How accurately can we predict market cap values for S&P 500 companies using historical data?

Molly Harrison

Introduction

One of the key metrics investors use to determine the value of a company's equity (and therefore its stock value) is its market capitalization or *market cap*. There are many approaches and considerations to analyze when making an investment, and more and more data science methods have begun to make an appearance in investment determination. In this report we are going to look at how accurately we can predict market cap values using historical data.

Typically, investors use methods such as *discounted cash flows* (DCF) and other models to determine the equity value of a company. The equity value and market cap for a company are different, as market cap is the valuation of the company through the lens of financial markets (share price multiplied by outstanding shares), while equity valuation comes from its financial conditions (assets and liabilities). Investors aim to find the equity value of a company in order to make sound and profitable investment choices. Even though these two financial concepts are different, they both represent the valuation of a company's equity and will be similar (equity value is the market cap with the additional consideration of other equity information). For our purposes, in this report, a company's value will be referred to as one concept, and we will be predicting market cap as a proxy for how the public markets perceive equity value, acknowledging that real equity value includes additional factors.

Methods to calculate equity value such as discounted cash flows involve forecasting financial data and estimating growth rates. They rely on historical data as well as research and knowledge of the company so the resulting estimated equity value is a compilation of real data and the investor's assumptions.

In this report we will investigate whether or not we can use supervised learning algorithms, focusing on neural networks, to predict market cap value (specifically looking at S&P 500 companies, which are the 500 biggest U.S. based companies). We will examine how accurate our predictions are considering we are removing the methodical calculations and investor assumptions that lead to an accurate equity value analysis.

The Data

```
library(tidyverse)
library(caret)
library(neuralnet)
library(readxl)
library(mice)

financial_data <- read_xlsx("Calcbench Data Query Export.xlsx")

financial_data[financial_data == 0] <- NA

clean_data <- financial_data[!is.na(financial_data$`MarketCapAtEndOfPeriod 2023`),]

colnames(clean_data) <- make.names(colnames(clean_data))

imputed_data <- mice(clean_data,
                         method = "rf",
                         seed = 1,
                         maxit = 5)

imputed_data <- complete(imputed_data)

imputed_data <- imputed_data |>
  na.omit()
```

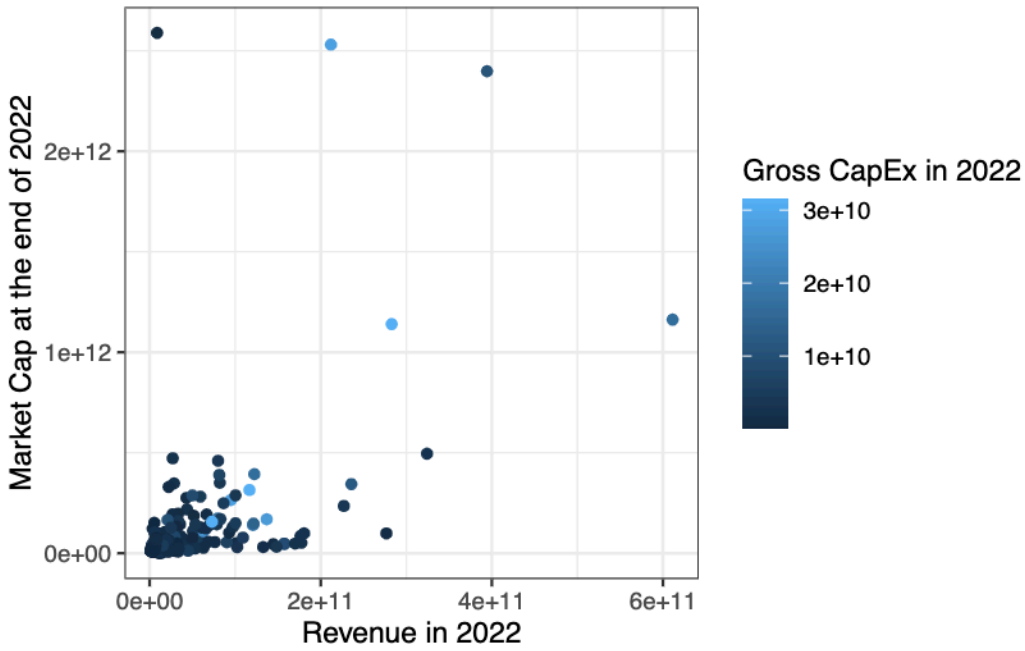
The data we are using is from Calcbench's bulk data interface (see references below) and exported into excel. It contains key financial data and market cap value (at the end of the year) for each S&P 500 company from 2018 to 2023.

