Teaching a Machine to Trade Stocks like Warren Buffett, Part I

Utilizing Machine Learning for Stock Fundamental Analysis

Marco Santos
Jan 12, 2020 · 10 min read ★

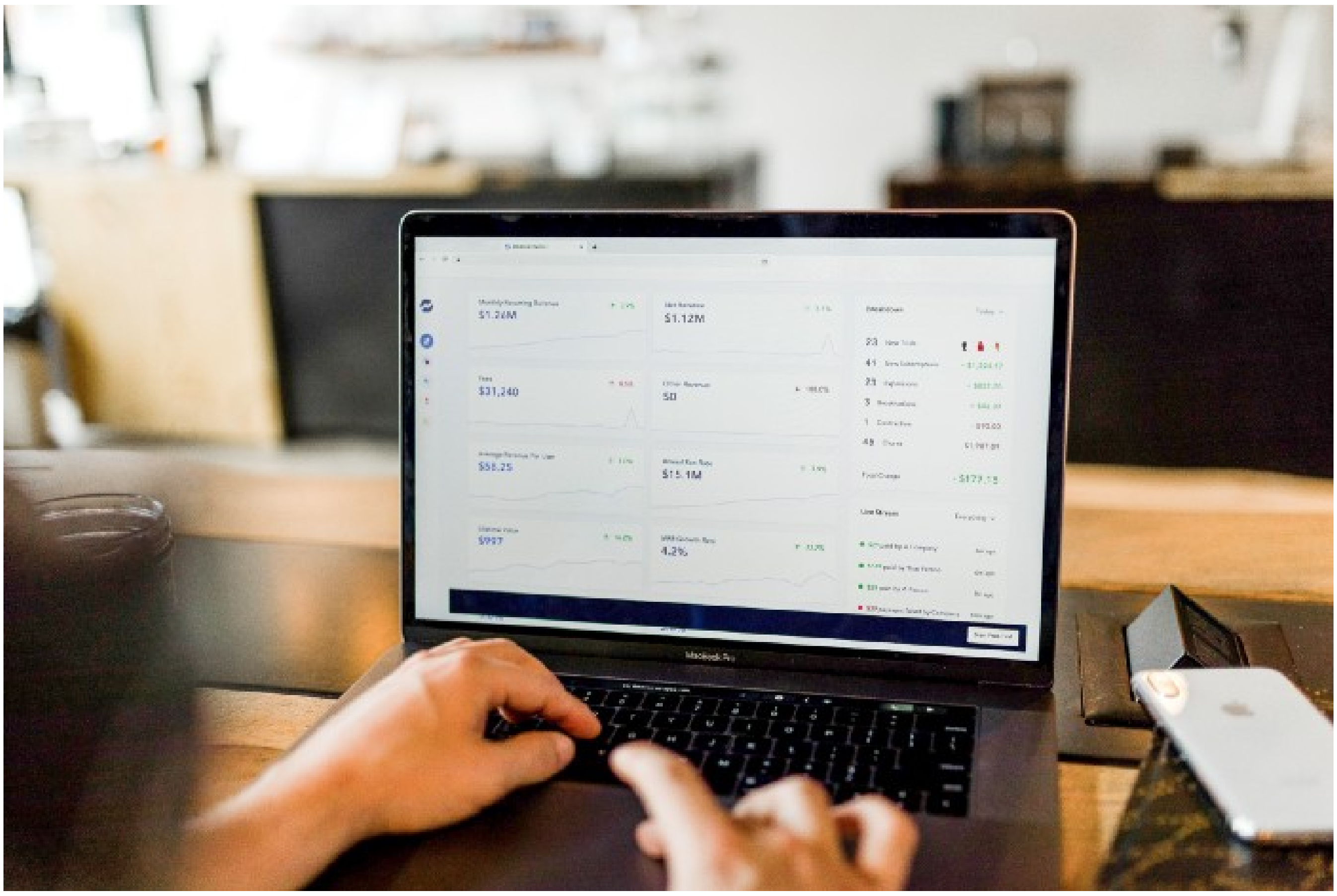


Photo by Austin Distel on Unsplash

Click below for Part 2:

Teaching a Machine to Trade Stocks like Warren Buffett,

Utilizing Machine Learning for Stock Fundamental Analysis

medium.com

When it comes to trading in the Stock Market, there are many different approaches to find the right stock. Many forms of analysis have emerged to detect which stock is worth the money. **Technical Analysis —** determines price patterns over time in hopes that these patterns will emerge again. **Sentimental Analysis —** is based on the public's general feeling or attitude towards a specific stock. If they feel negative, then the stock will tank. If they feel positive, then the sky's the limit. **Fundamental Analysis —** observes a company's financials like their *Balance Sheet* or *Cash Flow Statement* to determine if the company has value relative to their current stock price. These are not the only forms of stock analysis out there but they are arguably the most popular.

