https://github.com/rfinegan1/FinalProject-FE511
GitHub Repo: Code and Files present the process of the project

FINAL PROJECT

Ryan Finegan

FE-511 Intro to Bloomberg and Thomson Reuters

1. Abstract:

For this final project, I was inspired by the Carhart Four-Factor Model. This factor model seeks excess returns superior to the S&P 500 benchmark using factors discussed in the Fama and French Three Factor Model with the addition of a momentum factor. This project was not meant to emulate this study produced by Mark Carhart in 1997; instead this project was for my knowledge of the Bloomberg terminal and other finance data resources. In the process of this concluding project, I used the Bloomberg equity screener function, Python programming language, Yahoo Finance API, Scikit-Learn, and TensorFlow 2.x with Keras to create supervised machine learning classification and deep learning regression models observing securities relative to a low price to book ratio, small market capitalization, and a trailing twelve month positive return.

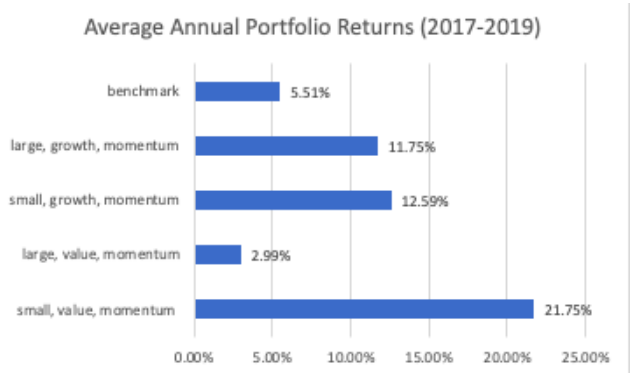
2. Inspiration:

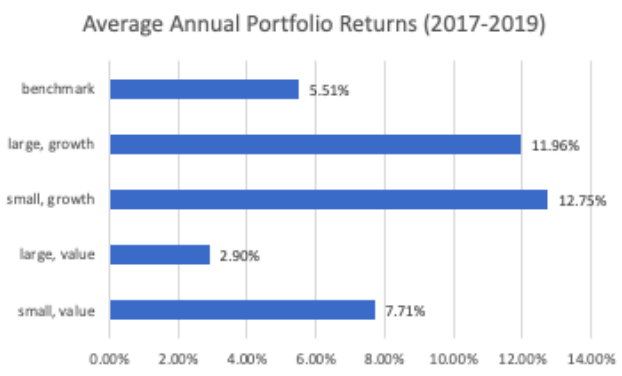
The Carhart Four-Factor Model has four factors which contain the size of firms, book-to-market values, a positive trailing period return, and excess returns on the market.

$$r_i = r_f + \beta_1 \cdot Mkt + \beta_2 \cdot HML + \beta_3 \cdot SMB + \beta_4 MOM + \epsilon$$

For my assignment, I looked into historical returns of stocks by observing a firm's size, price-to-book ratios, and trailing twelve month returns. For the High Minus Low factor, I instead looked at a value ratio of price-to-book where the low ratios were considered value stocks and high ratios above five were considered growth stocks. The Small Minus Big or size factor was identical such that a firm's market capitalization was the product of the total shares outstanding and the share price. The MOM, Up Minus Down, or momentum factor I experimented was the trailing twelve-month asset return. My portfolios were constructed using Bloomberg's EQS function using a one-year observation period of all active United States domiciled firms' market capitalizations, price-to-book ratios, and returns in that given twelve-month period. Portfolios were given an equal-weighted structure and the portfolio was never rebalanced over the two-year span where the individual stocks and portfolios' returns were recorded. The portfolios were created based on the 2016 yearly performance and fundamentals while the portfolio performance was based on the two proceeding years, 2017 to 2019. Each portfolio was equally weighted, and each portfolio contained stocks with similar factors and labeled as such. The individual portfolio returns were recorded to show further evidence of my reading into the Carhart Factors.

Performance Summary of Various Stocks (2017-2019):





3. Project:

The Nasdaq Composite Index is one of the three major market indices that comprises of thousands of stocks ranging in size, sector, and performance. This presents itself as an attractive index to randomly select securities from for experimental purposes.

3.1 Machine Learning Classification: K-Means Nearest Neighbor Overview:

Using the EQS function in Bloomberg, I extracted fundamental and performance data, with respect to the Carhart Four-Factor Model, of all the firms in the NASDAQ Composite as of December 31, 2013 and implemented this excel file into a Python script I developed to find the returns of each active security from 2014 to 2016. Two year returns over ten percent were considered excellent performing stocks while returns less than negative ten percent were labeled as poor performing. The good performing label was given to the stocks' experiencing returns greater than zero and less than 10 percent, and the bad performing label was given to the stocks with two year returns that were less than zero and greater than negative ten percent.