Visualizing the Data

Before beginning our predictions and analysis, we should first visualize the data. Below is a graph showing revenue vs. market cap in 2022 and is also colored by CapEx value to visualize three important variables in our data set.

```
library(ggplot2)

imputed_data |>
  ggplot() +
```

```
  geom_point(aes(x = Revenue.2022, y = MarketCapAtEndOfPeriod.2022,
                 color = CAPEXgross.2022)) +
  labs(x = "Revenue in 2022", y = "Market Cap at the end of 2022", color = "Gross CapEx in 2
  theme_bw()
```



From this we can see that there are extreme outliers which may have a large effect on our supervised learning algorithms. In this context, it makes sense to remove these values from the data set before making predictions.

```
imputed_data <- imputed_data |>
  filter(MarketCapAtEndOfPeriod.2023 < 1e+12) |>
  filter(Revenue.2023 < 2e+11) |>
  filter(MarketCapAtEndOfPeriod.2022 < 1e+12) |>
  filter(Revenue.2022 < 2e+11) |>
  filter(MarketCapAtEndOfPeriod.2021 < 1e+12) |>
  filter(Revenue.2021 < 2e+11) |>
  filter(MarketCapAtEndOfPeriod.2020 < 1e+12) |>
  filter(Revenue.2020 < 2e+11)|>
  filter(MarketCapAtEndOfPeriod.2019 < 1e+12) |>
  filter(Revenue.2019 < 2e+11)
```

Also, we will need to scale our data, especially before working with neural networks.

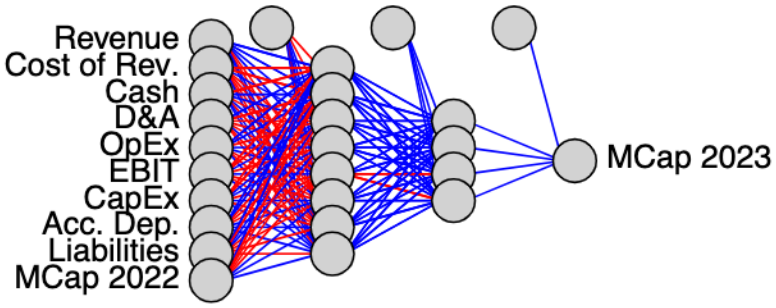
```r
scaled_data <- imputed_data
scaled_data[,3:62] <- scale(imputed_data[,3:62])
```

Neural Networks

One of the most prominent and important supervised learning algorithms in use today are neural networks. The general idea behind neural networks is to mimic how the brain processes data. The network consists of an input layer that takes in the data, hidden layers which process the data, and an output layer that provides the result (in our case this would be our market cap prediction). The connection from one layer to the next contain *weights* that affect how much that variable contributes to the next layer. At each layer, the model adds a bias value and the weight/bias applied to the data form the new layer of data to be passed on. Another important aspect for these networks is *backpropagation*. This is when the model takes the result of the first pass through the network, calculates the loss (i.e. how wrong the model is), and updates the weights in the hidden layers in order to reduce this loss. There are several types of neural networks, but we will be working with a simple application that does involve some backpropagation, which will be helpful for our algorithm accuracy.