Warren Buffett, considered to be one of the most successful investors in the world, is a big proponent of Fundamental Analysis. Now if one of the most successful investors in the world uses this strategy, then we should learn this strategy for ourselves. But not all of us have patience and time to learn Fundamental Analysis. So what if we can get a machine to do the learning for us?

Classification in Machine Learning

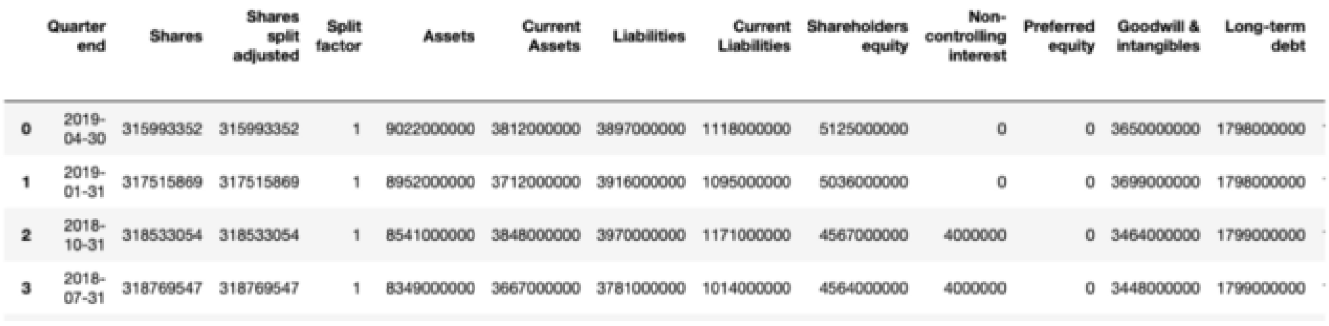
This is where Machine Learning comes into play. We are going to be implementing a machine learning classifier to determine whether a stock is a *Buy*, *Sell*, or a *Hold*. In order to determine whether a stock or company falls into one of these three categories or classes, we will be taking a look at each company's q*uarterly report*. These quarterly reports contain the necessary financial information we require to train our machine learning classifier in the ways of Fundamental Analysis.

The Fundamental Data — Quarterly Reports

A large amount of quarterly reports will need to be collected in order to train our classifier model. Stockpup.com contains the quarterly report history from numerous companies and has made them available to download in convenient CSV or Excel files. But, how many companies' quarterly report history should you download? As many as you can, if not all quarterly reports on the site. Some companies listed on the website no longer exist but that doesn't mean that their quarterly report history is useless. We can still use that data when training our classifier.

Examining the Quarterly Reports

Once we have downloaded the necessary fundamental data (quarterly reports), we can take a look at how the data is formatted below displayed using a Pandas DataFrame:



	Quarter end	Shares	Shares split adjusted	Split factor	Assets	Current Assets	Liabilities	Current Liabilities	Shareholders equity	Non-controlling interest	Preferred equity	Goodwill & intangibles	Long-term debt
0	2019-04-30	315993352	315993352	1	9022000000	3812000000	3897000000	1118000000	5125000000	0	0	3650000000	1798000000
1	2019-01-31	317515869	317515869	1	8952000000	3712000000	3916000000	1095000000	5036000000	0	0	3699000000	1798000000
2	2018-10-31	318533054	318533054	1	8541000000	3848000000	3970000000	1171000000	4567000000	4000000	0	3464000000	1799000000
3	2018-07-31	318769547	318769547	1	8349000000	3667000000	3781000000	1014000000	4564000000	4000000	0	3448000000	1799000000

Preview of the Quarterly Report history for one specific company

Here we can observe all the different columns and dates corresponding to each column. Now let's take a look at the *Price*, *Price High*, and *Price Low* columns. This information is usually not provided in quarterly reports but Stockpup.com is kind enough to supply this, which is important for us in determining if the stock is a *Buy*, *Hold*, or *Sell*.

