ECON136 Financial Markets and Modeling: Final Project

Aman Fatehpuria, Varsha Kishore, Anjaneya Malpani, Daniel Sonner, Nic Trieu

Project Description:

As Avi Thaker proposed in his in-class talk, machine learning models may be used to analyze micro-trends in stock market behavior. The goal of this project was to conduct a preliminary investigation of how machine learning may be applied to analyze daily stock movements.

We train three different models: Convolutional Neural Networks (CNN), Support Vector Machines (SVM), and Random Forests. We provide our methodology and insights in training these models. We also made a basic momentum model. Our results indicate that the machine learning models have the potential to outperform the momentum model.

Convolutional Neural Networks (CNN):

Convolutional neural networks (CNN) are currently the leading probabilistic model for classification with data locality assumptions. Since winning the ImageNet image classification contest in 2012 and beyond, they have been applied to a variety of computer vision tasks due to their effectiveness at capturing localized patterns. The main feature of the CNN is its combination of a deep neural net classification approach with the usage of convolutional filtering windows. These convolutional filters are trained to automatically extract relevant patterns from their window of raw data, making the CNN an efficient and powerful model for analyzing contiguous patterns. This insight is the reason why we chose the CNN as a model for capturing micro-trends in stock behavior.

Our convolutional neural network consists of two convolutional and pooling layers of 100 filters, and two dense classification layers of 200 and 100 neurons with the ReLU activation function. We implemented the model with TensorFlow. By stacking multiple convolutional filters, we allow the model to train filters on the outputs of its own filters, enabling the model to analyze patterns beyond the initial filter window size. Similarly, stacking multiple classification layers with the ReLU activation allows the model to perform non-linear classification mappings. Our choices for hyperparameters will be verified by the training plots shown in the results section.

CNN Results:

We obtained 5-year SPY data from IEX. The CNN was trained to predict whether the next day's SPY closing price would be higher or lower, given the following SPY features over the look-back period of time: 'change', 'changeOverTime', 'changePercent', 'close', 'high', 'low', 'open', 'vwap.' For a detailed explanation of these features, see the documentation on the IEX site, from which we obtained this dataset. Note that this feature set does not include any direct information for the day that we are predicting the closing price.

Our investigative goal was figuring out whether the look-back period really matters for model performance. We tried two different look-back periods: 8 days, and 32 days. The model hyperparameters were tuned such that they were able to achieve near 100% training accuracy only after a substantial training time (in an attempt to avoid overfitting).

Models were trained on the first half of the 5-year data, then tested on the second half of the data (i.e. the latter 2.5 years). The training results plots are given below.

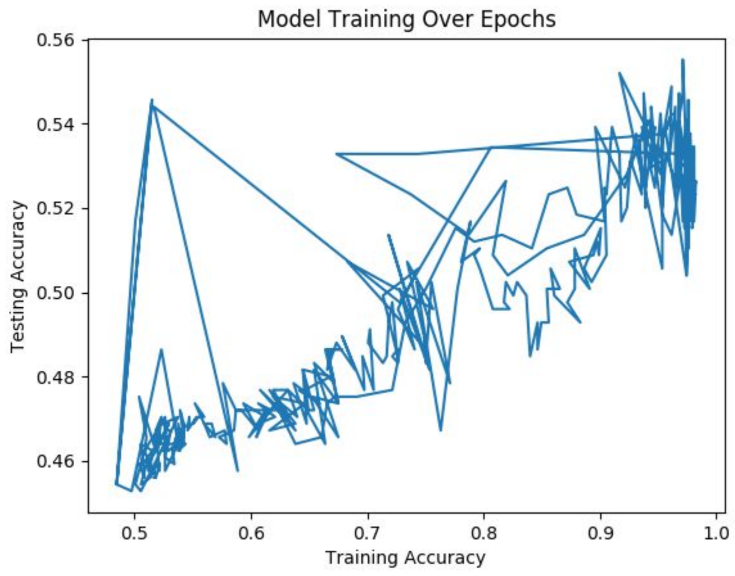


Figure 1: Timeline of 8-day-lookback model training over 500 epochs (iterations of training). Consecutive epochs are connected by the plotted line.