We will start with a neural network of all of our 2022 data (predicting 2023 market cap) to visualize it in our context.

```r
library(NeuralNetTools)
NN_model_first <- neuralnet(MarketCapAtEndOfPeriod.2023 ~ .,
                      data = imputed_data[,12:22],
                      hidden = c(8, 4),
                      linear.output = FALSE)

plotnet(NN_model_first, x_names = c("Revenue", "Cost of Rev.", "Cash", "D&A", "OpEx", "EBIT"
```



This model takes in each of the ten financial data variables we have in our data set as inputs. It has two hidden layers with 8 and 4 nodes, and at each of these layers, the model learns how much of each variable to apply at each node and adjusts its values to get closer to the prediction for 2023 market cap. Lastly, it uses backpropagation to learn from its error and readjusts the weights to produce a final prediction for the market cap value.

Now that we have built a neural network and discussed how they work, we can look at actually predicting market cap data and optimizing our algorithm.

First we need to split our data set in half to create a training data set and testing data set so that we can determine our model's accuracy.

```
training_rows <- sample(1:nrow(scaled_data),
                        size = nrow(scaled_data)/2)

train_data <- scaled_data[training_rows, ]
test_data <- scaled_data[-training_rows, ]
```

We will start by only using data from 2022 to predict market cap at the end of 2023.

```
NN_model <- neuralnet(MarketCapAtEndOfPeriod.2023 ~ .,
                      data = train_data[,12:22],
                      hidden = c(8, 4),
                      linear.output = TRUE)
```

```
predictions_scaled <- predict(NN_model, test_data[,12:22])

results_scaled <- data.frame(realMC = (test_data$MarketCapAtEndOfPeriod.2023),
                      predictedMC = predictions_scaled) |>
  mutate(difference = abs(realMC - predictedMC),
         diff = realMC - predictedMC)

mean(results_scaled$difference)
```

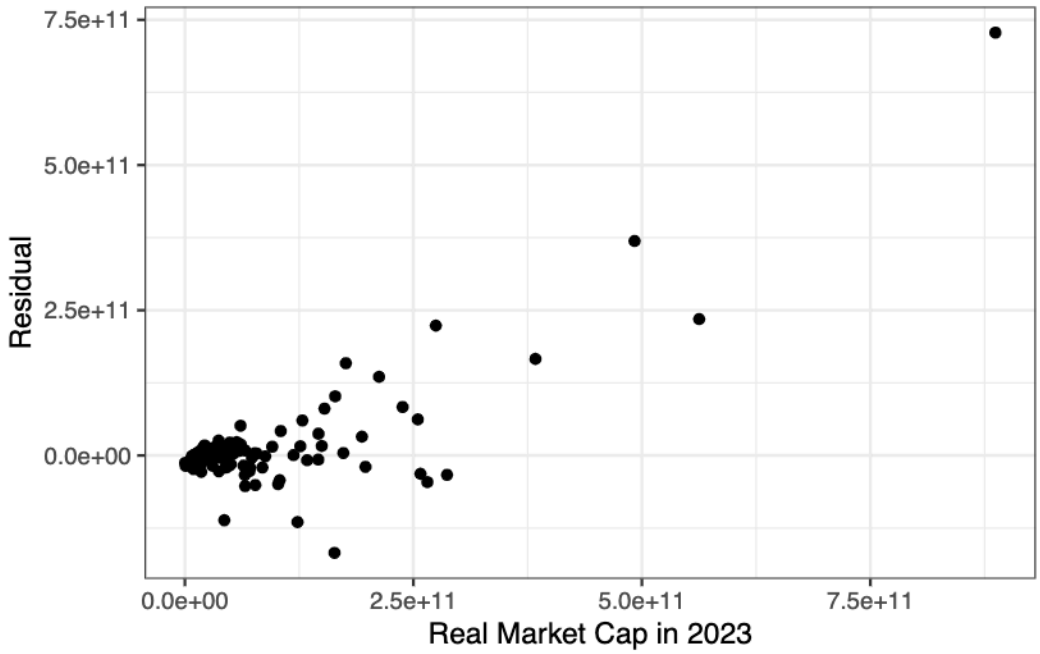
```
[1] 0.3120658
```

```
predictions <- abs(predictions_scaled * 88643586708 + 61827403017)

results <- data.frame(realMC = (test_data$MarketCapAtEndOfPeriod.2023* 88643586708 + 6182740
                      predictedMC = predictions) |>
  mutate(difference = abs(realMC - predictedMC),
         diff = realMC - predictedMC)

mean(results$difference)
```

```
[1] 27662628056
```

