SENG-550 Project: Automated Investing

Authors: Zachary Kahn, James Peralta, Satyaki Ghosh, Anil Sood

Objective

Build a big data system that uses statistical analysis and machine learning (ML) to create a hotlist of companies to invest in. Specifically, we are looking to develop a system that uses quantamental analysis to achieve this goal. Quantamental analysis uses the fundamentals of a company along with technical information to better understand the value of a company.

Background

Having investment backgrounds, we have seen firsthand how difficult it can be to seamlessly integrate all the knowledge, ideas, and data available to make informed investment decisions.

Not only this, but keeping up to date on the status of previously made investments, putting in metrics to keep track of historical performance, and continually being able to improve one's investing strategies are extremely difficult to do without the help of computers. While there are many robo-investing services available, everyone's investing principles and preferences are different. For this reason, we are looking to design a system that is more customizable to our needs that leverages big data analytics systems, such as Hadoop and Spark, to combine our investing knowledge with data with the goal of making it easy to track, improve, and automate our investments.

High Level Design

The system built uses data from Thomson Reuters Eikon API. All data is stored on BigQuery, which runs within Google Cloud Platform (GCP). The data collected is raw data which is stored in BigQuery and then gets cleaned and preprocessed using Cloud Dataproc, which allows us to use Spark. From here, model specific feature engineering is completed and the results are stored in BigQuery or Cloud Storage. Model features are then fed into various machine learning models, which is denoted as the ML Engine. The ML Engine utilizes Spark for distributed computation and Scikit-learn, Keras, Tensorflow, Talos, Elephas and MLlib for machine learning. The validated models results are stored in BigQuery for easy processing and visualization while the model weights themselves are stored in Cloud Storage. The model weights can be used for inference in the production pipeline (future work for this project), which will take in preprocessed data from the Eikon API and yield the final hotlist of companies selected along with various metrics about what the model predicts for each of the selected companies. An overview of the system design and ML Engine design are depicted in Figures 1 and 2.

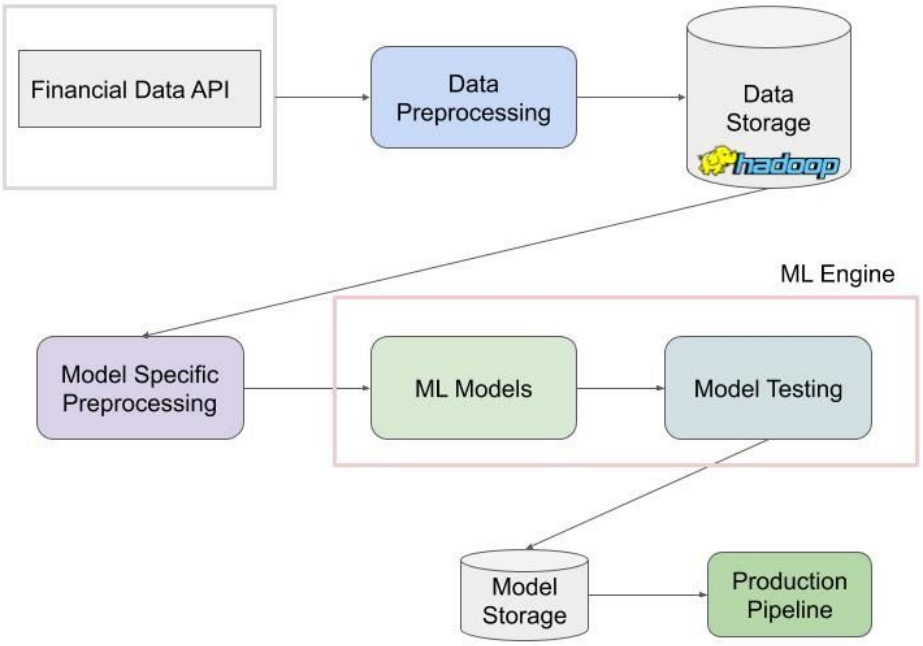

Figure 1: High level system design.