From Figure 1, we can see a clear correlation between training accuracy and testing accuracy. This is a promising result, as it indicates that what the model learns from past data may be applied years later. Future work, however, will need to include anomalous time periods such as the 2008 crash, etc.

Training accuracy approaches 100%, while testing accuracy reaches its limit around 56%. Learning was not entirely smooth, as seen by the various spikes and backpedaling in the graph. The early spikes in testing accuracy while still at low training accuracies may be attributed to the random initialization of the model's neuron weights. The variance in testing accuracy at the end is due to the model starting to overfit near 100% training accuracy.

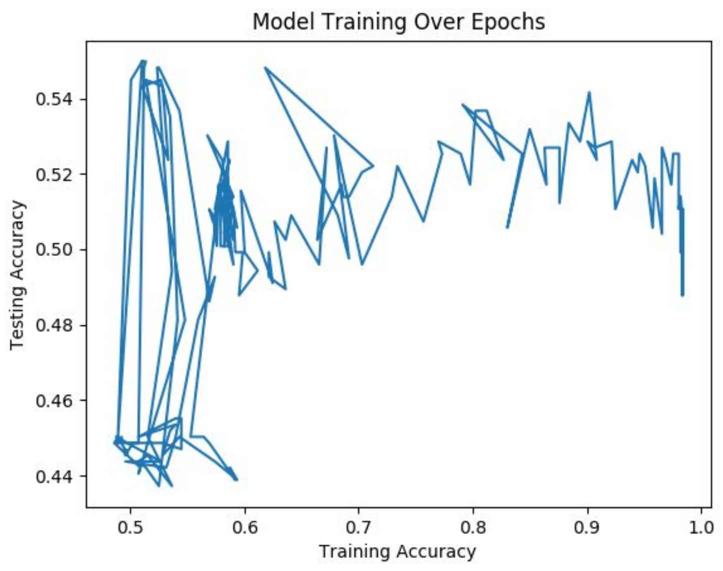


Figure 2: Timeline of 32-day-lookback model training over 200 epochs (iterations of training). Consecutive epochs are connected by the plotted line.

From Figure 2, we see less of a correlation between training accuracy and testing accuracy. Besides the drop in testing accuracy near end (where the model began to overfit on 100% training accuracy; we cut training here), there appears to still be a positive trend from 60% to 90% training accuracy. Interestingly, the model achieves even higher testing results in its earlier epochs; we attribute this to the randomized initial weighting, and the high volatility from epoch to epoch shows that the model is not yet stable at this time.

Most importantly, we note that the model never achieves as high a testing accuracy as the 8-day-lookback model, despite being given more information. Training accuracy approaches 100% again, but testing accuracy reaches its limit around 54%.

This indicates that the bulk of the necessary information was contained in the 8 previous days. With even more past data, the model simply becomes more prone to overfitting on noise that does not generalize to the testing set. This hypothesis is supported by the drastic testing accuracy drop that we see at the end.

Support Vector Machines:
Support Vector Machines (SVMs) are supervised learning algorithms that find a separating hyperplane in order to classify samples. In the past, using SVMs has led to good results in many different kinds of problems. So, we decided to use SVMs to predict the direction of stock prices.

We used the SVM implementation in the Scikit-learn python library. The first step was to select features that could be used to make predictions. The features we used with the SVMs were stock volume, close price, price change, boolean values indicating if price went up or down when compared to ‘x’ days ago (the ‘x’ values we used were 1,2,3,4,5,6), Relative Strength Index (RSI) and Stochastic Oscillator. The last two are indicators that we found by looking at past research[1]. The RSI is a popular momentum indicator which determines whether the stock is overbought or oversold. The Stochastic Oscillator is another indicator which follows the speed or the momentum of the price of the stock. After selecting features, we tuned the SVM classifier. Some of the hyperparameters we changed were cost, kernel and gamma. We found that using a a rbf kernel with default cost and gamma yielded the best results.