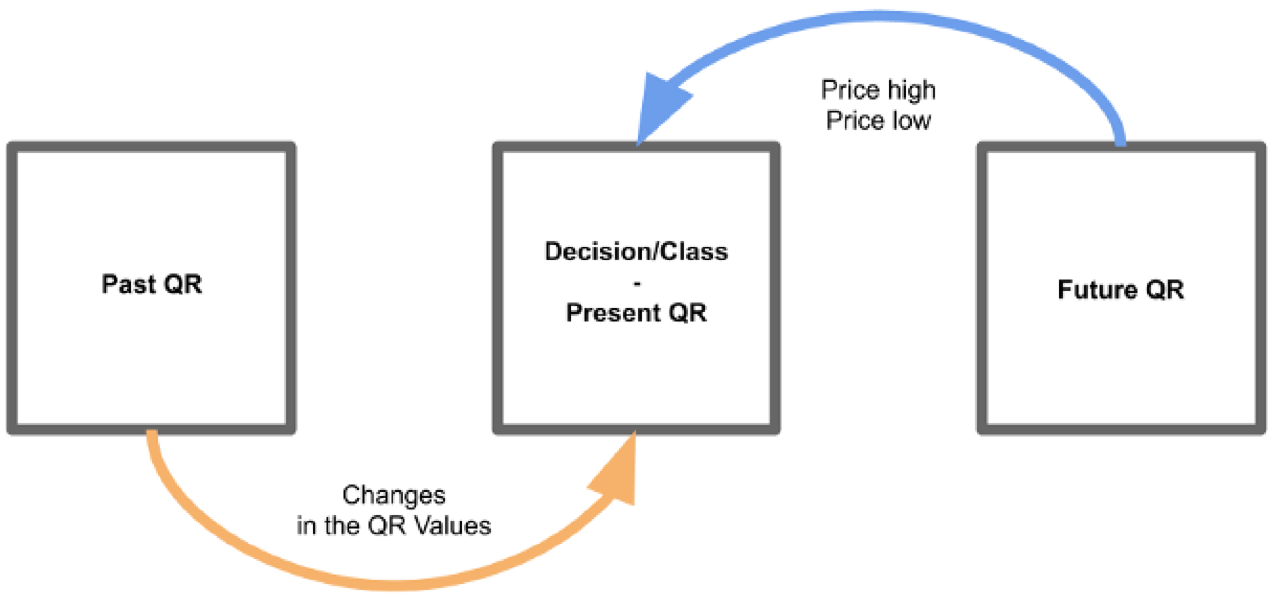
Now, there are various and unique ways to determine whether a stock is worth investing in or not. Could we potentially classify it as a *Buy* if the *Assets* increased and *Liabilities* decreased over the course of three quarters? Or maybe if the *Shares* increased and *Long-term debt* decreased? Anyways, we can see there are numerous options to determine the stock's class. Simply put, this is the basis of Fundamental Analysis.

Performing our own Fundamental Analysis

How do we fundamentally analyze these quarterly reports without knowing Fundamental Analysis? Since we are not Fundamental experts like Mr. Buffett, let's try to simplify it with our own method of fundamental analysis:

- Based on a quarterly report from any chosen quarter, observe how much the values changed from the previous report to the currently selected one.
- Then, observe the price values from the next quarterly report to see if there were any significant jumps in price.
- Finally, with the present report containing the changes from the past report and the future report's price behavior, determine if it is a *Buy*, *Hold*, or *Sell*.

Essentially, we are detecting if fundamental changes from the previous to the current quarter affect future prices. *We will be judging the performance of each quarterly report relative to its last report and then observing the future's price behavior*. See the diagram below for clarification:



The Decision (deciding if Buy, Hold, Sell) on the Present quarterly report (QR) takes in information using both the Past and Future quarters

We will be applying this method of analysis to each quarterly report to create our new fundamental data. This method will classify if a stock is worth investing in for the quarter. Evidently, we won't be able to use this method on the very first QR or on the most recent QR because the nature of the analysis requires a past and future quarterly report.

Cleaning and Formatting the Data

Creating our Class Labels

To categorize or classify each quarterly report, we will keep things uncomplicated. If the price went up a significant amount in the next quarter, then it's a *Buy*. If down, then it's a *Sell*. If neither, then a *Hold*.