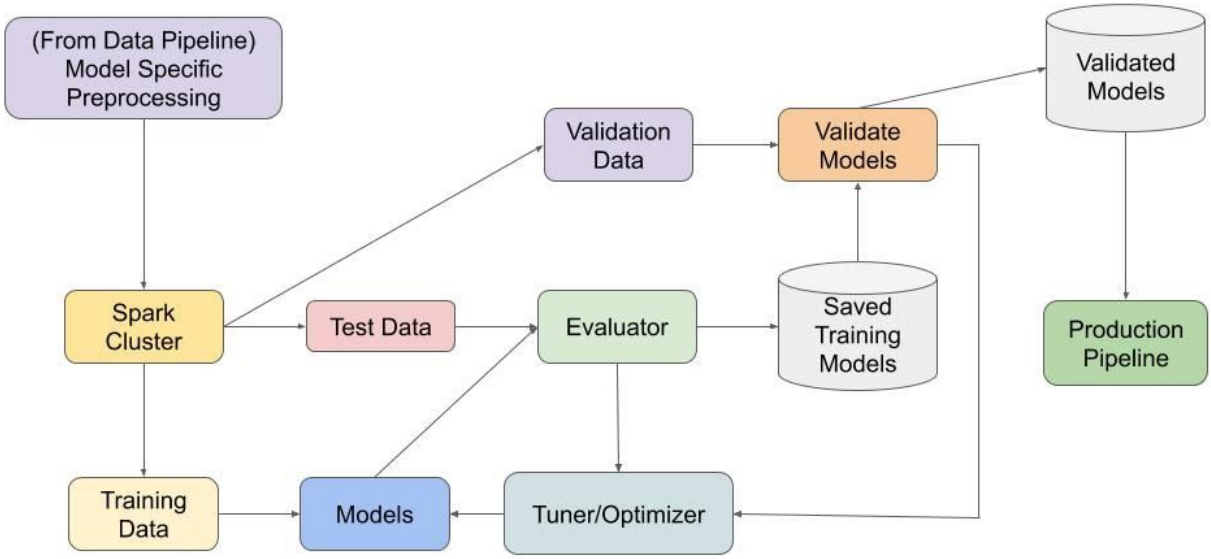

Figure 2: ML Engine.

System Inputs

Performing quantamental analysis using machine learning requires thoughtful feature selection and feature engineering. Based on our finance background, we chose the set of features to input into the models. There were two basic components for each model, the fundamentals and technicals.

The Fundamentals

The fundamentals are data that is usually available in a company's quarterly and annual reports. For the models build, the following (and more) fundamental data was included:

- Earnings per Share (EPS)
- Book Value per Share (BVPS)
- Revenue per Share (RPS)
- Liabilities
- Current Assets/Liabilities
- Gross and Profit Margins
- Historical Earnings and Revenue Growth
- Assets
- Earnings Before Interest, Taxes, Depreciation and Amortization (EBIDTA)
- Debt-Equity ratio

The Technicals

There are many technicals to explore, but there are a few core features that were included in each model:

- Daily Close Price
- Volume of Shares Exchanged Per Day

System Configuration

A lot of time was spent on researching the best combination of tools we should use on the GCP platform to optimize our build pipelines, big data jobs and parallelizing the machine learning models. Our goal was to have a fully automated job which performs all these tasks in different stages. This job would be responsible for polling our GitHub repository, dispatching the relevant data preprocessing jobs and applying machine learning algorithms on a Cloud Dataproc cluster and finally saving all the outputs. We achieved this by creating a Cloud Build pipeline which was attached to our GitHub repository and it ran our defined jobs every time there were changes made to the master branch. Our Cloud Build pipeline can be found at the root of the repository. We also created a few bash scripts that were designed to create, run and delete the Cloud Dataproc clusters and those can be found in the gcp_scripts folder in the repository.

Due to our heavy data processing and complex neural networks requirements we were not able to use the default Dataproc configurations. By using the default configuration, some of our jobs

were taking more than 12 hours to complete and thus, it was too time consuming to optimize our models. Therefore, we conducted some experiments to evaluate the performance of the clusters under various workloads and different configurations. For our use-case, we settled on a 1 master and 4 worker node design. Each node in the cluster was configured to use the n1-highmem-16 machine type which had 16vCPU units and 104GB of memory. We also added 5 nvidia-tesla-k80 preemptible GPU accelerators to each worker node in the cluster, since it noticeably improved most of the machine learning tasks. Further optimizations were carried out at the specific task granularity instead of throughout the entire cluster.