SVM Results
Below is a summary of results obtained when a SVM was used to predict direction of different stocks. For each stock, we obtained 5 year data from IEX and then split the data into test and train sets. The first 80 percent of the data was used to train the classifier and the next 20 percent was used for testing.

Stock	Train Accuracy	Test Accuracy
SPY	100%	58.23%
MSFT	100%	57.43%
INTC	100%	53.41%
GOOGL	100%	55.02%

As we can see from the table we get test accuracies of about 55%. However, the train accuracies are 100%. This indicates that the SVM is overfitting to the training data. In order to prevent this we need more data to train these models or we need to find less complex models.

Random Forests

Random forests are created by using multiple decision trees on the same dataset, and then 'averaging' out the results obtained across all trees.

We ran Random Forests to test whether we can predict the direction of movement correctly. The model was made in R with data being imported from IEX. We also added some features for volatility (ATR) and momentum (ADR) using the ta-lib library created by mrjbq7 (on github)[2]. The data was run using the packages Caret in R. Other supporting packages in R were added too. The results were as shown below:

---

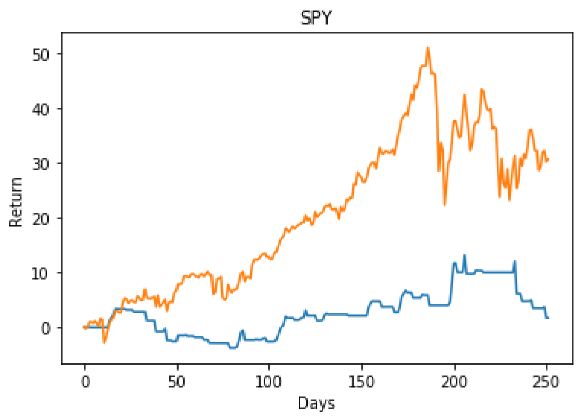
[1] Luckyson, et al. “Predicting the Direction of Stock Market Prices Using Random Forest.”*[1605.00003] Predicting the Direction of Stock Market Prices Using Random Forest*, 29 Apr. 2016, arxiv.org/abs/1605.00003.
[2] https://github.com/mrjbq7/ta-lib

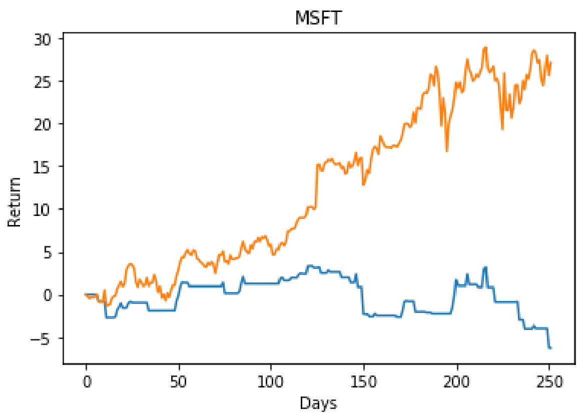
Stock	Test Accuracy
SPY	52.25%
MSFT	56.23%
INTC	55.70%
GOOGL	56.12%

