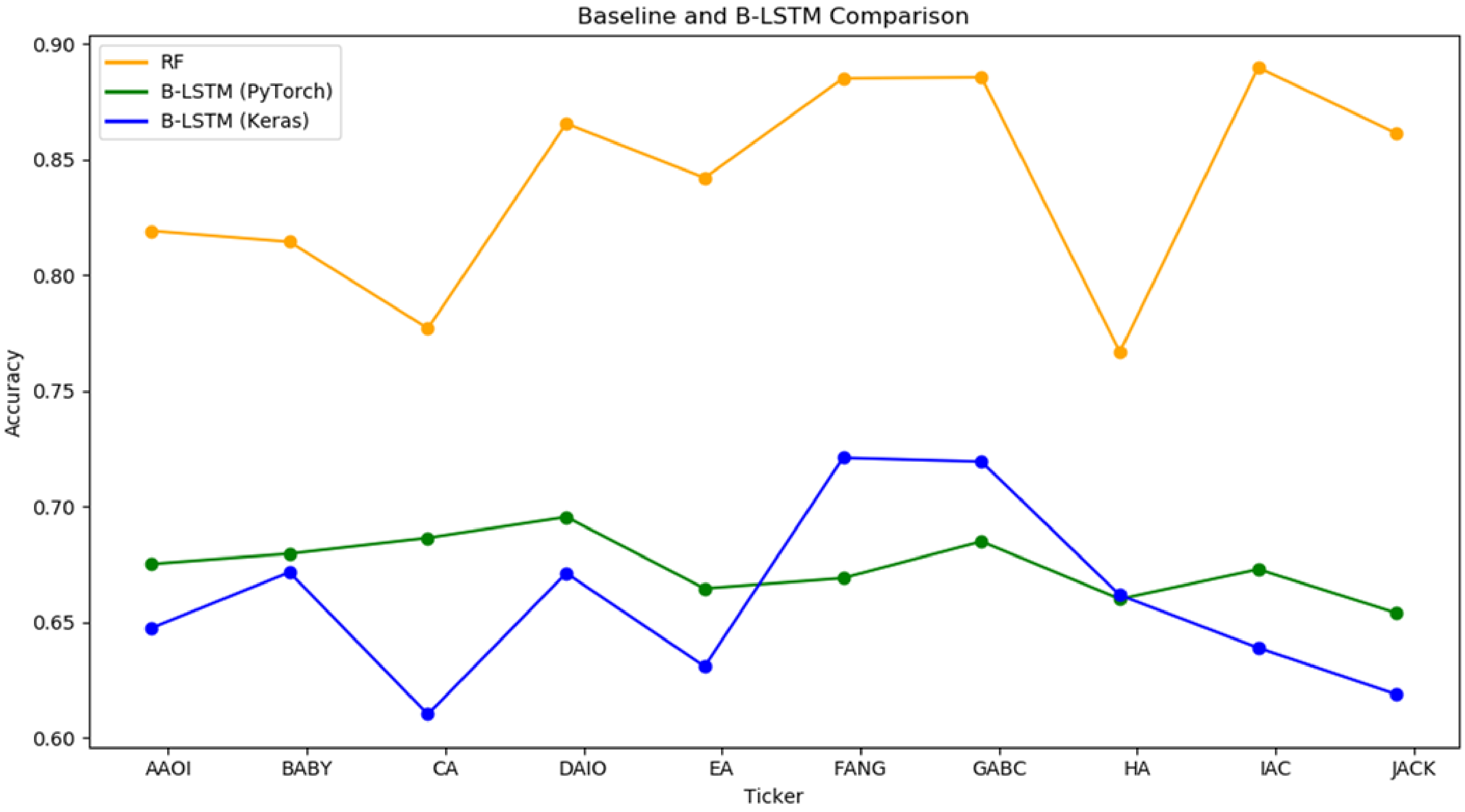
B-LSTM Architecture

- Window_length = 10 (10 sequences of features data per buy/sell target)
- Shift_by = 1 (maximizing utility of LSTM hidden & cell states)
- 2 hidden layers per cell, 128 neurons
- 1 output neuron, Sigmoid activation, round value to get 0 or 1
- Cost = BCE
- Learning rate = 0.001
- Optimizer = Adam
- Num_epochs = 4
- Batch_size = 20

B-LSTM Test Accuracies (PyTorch)

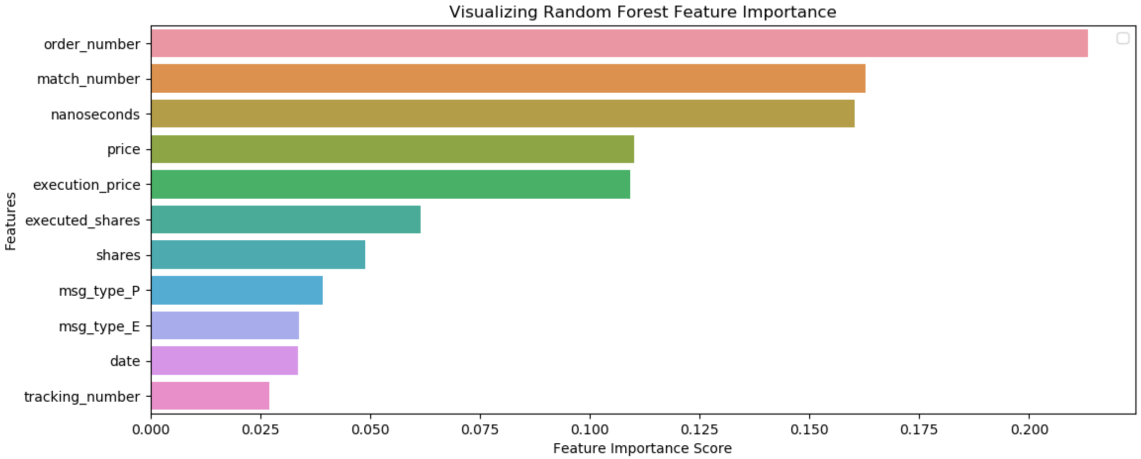


Model Accuracy Comparison



Conclusions

- Forests: low investment, high return
    - Beat Rosenthal -- and previous methods’ -- accuracies !
- B-LSTM implementation?
- How do we know? → timestamp/proxy-for-timestamp features in RF feature importance vec



Future Work

- More exploration with B-LSTM model, modifying implementation, parameters
- Transfer learning: predict onto TAQ data
- Match Quotes data with Trades data (better features?)
- Hierarchical B-LSTM to maximize info in Quotes data (multiple quotes per trade)