Data Collection

Historical fundamental and technical price data for all companies on the NASDAQ from 1990 to present day was collected and uploaded to GCP. Data was pulled for over 3000 companies, and various preprocessing was completed. The scripts that performed the data collection are located in the Grab_Data directory of the repository.

Exploratory Analysis

We decided to do some exploratory analysis on the fundamental data using some financial metrics to see if we could discover any interesting facts or patterns. We used metrics from Benjamin Graham who was a British-born American investor, economist, and professor. He was widely known as the "father of value investing" [4]. He defines the following metrics which can be used as a starting point to analyze and evaluate the financial health of companies. The companies which met most of these metrics were considered valuable investments or could at least be considered viable investment options for further in depth analysis. The metrics used were the following:

1. Sales above 500 million
2. Current assets is twice or more the current liabilities
3. Working capital is more than the long term debt
4. Positive Earnings for the past 10 years (used last 5 years)
5. Div Payments for past 20 year (used last 5 years)
6. Earnings increased by 1/3rd in past 10 years
7. Current Price is less than or equal to 15 * earnings per share
8. Current value is less than or equal to 1.5 times the book value

Using PySpark, we created a table for company ticker names and eight metrics where the metrics are one hot encoded. An interesting finding was that only one company met all 8 metrics, the company was Hurco Industries Ltd. As expected, there were more and more companies as we go down the count of how many metrics these companies met. These metrics provide a helpful filter for deciding which companies to further analyze. Tying this filtering into the ML Engine is an area of future work. We also clustered these companies using K-means clustering to see if there were any companies that were related based on their Ben Graham metric score. As this data was encoded into either 0 or 1 and there were 8 columns, there can

be a maximum of 2^8 = 256 clusters. Therefore, as expected, as we increased the number of clusters we got better and better silhouette value, which maxed to 1 around 140 clusters. According to Wikipedia, “The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters” [5]. However, that many clusters may prove useless when analysing, so we decided to create some smaller clusters. For future work, we can analyze these clusters to see if there are some interesting facts in them.

Data Cleaning and Joining

Our data consisted of two types: technicals and fundamentals. The technical data contained everyday metrics (ex. stock price), and the fundamental data contained the quarterly information for each company (ex. earnings). The collected data had many missing and duplicate values and needed to be preprocessed before it could be used. Data cleaning was performed by a python spark script to remove empty and/or redundant rows and columns. Some of our analysis needed fundamental and technical data combined together, so we decided to create a python spark script to join the two datasets into a new combined dataset. We started off with the technical dataset containing 24,847,880 rows and 12 columns and the fundamental data set containing 195,636 and 40 columns. Some of the metrics collected were from the datasets displayed below:

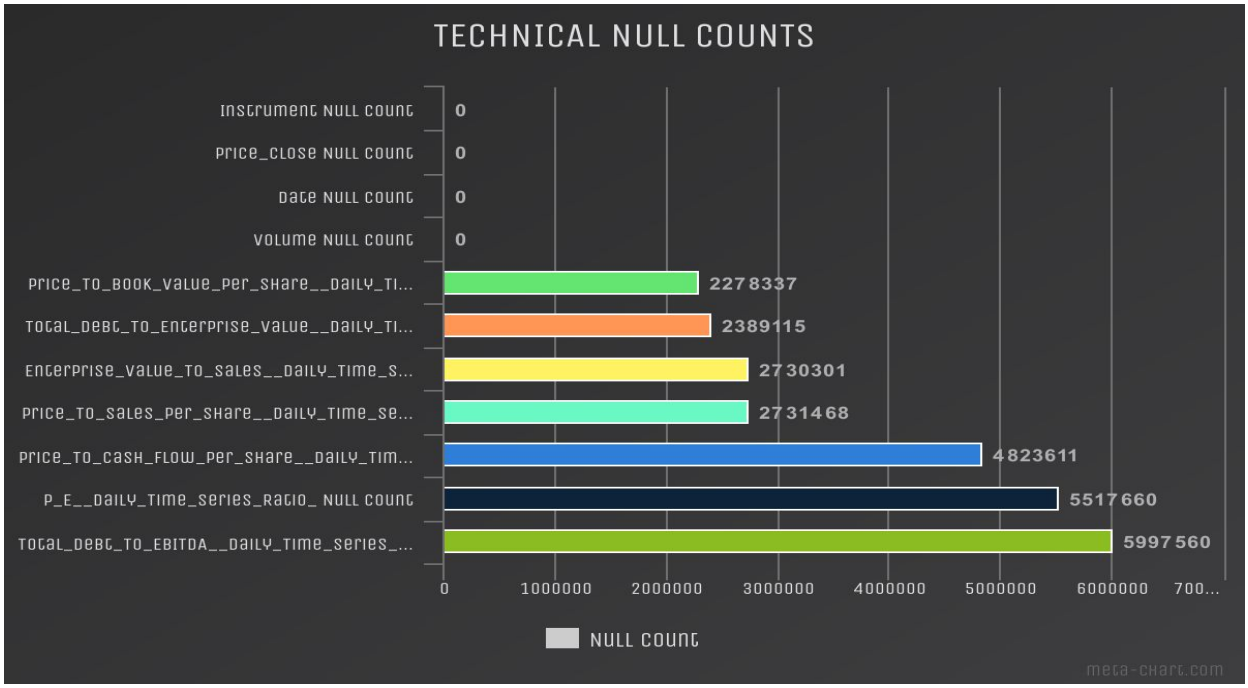

Figure 3: Technical Data Null Counts.

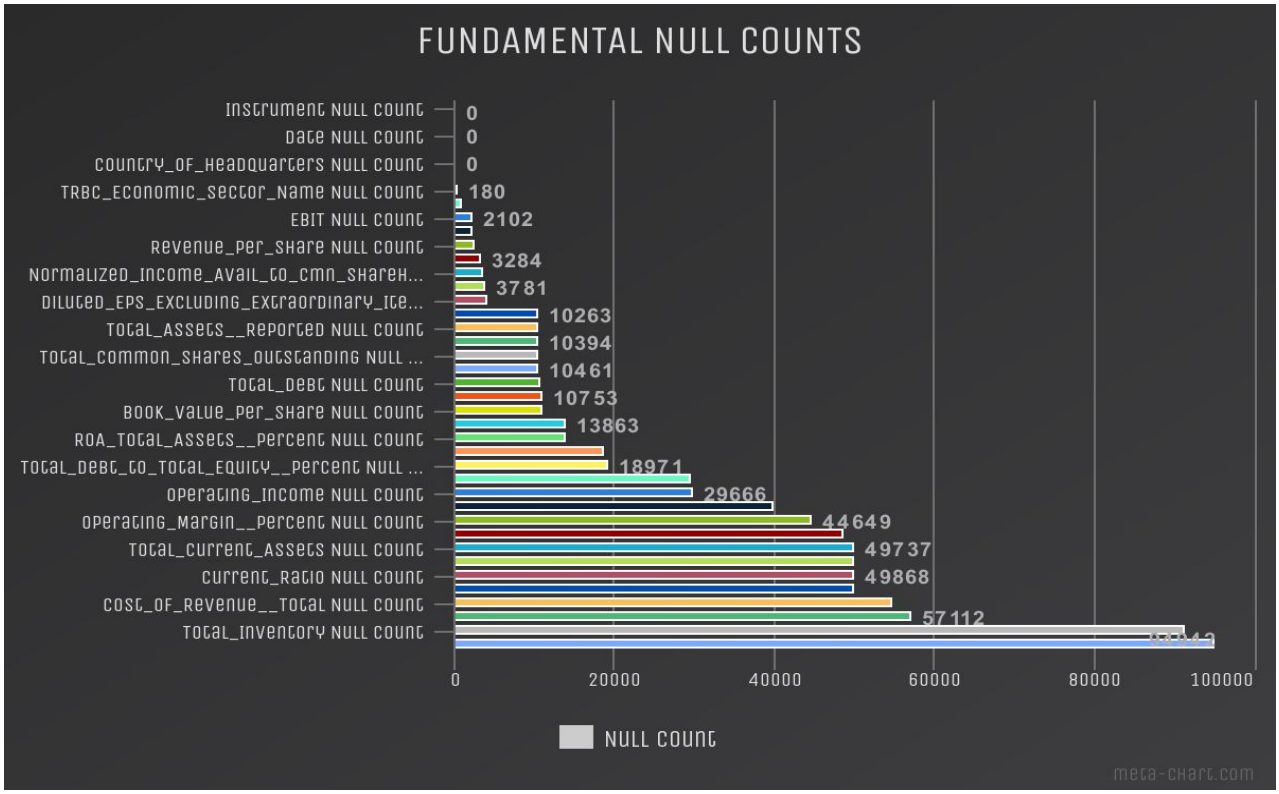

Figure 4: Fundamental Data Null Counts.