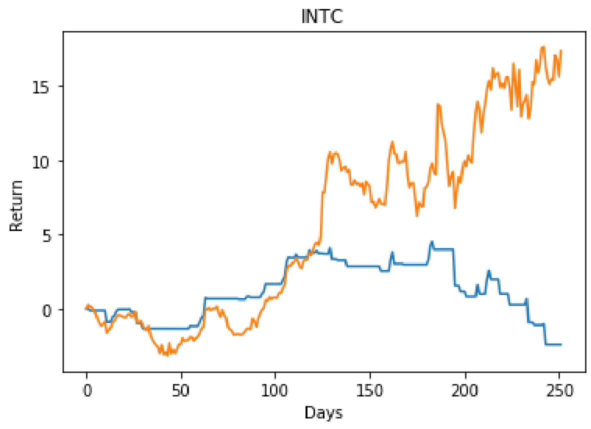
Momentum Model

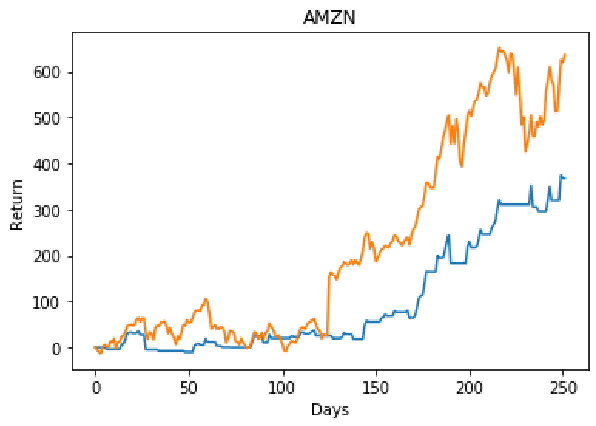
Remembering Avi’s advice that simplest strategies are often the best we additionally coded a very simple momentum strategy. The strategy was that if the stock went up for the previous 2 days we would buy the stock. When it becomes no longer true that the stock has gone up for the previous two days we exit the position (or said another way, we sell once the stock goes down at close). This strategy only considers closing prices. Testing on some arbitrarily selected stocks we see it generally underperformed buy and hold:

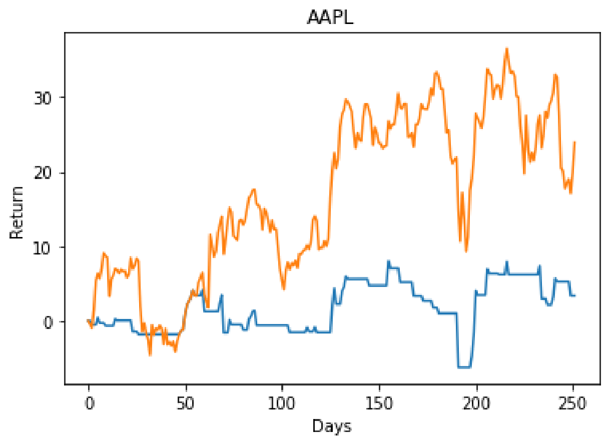
Ticker	Momentum ($)	Buy and Hold ($)
SPY	+1.63	+30.67
MSFT	-6.27	+27.12
INTC	-2.41	+17.36
AMZN	+367.87	+635.32
AAPL	+3.36	+23.89
AMD	+1.15	-0.81
TSLA	-21.01	-18.97