We have two values here: our scaled prediction and the unscaled value of this prediction so that we can evaluate it in context. With a mean difference of about $27.7 billion dollars, this algorithm is fairly accurate given our context. It is important to note that since some of our data reaches into the trillions, the residual produced by this prediction vs. actual value may be good for this data value, but when we take the average of all residuals, it has a large impact. In order to visualize how accurate our algorithm is, we can plot our residuals:

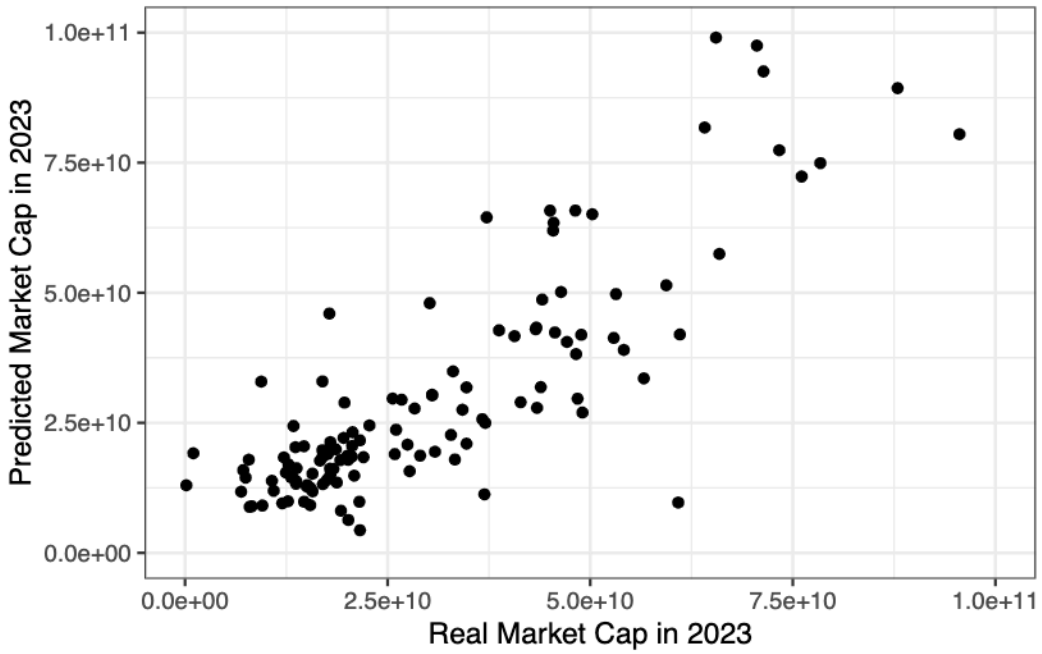
```
library(ggplot2)
results |>
  ggplot() +
  geom_point(aes(x = realMC, y = diff)) +
  labs(x = "Real Market Cap in 2023", y = "Residual") +
  theme_bw()
```



Looking at this, we can see that the majority of our market cap predictions are accurate, as most of the residuals are in a large cluster near the `y (residual) = 0` line. The farther we move away from this cluster on the x-axis, the larger the residuals become. This trend is going to have a large impact on our mean difference accuracy metric, so it is important to take that into account when analyzing this model.