In the technical dataset, there was 21,096,796 or 84.9% rows missing values in one or more columns and 151,423 or 77.4% rows for our fundamental data. After cleaning up the two datasets we were left with 10,908,118 rows for technicals and 175,050 rows for fundamentals. We had to remove about 56.10% of the rows from the technical set and 10.52% from the fundamental set. These data points were missing crucial features that would be necessary for our machine learning algorithms. We believe we still had enough data points to accurately represent the stock market. By merging the two datasets we generated a table that contained 10,059,333 rows and 51 columns. This is the dataset that was used by most of the models to predict the stock prices. The PySpark script that performed all these tasks can be found in the data_cleaning directory of the repository.

Feature Engineering

The features in all the datasets we're displayed in three major formats: percent, absolute values, and per share basis. To meaningfully compare metrics between different companies, either ratios were made between all metrics or each metric was normalized. Some models explored (described later) were better suited to have ratio data as the input, while others were better suited for normalized data. We split the data into training (~60% of data), validation (~20% of data), and test sets (~20% of data) when training our models. The feature engineering for each model is discussed in the Machine Learning Models section.

Parallelizing Model Training

To train our machine learning models at scale, we used Apache Spark's ***MLlib*** and ***Elephas*** [2] for distributed deep learning. MLlib was able to provide us with the basic algorithms such as K-means clustering and Linear Regression. For the more advanced techniques, we implemented the models using ***Keras*** and used Elephas as a wrapper when fitting the model to distribute the training.

Machine Learning Models

The goal of the models built was to determine which companies are undervalued. Understanding what constitutes value in a public company is part of the challenge of building models for analyzing financial data. The idea our team developed to determine the top list of undervalued companies was to make various supervised and unsupervised models that determine which companies are undervalued, and then combine the outputs of these models in meaningful ways. The combining of model results was done using some models as filters to others and then ensuring that two different models still reached similar conclusions for any given company in the hotlist. A description of each model and where they are used in the machine learning pipeline is described next.

K-means Clustering on Fundamental Data

Before proceeding with any supervised learning, we first wanted to understand if there was a way we could bin companies into growth, undervalued and not good investment bins. The goal was that this would allow us to then cross reference any model result from one of the supervised learning algorithms and ensure it made sense based on the bin a particular company was in. The clustering was performed only on the fundamental data, meaning we were trying to group companies that had similar important fundamental metrics. Principal Component Analysis (PCA) was used to determine the 5 most important fundamental metrics, and these metrics were used to perform the clustering. Once every company had been binned into a group, the next question was to determine what each of these bins meant. This was done by looking at the average year of year growth of each company on a quarterly basis and then joining these results with the clustering results. This enabled us to look at a given company over time, their growth year over year (for a particular year), and what bin that growth and set of fundamental data corresponded with. From there, the data was aggregated and the average growth rates of each bin was determined. The bin with the highest average growth rate was what our team deemed undervalued. This is because companies in this bin had the best combination of fundamentals that led the highest growth. The results are summarized in following table.

Row	Cluster	Average Year Over Year Return (%)
1	0	13.60
2	1	13.88
3	2	17.23
4	3	35.52
5	4	14.67

Figure 5: Clustering results on fundamental data for all companies on the NASDAQ.

Two very interesting results arose from the K-means clustering analysis. The first was that there was one bin that had a significantly higher year of year return (YoYR), bin 3 (the “undervalued” bin), with and average YoYR of 35.52%. The other interesting result obtained was that we could see how the same company changed bins over time. Bins 0-2 corresponding with startup-like and relatively young companies. These companies either tended to grow a lot or fail. Bin 4 corresponded with more mature companies, companies that were already proven investments but had already experienced most of their growth. Bin 3 was the sweet spot for most companies in their history, the point when they experienced the most growth.

The results obtained from the clustering were very useful and will be used as an important filter in the pipeline (future work). Any company that is deemed undervalued and has a large projected upside will be cross referenced with its corresponding bin. If the company is in the undervalued bin (bin 3), then it passed through the filter. If it is not, it could be further analyzed, but in most cases will not make it to the hotlist.