Here are our specific class requirements for each observed quarterly report:

- **Buy** — *Price high* and *Price low* increase by 3% or more in the next quarter.
- **Sell** — *Price high* and *Price low* decrease by -3% or more in the next quarter.
- **Hold** — If neither happens.

There are other possible outcomes we could consider but for the sake of simplicity, let's keep these conditions as they are. We won't know the next quarter's *Price high* and *Price low* for the most recent quarterly report because that would be knowing the future, which is impossible.

Fundamental Data from Quarterly Reports

For our fundamental data (*Shares*, *Assets*, etc.), like mentioned before, we will be observing two QRs to create new values. The changes from the previous QR to the current or present QR will be measured as a percent change instead of their actual value. For example:

> *Let's say in the previous quarter, Shares show a value of 1,000. Then in the current quarter, Shares now show a value of 1,100; an increase of 10%. Now, we replace the Share value of 1,100 with 10% for our current QR.*

We do this process for every QR (excluding the very first one because we can't compare it to something that doesn't exist). Now every QR has percent change for each Fundamental value.

Coding out our Fundamental Analysis

Now that we have the QRs formatted with percent changes and labeled as *Buy*, *Hold*, or *Sell*, we can move on to showing the programming process:

Importing Libraries and Dataset

```python
# Pandas in order to view and format the data
import pandas as pd

# Pandas option so we can see all rows and columns
pd.set_option('display.max_rows', 200)
pd.set_option('display.max_columns', 70)

# Numpy for necessary calculations
import numpy as np

# Jupyter widget progress bar
from tqdm import tqdm_notebook as tqdm

# Pickle to import and export the formatted datasets
import _pickle as pickle

# Loading a pickle file that contains a dictionary with the keys being stock tickers ar
with open("stockpup.pkl",'rb') as fp:
    stocks_df = pickle.load(fp)
```

class_lib.py hosted with ❤ by **GitHub** **view raw**

The pickle file we loaded won't be found on the Github.

The pickle file is a dictionary of DataFrames that contain the QRs for every ticker/company found on Stockpup.com. In order to follow along, you'll have to replace the pickle file we used with a DataFrame of QRs for each ticker you were able to download formatted as a dictionary of DFs.

Helper Functions to Assist in the Preprocessing

```python
def setting_index(df):
    """
    Returns a sorted datetime index
    """
    df['Quarter end'] = pd.to_datetime(df['Quarter end'])
    df.set_index("Quarter end", inplace=True)
    return df.sort_index(ascending=True)
```

```python

def class_creation(df, thres=3):
    """
    Creates classes of:
    - buy(1)
    - hold(2)
    - sell(0)

    Threshold can be changed to fit whatever price percentage change is desired
    """
    if df['Price high'] >= thres and df['Price low'] >= thres:
        # Buys
        return 1

    elif df['Price high'] <= -thres and df['Price low'] <= -thres:
        # Sells
        return 0

    else:
        # Holds
        return 2
```

class_help_func.py hosted with ❤ by **GitHub** **view raw**

We will use these functions later on

Iterating through each Ticker to Transform the Data

```python
# Setting the index as the Date
for i in tqdm(stocks_df.keys()):
    stocks_df[i] = setting_index(stocks_df[i])

# Replacing all "None" values with NaN
for i in tqdm(stocks_df.keys()):
    stocks_df[i].replace("None", 0, inplace=True)

# Creating a new dictionary that contains the numerical values, then converting all va
num_df = {}
for i in tqdm(stocks_df.keys()):
    num_df[i] = stocks_df[i].apply(pd.to_numeric)

# Replacing values with percent difference or change
pcnt_df = {}
for i in tqdm(num_df.keys()):
    pcnt_df[i] = num_df[i].pct_change(periods=1).apply(lambda x: x*100)

# Replacing infinite values with NaN
```

```python
for i in tqdm(pcnt_df.keys()):
    pcnt_df[i] = pcnt_df[i].replace([np.inf, -np.inf], np.nan)

# Creating a new DataFrame that contains the class 'Decision' determining if a quarter
new_df = {}
for i in tqdm(pcnt_df.keys()):
    # Assigning the new DF
    new_df[i] = pcnt_df[i]

    # Creating the new column with the classes, shifted by -1 in order to know if the p
    new_df[i]['Decision'] = new_df[i].apply(class_creation, axis=1).shift(-1)

# Excluding the first and last rows
for i in tqdm(new_df.keys()):
    new_df[i] = new_df[i][1:-1]

# Combining all stock DFs into one
big_df = pd.DataFrame()
for i in tqdm(pcnt_df.keys()):
    big_df = big_df.append(new_df[i], sort=False)

# Filling the NaNs with 0
big_df.fillna(0, inplace=True)

# Resetting the index because we no longer need the dates
big_df.reset_index(drop=True, inplace=True)

# Dropping the price related columns to prevent data leakage
big_df.drop(['Price', 'Price high', 'Price low'], 1, inplace=True)

# Exporting the final DataFrame
with open("main_df.pkl", 'wb') as fp:
    pickle.dump(big_df, fp)
```