We can also graph our real market cap vs. our predicted market cap and "zoom in" to get another look at how our model is performing:

```
results |>
  ggplot() +
  geom_point(aes(x = realMC, y = predictedMC)) +
  xlim(0, 1e+11) +
  ylim(0, 1e+11) +
  labs(x = "Real Market Cap in 2023", y = "Predicted Market Cap in 2023") +
  theme_bw()
```



In context, if an investor were to be using this model to predict market cap in 2023, they should analyze the outliers and not make any decision based on the predictions for those companies without looking into the company data/information to see why the difference was so extreme (i.e. was there something that happened this year that led to this outcome that may not be the case for other years?).

This last model only looked at 2022 data to predict 2023 market cap. Let's repeat the same steps to create a model using all historical data.

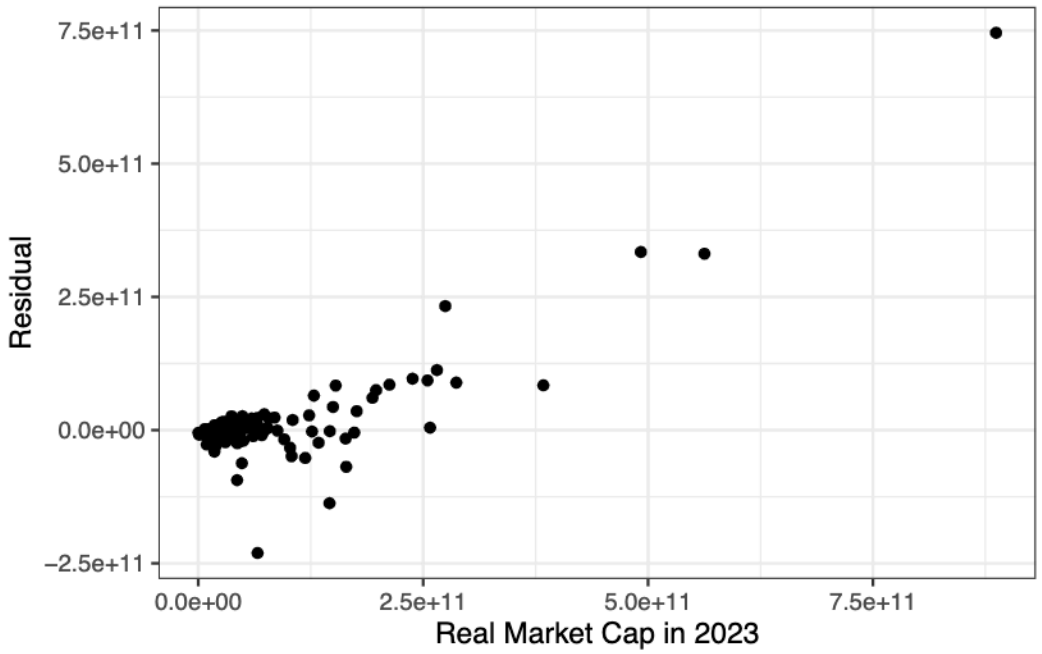
```
NN_model_large <- neuralnet(MarketCapAtEndOfPeriod.2023 ~ .,
                      data = train_data[,12:62],
                      hidden = c(40, 20, 10),
                      linear.output = TRUE)

predictions_scaled2 <- compute(NN_model_large, test_data[,12:62])

predictions_scaled2 <- predictions_scaled2$net.result

results_scaled2 <- data.frame(realMC = (test_data$MarketCapAtEndOfPeriod.2023),
                      predictedMC = predictions_scaled2) |>
  mutate(difference = abs(realMC - predictedMC))

mean(results_scaled2$difference)
```

```
[1] 0.3290123
```

```
predictions2 <- abs(predictions_scaled2 * 88643586708 + 61827403017)

results2 <- data.frame(realMC = (test_data$MarketCapAtEndOfPeriod.2023* 88643586708 + 618274
                       predictedMC = predictions2) |>
  mutate(difference = abs(realMC - predictedMC),
         diff = realMC - predictedMC)

mean(results2$difference)
```

```
[1] 29151639149
```

```
results2 |>
  ggplot() +
  geom_point(aes(x = realMC, y = diff)) +
  labs(x = "Real Market Cap in 2023", y = "Residual") +
  theme_bw()
```