Linear Regression

For this experiment we wanted to use a linear regression model to predict the stock price of a company. The train and test data was shuffled which removed any time-series relationships because we wanted the model to learn which features contributed the most to the stock price instead of learning curves in the dataset. The first experiment was to train the linear regression model on all companies from 1999-2017 and predict the 2018-2019 stock prices for 9 companies in the tech sector including Facebook, Google, and Apple (full list in Figure 6). The second experiment was to train the linear regression model on just the companies in the technology sector from 1999-2017 and predict the same 9 companies stock prices from 2018-2019. The only tunable parameters we’re the amount of regularization and the number of iterations. We found the linear regression would converge after 300 iterations of the algorithm. After our experiments we concluded that a linear regression model trained on the sector the company was in would produce the best results (average decrease of 1,714 MSE between the 9 companies).

Company	MSE of year over year baseline	MSE after training LR on all companies	MSE after training LR on tech sector
Facebook	133	1416	384
Google	2687	9548	1153
Apple	213	4028	1515
Netflix	1084	294	617
Microsoft	1444	3340	1814
PayPal	86	1861	255
AMD	30	276	252
Intel	13	1176	486
Nvidia	929	370	401

Figure 6: Mean Squared Error of the 9 companies used as our test set.

Dense Neural Network

Due to the nature of Neural Networks, these experiments required more hyperparameter tuning. We approached this by first doing some manual parameter tuning with the amount of layers, size of each layer, learning rate, batch size, and dropout. We started to notice how each parameter affected the mean squared error. For example, having more than 1 hidden layer seemed to produce terrible mean squared errors on the test set (up to 9.5×10^{12} when using 3 hidden layers of 128 nodes). Having 1 hidden layer produced the best results but to empirically test this, we used ***Talos*** to generate and train 729 permutations of DNNs. To use Talos, we had to create a dictionary of all of the hyper parameters we wanted to try (Figure 7.).

```
dnn_params = {'lr': [0.01, 0.05, 1],
              'first_neuron': [64, 128, 256],
              'activation': ['relu'],
              'hidden_layers': [1, 2, 3],
              'batch_size': [32, 64, 128],
              'epochs': [10, 15, 25],
              'dropout': [0.1, 0.2, 0.5],
              'shapes': ['brick', 'funnel']}
```

Figure 7: Talos Hyperparameters.

We then passed this dictionary into the ***Talos Scan function*** to generate all permutations of DNNs and train them. Due to the resource constraints ($), we ran this function on only Facebook data. When it was finished executing, it returned to us a pandas dataframe of the results for all permutations.

round_epochs	val_loss	val_mean_squared_error	loss	mean_squared_error	activation	batch_size	dropout	epochs	first_neuron	hidden_layers	lr	shapes
25	1.1302778182310216	1.1302778182310216	8.442082744794705	8.442082744794705	relu	32	0.01	25	64	1	0.01	brick
25	1.181704369432786	1.181704369432786	4.22076451285215	4.22076451285215	relu	64	0.01	25	256	1	0.01	brick
25	1.406678135254804	1.406678135254804	8.216965637341758	8.216965637341758	relu	32	0.01	25	128	1	0.01	brick

Figure 8: Top 3 models out of 729.

round_epochs	val_loss	val_mean_squared_error	loss	mean_squared_error	activation	batch_size	dropout	epochs	first_neuron	hidden_layers	lr	shapes
15	9478980969662.3438	9478980969662.3438	646157247038412 80.0	646157247038412 80.0	relu	64	0.05	15	128	3	0.05	brick
15	19446277653906.371	19446277653906.371	312741900352510.94	312741900352510.94	relu	64	0.05	15	256	2	0.05	brick
10	1.98010352528807e+16	1.98010352528807e+16	27603018404631492.0	27603018404631492.0	relu	64	0.01	10	64	2	1.0	brick

Figure 9: Bottom 3 models out of 729.