A couple things to note:

- *First and last rows are excluded because the first row is unable to show change and the last row has no future quarter to find price values.*
- *All DataFrames from the dictionary of DFs are combined as one in order to train our classification model later on.*

- *Index is reset because dates are no longer required. Each row or QR now contains information from the past and future QRs so dates are no longer important for the model.*
- *Price related features or columns are dropped to prevent data leakage. In normal QRs, these features are not included.*

Finally, we have our fundamental data correctly formatted and transformed in the way that we desired. We export the final DF for data exploration and to train the classification model.

Exploring the Data

Next, in order to become more familiar with our data, we will have to perform some simple **Exploratory Data Analysis**. We do this to better understand our data and make sure there are no lingering issues we may have missed when we transformed the data. So let's begin coding it out with some visualizations.

Library Imports and Loading in the DF we created earlier

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import _pickle as pickle

# Opening the .pkl file created
with open("main_df.pkl",'rb') as fp:
    final_df = pickle.load(fp)
```

class_eda_lib.py hosted with ❤ by **GitHub** **view raw**

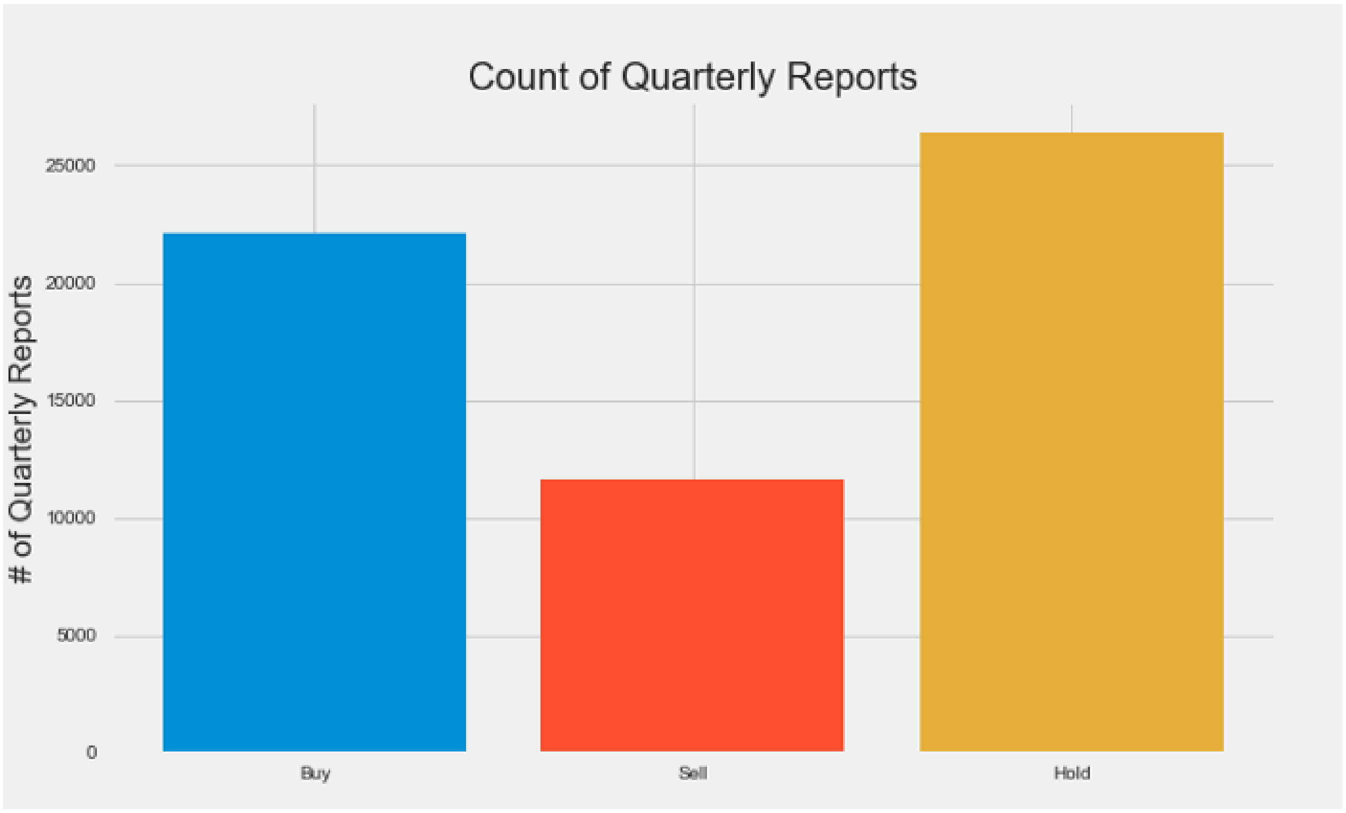
Visualize the Count of our Classes