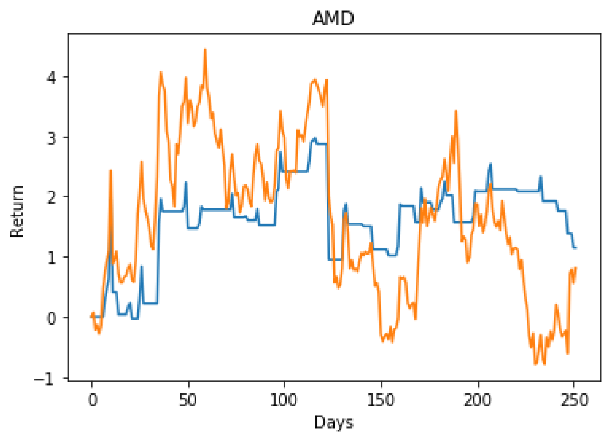

SPY
Return
Days
0
10
20
30
40
50
0
50
100
150
200
250

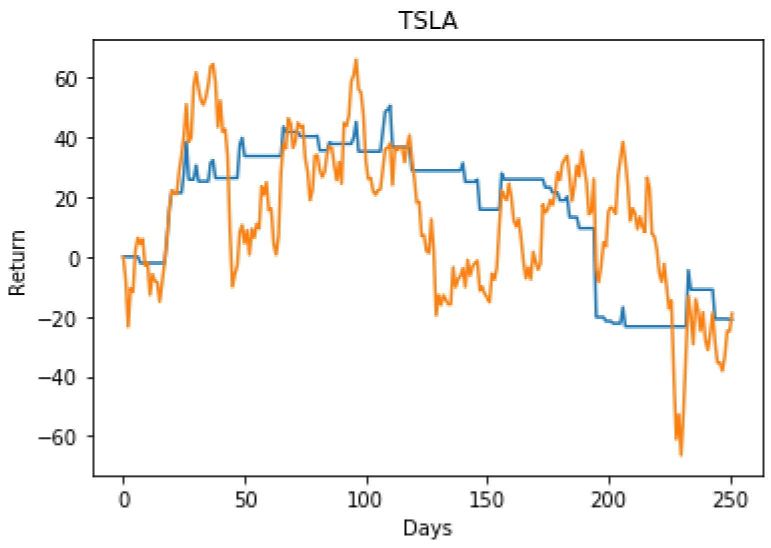

MSFT
Return
Days
−5
0
5
10
15
20
25
30
0
50
100
150
200
250


INTC
Return
Days
0
5
10
15
0
50
100
150
200
250


AMZN
Return
Days
0
100
200
300
400
500
600
0
50
100
150
200
250


AAPL
Return
Days
0
10
20
30
0
50
100
150
200
250


AMD
Return
Days
−1
0
1
2
3
4
0
50
100
150
200
250



We do understand that the backtesting of this strategy is somewhat simplistic. It assumes no slippage or commission/fees. It further assumes that we can buy at the closing price without any market impact, although this may be fairly realistic for the high volume stocks we chose assuming we are a small trader. Generally this strategy seems unsuccessful, underperforming for every stock except AMD.

Conclusion:

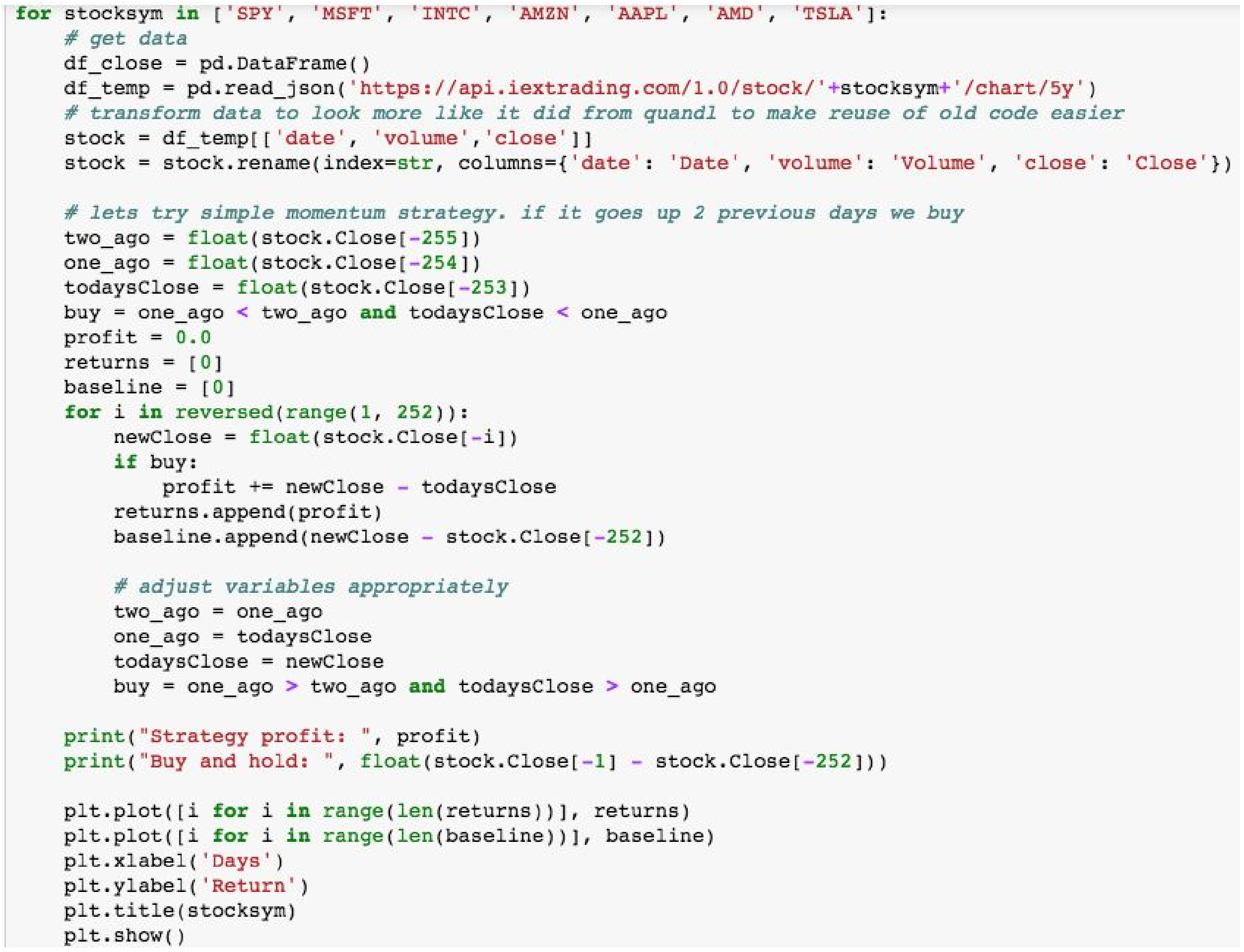
We tested CNNs, SVMs and Random Forests to analyze micro trends in stock behavior. The models gave us a testing accuracy of roughly 55%. If we consider 50% as a result of randomness, we have a 5% margin. Since we did not include trading fees and charges in these model the 5% margin may not really lead to us making money in the long run. We also tested a simple momentum model and realized that it also did not perform very well when testing against previous data.