We noticed that the top 3 models (see Figure 8) had the same number of hidden layers (1 hidden layer), dropout (0.01), learning rate (0.01), and ran for the same number of epochs (25 epochs). The size of the batches and neurons in each layer didn't matter much with each model achieving around 1 for validation mean squared error. The worst models had more than 1 hidden layer and their learning rates were at least 5x higher than our learning rates from our top 3. This was also apparent in our manual testing, where we were noticing that the learning rates were very important in the performance of our model. The best model for predicting stock price is shown in Figure 10, which achieved a mean squared error of 1.13.

round_epochs	val_loss	val_mean_squared_error	loss	mean_squared_error	activation	batch_size	dropout	epochs	first_neuron	hidden_layers	lr	shapes
25	1.1302778182310216	1.1302778182310216	8.442082744794705	8.442082744794705	relu	32	0.01	25	64	1	0.01	brick

Figure 10: Top Performing DNN.

The last step was to take this model configuration and perform the same experiment we did for the linear regressor. Figure 11 shows how each DNN compared to our baseline year over year model. The simple baseline model was able to outperform the DNN. This is due to the part that running on too many epochs was giving us NaN values for the loss function. With more time and research we can possibly get the MSE to outperform our baseline.

Company	MSE of year over year baseline	MSE after training DNN on tech sector
Facebook	133	316
Google	2687	934,672
Apple	213	617

Netflix	1084	24,966
Microsoft	1444	4,352
PayPal	86	7,092
AMD	30	23,735
Intel	13	15,924
Nvidia	929	2,852

Figure 11: Mean Squared Error of the 9 companies used as our test set.

Encoder-Decoder Long-Short Term Memory (LSTM) Network

Stock market prices change on a daily basis. Identifying patterns in this type of time series data is challenging, however, one type of neural network that is particularly good at understanding temporal dependencies of inputs and how they map to outputs are LSTMs. LSTMs are a type of Recurrent Neural Network (RNN). For a more comprehensive overview of LSTMs please refer to this article [2]. The question posed to the LSTM, based on how the input was structured, was given matrix of share prices over a given number of days (the window) as well as trading volumes and other fundamentals about a company for the input dates, can the LSTM predict a number of future daily stock prices. What was different about the analysis done compared to other LSTM stock market models was that both technical and fundamental data was being fed into the model to predict prices. We were also training on a larger window of historic data and trying to predict farther into the future. The goal of this model was to use quantamental analysis to understand if a company appeared undervalued and had a potential to breakout over the next 3 to 6 months. The LSTM was trained on various time frames, but most used data over the past 10 years. The input to the LSTM model was crucial to obtain meaningful results, and is described next.

A dataset for a particular company was split into a given input window size (w_{in}). This represented the number of days that was fed into the LSTM to predict a given output window size (w_{out}). What is important to note is that w_{in} does not have to equal w_{out}, which was made possible using the Encoder-Decoder pattern within the LSTM. This was critical because often it is beneficial to have a much larger input window and then only predict a small amount into the future. This is because the farther the prediction is made the less accurate it tends to become. What is interesting is that, with investing, longer time horizons are usually easy to predict. This model was trying to capture short to mid range breakouts (3-6 months) while some of the other models were aimed at finding companies that were undervalued and had lots of potential over the long term (5-10 years). Combining these results lead to exactly the type of companies that we were looking to invest in. The Encoder-Decoder LSTM architecture chosen can be seen in the figure below.