For this portion, a supervised classification technique called K-Means Nearest Neighbors was deployed to group securities together based on the size, value, and momentum factors to a label of excellent, good, bad, and poor performance. KNN considers that similar objects occur in or around the same vicinity. Therefore, the main underlying purpose of this algorithm is to assemble datapoints together grounded on the proximity of those datapoints.

3.2 Main Variables to K-Means Nearest Neighbors:

The main parts to this equation will be K, the Euclidian distance formula, training data, and testing data. The training data was 67% of the randomly selected 150 securities in the NASDAQ Composite Index possessing all the features (fundamental factors as of December 31, 2013) and labels (security return from 2014 - 2016). The testing data was 33% of the same randomly selected securities of the NASDAQ Composite possessing the features (fundamental factors as of December 31, 2013) and labels (security return from 2014 - 2016). The K variable is the number of nearest neighbors used to allocate a label to that specified datapoint. For example, you have many datapoints spread out on a graph. Your K is set at six; therefore, the specified datapoint is

labeled based on the closest six other datapoints based on, in this case, the Euclidean distance formula. The Euclidean distance formula is as follows:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space

q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

3.3 Sklearn Model Process:

I used a for loop to iterate through a variable allocated to K from one to fourteen. As K increased, the accuracy of the model fitted to the training data and applied to the testing features became increasingly higher before it stopped at 59% accuracy. I would have applied higher K values; however, this was not the main segment of my project and time was limited. Therefore, I did not allocate too much time as all I needed was enough accuracy to input the three factors to see what label the model recommended. The idea behind this was to forecast where a stock with certain factors would be grouped based on the fitted model. In its simplest form, this algorithm was meant to be a recommender system to output an expected class-based return grounded on a stock's inputted factors.

3.4 Classification Model Factor Return Prediction:

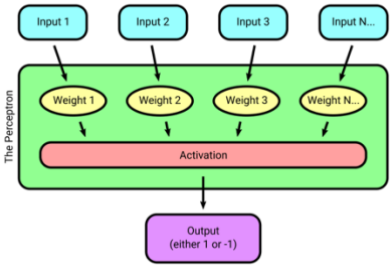
Once the model presented over fifty percent accuracy, I inputted a fictional security that displayed Carhart Factors. The firm's market capitalization was two billion US dollars, a value stock of 5 price-to-book ratio, and a momentum factor of a positive trailing twelve-month return. Once the model was accurate enough, this fictional security was grouped with the label of 4, or in this case, a high performing stock yielding a two year return greater than ten percent given the 2013 fundamentals and 2014 to 2016 performance time window. This prediction further explains that securities trading with a small market capitalization, low price-to-book ratio, and strong historical performance will, on average, produce higher returns than its counterparts.

3.5 Screener:

Using the requests library as well as other string manipulation and web scraping techniques, I scraped FINVIZ.com for potential stocks that possess a small market capitalization between 300 million and 2 billion US dollars, price-to-book ratio less than or equal to five, trailing twelve month positive return, and an average strong buy analyst recommendation of one. After running this script and removing duplicates, it yielded 20 securities to be used on a deep neural network using TensorFlow with Keras.

3.6 Deep Neural Networks Overview - The Perceptron:

The basics for a deep learning neural network include at its simplest form a perceptron. Input data is applied into the perceptron and from there weights are applied to each input. A summation occurs from the product of the weights and inputs with an additional bias variable. Then, an activation function is applied for non-linearity, in my model's case a Leaky Rectified Linear Unit. Finally, an output is generated, and in this case, it is a regression problem predicting the stock price one day in the future, so only one output is necessary. The perceptron is displayed below.



3.7 Deep Neural Networks - My Model:

Due to the variation of the securities collected, I knew a communal deep learning model would not be optimal. In contrast, it would be interesting to input a list of securities into a singular model to predict multiple adjusted close prices one day in the future and observe the outputs. Before constructing the model, I was aware of the volatile nature of small cap stocks. Relative to large cap stocks, securities with smaller market capitalizations have exponentially more room to grow coupled with the insufficient resources to deal with bearish sentiments during a downturn. Therefore, I believed a possible communal driver of these securities could be the CBOE Volatility Index. Furthermore, a commonality I observed was the link to the United States markets. The SPDR S&P 500 Trust ETF was my second driver. As features, the model observed the three day and nine day moving averages of both the VIX and SPY ETFs along with the three day and nine day moving averages of the select security. The label to be forecasted was the adjusted close price of the stock given one day in the future. Once the features and labels were split up, training, testing, and validation data was created by splitting the features and labels first into training and testing data with 80% allocated to training. The testing data was then split evenly between testing and validation data. The model had 5 layers: an input layer with 100 units, an output layer with one unit, and three hidden layers in the middle with 75, 50, and 25 units respectively. Each layer had a leaky rectified linear unit activation function for non-linearity and He initializer for the exploding or vanishing gradient problem as the activation outputs pass forward through the deep learning network. He initialization is said to be best for Leaky ReLU activation functions. The Leaky ReLU function formula is below.