This model using all data is actually less accurate than our previous model using only 2022 data. This makes sense because the most important data for predicting the next year's finances are usually those of the current year (i.e. 2022 for 2023), and data from prior years are used mainly to find growth rates, which we are not considering in our model.

Overall, because this is a simple neural network with only a few hidden layers and minimal backpropogation, it is only somewhat accurate. With a more advanced network algorithm and additional hidden layers, we could improve and optimize this model for real-world application.

Other Supervised Learning Algorithms

Neural networks are an extremely useful and effective algorithm for making predictions, but it is also useful to look at other algorithm types as well. Other algorithms could end up being a better fit depending on the data set and even if they are not, their features can give us insight to how we can better approach our analysis or neural network algorithm optimization.

Let's look at how k-nearest neighbors would predict our data. This algorithm finds the k points that have the closest distance to our input data value and takes the average of their outcome variable. In other words, it would find the k neighbors with the closest revenue, cost of revenue, cash, etc. (all of our inputs), and take the mean of their market caps in 2023.

```
knn_model <- train(MarketCapAtEndOfPeriod.2023 ~ .,
                   data = train_data[,12:62],
                   method = "knn")

knn_model$results
```

Looking at mean absolute error (MAE, the same accuracy metric we used above), and comparing it to the scaled MAE for our neural network, this algorithm is actually performing better on the data, with the lowest MAE at 0.2325 for k = 7.

We can also use a random forest to predict market cap. A random forest is a collection of trees (decision trees that make predictions based on our variables) that are built on random samples of our data, where each tree has a limited number of variables it can make decisions based on. A random forest is a allows us to explore the data in different ways while also preventing overfitting.

```
rf_model <- train(MarketCapAtEndOfPeriod.2023 ~ .,
                   data = train_data[,12:62],
                   method = "ranger",
                   importance = "impurity")

rf_model$results
```

This is the lowest MAE across all of our algorithms with 0.1459 for 50 variables. This means that our random forest is the most accurate when we give it more variables to choose from at each tree, which was the opposite outcome of our neural network analysis.

Random forests also allow us to look at what variables were the most important when making splits in our decision trees.

```
library(caret)
varImp(rf_model)
```

```
ranger variable importance

  only 20 most important variables shown (out of 50)

                               Overall
MarketCapAtEndOfPeriod.2022   100.0000
MarketCapAtEndOfPeriod.2021    71.2962
MarketCapAtEndOfPeriod.2020    64.8200
MarketCapAtEndOfPeriod.2019    50.2265
MarketCapAtEndOfPeriod.2018    42.3132
EBIT.2019                       3.7126
Cash.2018                       3.7084
EBIT.2021                       2.9962
Cash.2021                       2.5728
Cash.2019                       1.8749
EBIT.2022                       1.8209
EBIT.2020                       1.7011
EBIT.2018                       1.5385
Revenue.2022                    0.8537
Cash.2022                       0.8240
DepreciationAmortization.2022   0.8201
Cash.2020                       0.7667
DepreciationAmortization.2020   0.6701
OperatingExpenses.2018          0.6537
DepreciationAmortization.2018   0.6000
```