```python
# Separating each class into respective DataFrames
buy_df = final_df[final_df['Decision']==1].loc[:, final_df.columns != 'Decision'].reset
hold_df = final_df[final_df['Decision']==2].loc[:, final_df.columns != 'Decision'].rese
sell_df = final_df[final_df['Decision']==0].loc[:, final_df.columns != 'Decision'].rese

# Visualizing in matplotlib
```

```python
plt.figure(figsize=(10,6))
plt.style.use('fivethirtyeight')

# Plotting the count of each DataFrame of each class
plt.bar("Buy", buy_df.shape[0])
plt.bar("Sell", sell_df.shape[0])
plt.bar("Hold", hold_df.shape[0])

plt.ylabel("# of Quarterly Reports")
plt.title('Count of Quarterly Reports')
plt.show()
```

class_bal.py hosted with ❤ by **GitHub** **view raw**

This will display a chart of our class balance



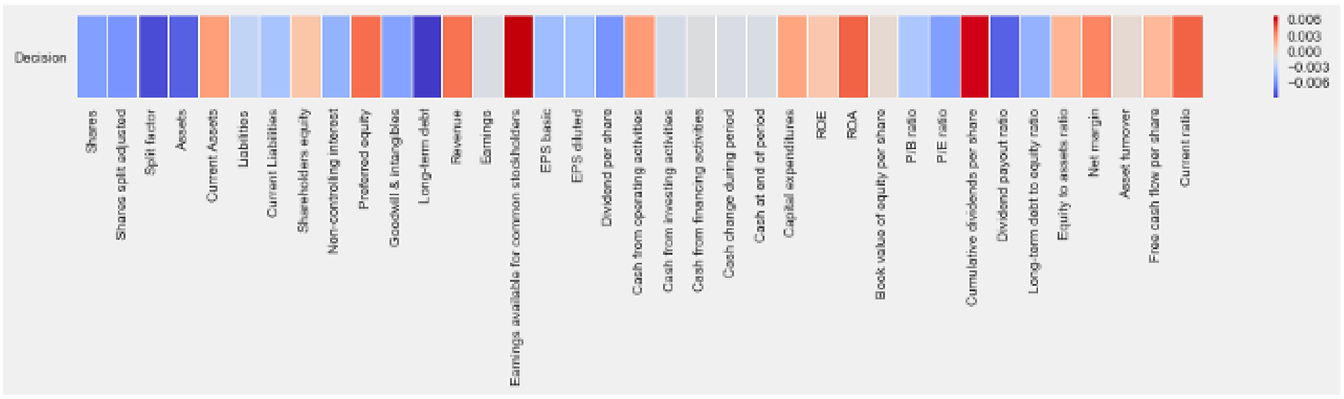
# of Quarterly Reports labeled as Buy, Hold, Sell

As you can see, there is a class imbalance issue with our data. This may be a problem but we do not want to discard data points so that each class equals the class with the least amount. Even though that may be a viable solution to class imbalance, there are other options we can explore. Another option would be using a different *evaluation metric* when validating our classification model. We will expand on this option when get to the modeling phase.

Correlations in our Data