$$\max(0.1x, x)$$

A mean squared error loss function was applied because that is most common for regression problems. The mean squared error loss is the sum of squared distances between the actual label and the predicted variables. A loss function is just a way of evaluating how well my model is performing. That being said, I also added the mean absolute percentage error as an added metric

to be used on the validation data because that metric being in a percentage helps me better. Also, observing the validation data's performance can help with identifying if the model is overfitting on the training data. Optimizers are used in deep learning to change attributes like weights and learning rates to reduce losses. The model used an Adam optimizer.

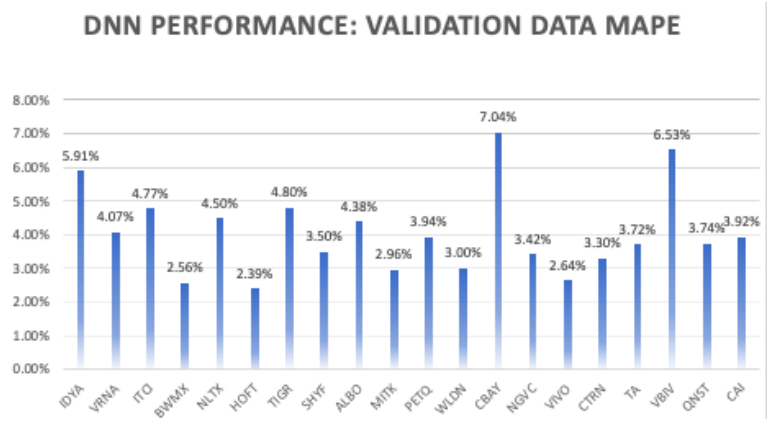
$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Finally, A min-max scaler was applied to normalize the feature data between datapoints one and zero.

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

3.8 Model Performance:

The model only had five epochs with the design to see if a few stocks indicated signs of strong performance of price prediction. All of the 20 stocks experienced a mean absolute percentage error below 10% with three in particular that worked very well.



If you reference the outputs of the code in the Jupyter Notebook at the link in the title page, tickers HOFT, VIVO, and CTRN performed extremely well as the validation set steadily experienced a diminishing loss meaning the neural network was learning the relationships between the features and labels well. Given these results, the model can be fine-tuned to add features that are relative to the drivers of that security's financial success or downfall. I will provide samples of these outputs below. The first number in the list is the mean absolute percentage error of the first epoch, and the last number in the list is the model's final epoch.

```
|||Chosen Security: HOFT|||
Value Mean Absolute Percentage Error ['11.96', '4.18', '2.39', '2.64', '2.61']

|||Chosen Security: VIVO|||
Value Mean Absolute Percentage Error ['10.45', '2.55', '2.64', '2.95', '2.66']

|||Chosen Security: CTRN|||
Value Mean Absolute Percentage Error ['8.71', '4.93', '4.21', '3.93', '3.30']
```

Conclusion:

Using Bloomberg and Python, the results of the portfolio's containing stocks with similar factors produced returns similar to expectations based on the Carhart Four Factor Model. The portfolio containing securities with a small size, low price to book ratio, and momentum performed superior returns to the benchmark, as well as the other portfolios comprised of stocks with different factors. The utilization of K Means Nearest Neighbors Classification additionally verified the back-testing results when a model was constructed to be fitted to historical factors and returns. When the fitted model was tasked to assign a return performance label to a stock consisting of HML, SMB, and UMD factors, the output was that of the highest classification of return performance. Furthermore, after web scraping for real-time stock factors, securities with factors in relation to the Carhart Four Factor Model were selected. A deep neural network was constructed to try to forecast the tomorrow stock price given features that included the specific stock's three- and nine-day moving averages and the VIX index and SPY exchange traded fund three- and nine-day moving averages. The neural network comprised of an input layer, three hidden layers, and an output layer to predict tomorrow's stock price. All twenty securities performed well in the general model posting a mean absolute percentage error of the validation dataset that was under ten percent; however, a select few stocks would benefit from adding other features that are more in line with their overall business model.

Future Work:

I would like to expand on this model in either of these two directions. For the first option, I would like to research the three firms that performed well in the deep neural network model. With the expectation to complement the model with two to three more features, I would like to discover drivers that are representative of the stock's financial performance, such as certain commodity prices or financial performance of other firms they rely on for their success. The other route of interest would combine my knowledge of Tweepy and extracting Twitter data and observe if a sentiment feature could improve the results of the model. Recently, sentiment data has been a growing interest in forecasting financial security data, and I believe it would complement the tick price moving average features well.