The machine learning models can be improved by exploring other indicators and doing further backtesting. To know if we will actually make money it is necessary to incorporate trading fees into this model and recalculate the margin.

From our investigation it has become clear the machine learning models can be used to analyze micro-trends to some extent but it requires further investigation and the models need to be more generalizable (training accuracy of 100% and testing accuracy of ~55%). This investigation does not give a definitive answer to the question - can money be made by using these models? - since the models do not account for fees.

Link to code: https://github.com/NLTrieu/econ136_finalproject

Appendix A: Momentum code



```
for stocksym in ['SPY', 'MSFT', 'INTC', 'AMZN', 'AAPL', 'AMD', 'TSLA']:
    # get data
    df_close = pd.DataFrame()
    df_temp = pd.read_json('https://api.iextrading.com/1.0/stock/'+stocksym+'/chart/5y')
    # transform data to look more like it did from quandl to make reuse of old code easier
    stock = df_temp[['date', 'volume','close']]
    stock = stock.rename(index=str, columns={'date': 'Date', 'volume': 'Volume', 'close': 'Close'})

    # lets try simple momentum strategy. if it goes up 2 previous days we buy
    two_ago = float(stock.Close[-255])
    one_ago = float(stock.Close[-254])
    todaysClose = float(stock.Close[-253])
    buy = one_ago < two_ago and todaysClose < one_ago
    profit = 0.0
    returns = [0]
    baseline = [0]
    for i in reversed(range(1, 252)):
        newClose = float(stock.Close[-i])
        if buy:
            profit += newClose - todaysClose
        returns.append(profit)
        baseline.append(newClose - stock.Close[-252])

        # adjust variables appropriately
        two_ago = one_ago
        one_ago = todaysClose
        todaysClose = newClose
        buy = one_ago > two_ago and todaysClose > one_ago

    print("Strategy profit: ", profit)
    print("Buy and hold: ", float(stock.Close[-1] - stock.Close[-252]))

    plt.plot([i for i in range(len(returns))], returns)
    plt.plot([i for i in range(len(baseline))], baseline)
    plt.xlabel('Days')
    plt.ylabel('Return')
    plt.title(stocksym)
    plt.show()
```