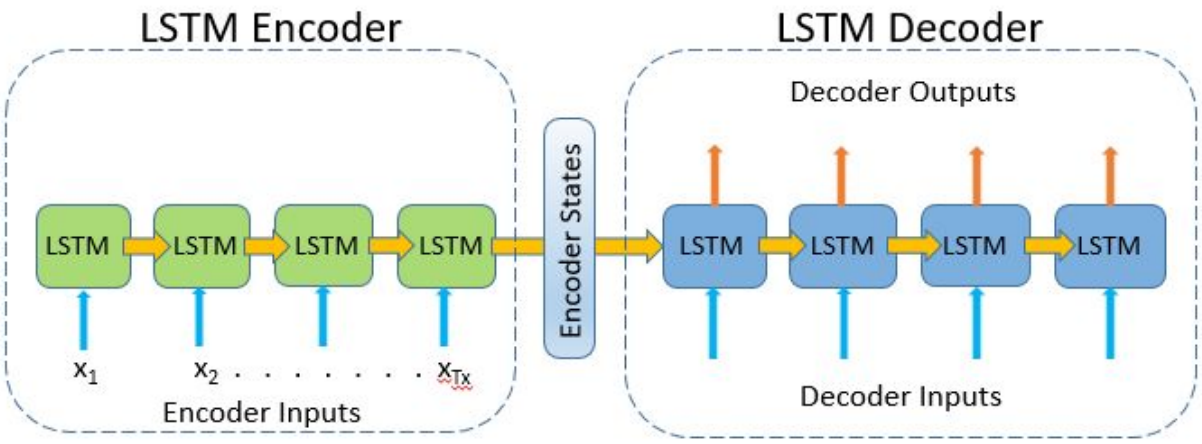

Figure 12: Encoder-Decoder LSTM Architecture [3].

To train the model, the features for a given window, say day *i* to day $i+w_{in}$ (including price) were passed to the LSTM and the output label was the price from day $i+w_{in}+1$ to day $i+w_{in}+1+w_{out}$. This type of window was passed in for every single day of the training data.

There were various issues and optimizations that were performed. The major issue was the loss function was reporting a Nan value after a certain number of epochs. This was most likely due to the vanishing (and exploding) gradient problem that can plague LSTMs. To combat this (as well as overfitting) a regularizer was added to the loss function and two dropout layers were added to the network. However, there were still instances where the loss function became Nan. This is a previously reported issue for LSTMs using Keras and future work is devoted to investing this error further. Even with this issue, models were still made for many companies. Due to resource constraints though, only a small fraction of the companies contained in our database were modelled using the LSTM.

A subset of the results for the models are summarized in Figure 13.

Company	Training Percent Error (%)	Validation Percent Error (%)	Naive Model Percent Error (%)	Training RMSE	Test RMSE	Predicted Future Price (6-12 months)	Potential Upside (%)
Tesla	17.4	13.3	33.1	8121.7	5685.4	394.4	52.5
Facebook	17.6	17.4	24.6	3638.0	5466.8	190.8	4.7
PayPal	47.1	6.0	34.8	2001.6	379.8	75.7	-30.3

Google	86.1	NaN	20.7	102338.1	NaN	1389.2	19.4
Nvidia	38.2	630.8	58.1	2396.6	61102.2	138.0	-16.6

Figure 13: LSTM Results.

An LSTM model was made for each company, and the results above were obtained after 5 epochs of training. As future work, we plan to explore increasing the number of epochs for training but decided not to at this point because of resource constraints and possible exploding gradient issues. As a sanity check, the percent error for each model (both the test and validation) was compared with that of a naive model. The naive model estimated the daily price for a given year as the average price of the previous year multiplied by the company's average percent growth rate. The models for Tesla and Facebook yielded significantly better results than the naive model. However, the other models performed worse than the naive model.

Overall, the results were promising and could be improved by training with more epochs, more hyperparameter tuning, and optimizing the best input and output window sizes.

Conclusion

This project consisted of using big data technologies and machine learning to build a program that allowed us to identify undervalued companies. We were successfully able to build a system that collected data, processed it, extracted important features, built models, and made predictions. Numerous technologies were used to accomplish what was done, most of which was provided by GCP and various machine learning libraries. Much progress has been made in the development of a fully automated machine learning investment system, but there are still lots of areas for improvement and future work.

Some of these areas include:

- Using the K-means clustering results as filters for which companies to model.
- Investigating Nan loss function results with Keras LSTMs.
- Further hyperparameter optimization.
- Building additional models.
- Making a dashboard and frontend that displays all results in a central location so that people can easily make decisions on the results.

However, even with all that remains, our team learned a tremendous amount about machine learning and how to build a machine learning pipeline at scale. Many of the models built show promising results, and we are excited to see how they perform in the future.

References

1. Elephas: https://github.com/maxpumperla/elephas
2. Understanding LSTM Networks. (n.d.). Retrieved December 3, 2019, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.
3. Battula, M. (2019, March 4). Time series forecasting with deep stacked unidirectional and bidirectional LSTMs. Retrieved December 3, 2019, from https://towardsdatascience.com/time-series-forecasting-with-deep-stacked-unidirectional-and-bidirectional-lstms-de7c099bd918.
4. Benjamin Graham: https://en.wikipedia.org/wiki/Benjamin_Graham
5. Silhouette Score: https://en.wikipedia.org/wiki/Silhouette_(clustering)