Related Data

Earlier, we found that our neural network was working best when it was only using 2022 data to predict 2023 market cap. Now, we are going to look at a slightly altered data set that can help us explore this relationship further.

```
new_financial_data <- read_xlsx("Calcbench Data Query Export 2.xlsx")

new_financial_data[new_financial_data == 0] <- NA
```

```
new_clean_data <- new_financial_data[!is.na(new_financial_data$`MarketCap NextYear`),]

colnames(new_clean_data) <- make.names(colnames(new_clean_data))

new_imputed_data <- mice(new_clean_data,
                         method = "rf",
                         seed = 1,
                         maxit = 5)

new_imputed_data <- complete(new_imputed_data)

new_imputed_data <- new_imputed_data |>
  na.omit()

new_imputed_data <- new_imputed_data |>
  filter(MarketCap.NextYear < 2e+12) |>
  filter(Cash.PrevYear < 5.0e+10)

new_scaled_data <- new_imputed_data
new_scaled_data[,3:12] <- scale(new_imputed_data[,3:12])

new_scaled_data_ft <- new_scaled_data |> slice(1:nrow(new_scaled_data)-1)


new_training_rows <- sample(1:nrow(new_scaled_data_ft),
                        size = nrow(new_scaled_data_ft)/2)

new_train_data <- new_scaled_data_ft[new_training_rows, ]
new_test_data <- new_scaled_data_ft[(-new_training_rows), ]
```

This data has all of the same variables as before, but are now labeled as the data for the previous year (.PrevYear) and the market cap data for the year following it (.NextYear). All of the years are combined vertically, and the market cap information for the previous year is not included in the row.

```
new_NN_model <- neuralnet(MarketCap.NextYear ~ .,
                      data = new_train_data[,3:12],
                      hidden = c(5),
                      linear.output = TRUE)

new_predictions_scaled <- predict(new_NN_model, new_test_data[,3:12])
```

```r
new_results_scaled <- data.frame(realMC = (new_test_data$MarketCap.NextYear),
                      predictedMC = new_predictions_scaled) |>
  mutate(difference = abs(realMC - predictedMC))

mean(new_results_scaled$difference)
```

```
[1] 0.2925384
```

```r
new_predictions <- abs(new_predictions_scaled * 134389188796 + 65833103905)

new_results <- data.frame(realMC = (new_test_data$MarketCap.NextYear* 134389188796 + 6583310
                      predictedMC = new_predictions) |>
  mutate(difference = abs(realMC - predictedMC))

mean(new_results$difference)
```

```
[1] 3.82e+10
```

This model is much more applicable and related to how an investor would create a DCF. Now that we have this model and tested it, we would be able to actually apply it to data from this year (for example) and make predictions for future years.

Conclusion

In summary, algorithms such as neural networks and random forests can be a useful way to make predictions for market cap values, but should not be the *only* source of evaluation.

This method is much faster than direct calculations such as a DCF, but is unable to capture the true relationships between the variables the way that typical finance methods do.

These models are looking at trends across all data within the S&P 500 company financial market, but real financial modeling methods mainly look at the data from the company they are trying to predict for (only looking at other comparing companies for additional insights such as growth rate trends).

When dealing with important investments, a careful analysis that is tailored to that specific company and involves the "human" aspect of the investors assumptions is the best option. However, these models, especially a well-trained and detailed deep learning model can be a great addition to an investor's toolkit and variable importance analysis can also help get a better understanding of what needs to be examined in further detail.

Notes

- Further topics/data sets that could be interesting to analyze:
  - Use a data set that includes all of the balance sheet, income statement, and cash flow statement information as variables.
  - If we used a data set that had the information needed to do a DCF and *only* that information (i.e. future cash flows, weighted average cost of capital, growth rates, revenues, operating expenditures, etc.), would our model be able to pick up on the relationship between these variables and the equity value in the same way that a DCF uses their relationship?
- We are using a linear model/relationship for our neural network, so we are not discussing activation functions and their importance for allowing neural networks to find complex relationships (such as image recognition).

References

Calcbench Website (homepage)

`neuralnet` Documentation

Data Camp Neural Network Article

`NeuralNetTools` Documentation

Scaling and Unscaling