```python
def CorrMtx(df, dropDuplicates = True):
    """
    Takes in a Correlation DF and excludes nonessential visuals.
    Creates a more visually pleasing correlation matrix
    """

    # Exclude duplicate correlations by masking uper right values
    if dropDuplicates:
        mask = np.zeros_like(df, dtype=np.bool)
        mask[np.triu_indices_from(mask)] = True

    # Set background color / chart style
    sns.set_style(style = 'white')

    # Set up  matplotlib figure
    f, ax = plt.subplots(figsize=(11, 9))

    # Add diverging colormap from red to blue
    cmap = sns.diverging_palette(250, 10, as_cmap=True)

    # Draw correlation plot with or without duplicates
    if dropDuplicates:
        sns.heatmap(df, mask=mask, cmap=cmap,
                square=True,
                linewidth=.5, cbar_kws={"shrink": .5}, ax=ax)
    else:
        sns.heatmap(df, cmap=cmap,
                square=True,
                linewidth=.5, cbar_kws={"shrink": .5}, ax=ax)


# Correlation DF of all classes
corr = final_df.corr().iloc[[-1],:-1]

# Plotting the Correlation DF as a heatmap
plt.figure(figsize=(14,1))
sns.heatmap(corr, annot=False, linewidths=.1, cmap="coolwarm")
plt.xticks()
plt.yticks(rotation=0)
plt.show()
```

This will display the correlation matrix



Correlation of features with the Decision/class label

From what we can see, some features are influential in determining the class label. Some have little to no correlation to the *Decision* of whether or not the stock/quarterly report is worth Buying, Selling, or Holding. Now that we know some features from the QR are unimportant in determining the class label, we can remove those features from the dataset.

Feature Engineering and Selection

Now that we have explored our data a little bit, we could continue doing some more exploration or move onto the **Feature Engineering** process or more specifically, **Feature Selection**. Feature engineering is the process of altering our dataset in order to enhance our machine learning models. There are a number of options for engineering the features of our dataset. They include but are not limited to:

- Creating interactive features. Have two distinct features interact with each other to create a brand new feature.
- Reducing the number of features. Since we have over 30 fundamental features, we could eliminate unimportant features in order to improve the performance of our models.
- Using domain knowledge. If we knew which features were most essential to a stock’s price based on economic and financial research, then we could attribute weights to those features.

Feature Selection

For the purposes of this project, we will simply be eliminating unimportant features. By doing so, we can potentially improve our model's accuracy and also reduce training time. This action is known as *Feature Selection*; the process of selecting features we consider necessary for our model. We will show 2 different ways of Feature Selection. This way we can compare and contrast the performances of our model with slightly different datasets due to our methods of selection.

Method #1: Selecting the Top 10 Features based on Correlation Values

```python
# Correlation DF of all classes
corr = final_df.corr().iloc[[-1],:-1]

# Sorting our Correlation DF by their absolute values and selecting the top 10
top10_corr = corr.transpose().apply(abs).sort_values(by='Decision', ascending=False)[:1

# Creating a new DF with the features from the top10_corr and joing the 'Decision' clas
top10_corr_df = final_df[top10_corr.index].join(final_df.Decision)

# Pickling the DF for use in our Classification models
with open("top10_corr_df.pkl", "wb") as fp:
    pickle.dump(top10_corr_df, fp)
```

top10_corr.py hosted with ❤ by **GitHub** **view raw**

Long-term debt
Split factor
Assets
Dividend payout ratio
Earnings available for common stockholders
Cumulative dividends per share
Dividend per share
Shares split adjusted
P/E ratio
Shares

Our Top 10 correlated features (ordered from most correlated to least)

Using the correlation matrix we created while we were exploring the dataset, we select the top 10 features correlated with the *Decision* class labels (*see above image*). This is done by:

1. Taking the absolute value of each feature (to handle negative correlations).
2. Sorting by the absolute values.
3. Slicing the number features to only include the top 10 most correlated.
4. Joining the *Decision* class labels so that they are included in the new DF.
5. Exporting the Top 10 Correlation DF as a pickle file.

Nice! We now have a new reduced but concentrated dataset for our classification models. Hopefully this new dataset enhances our classification models and speeds up training time.

Method #2: Selecting the Top 10 Features based on a Tree Classifier

This next method of Feature Selection will be a little bit more complicated than the one we just did. By using Sci-Kit Learn's library, we will be implementing a *Decision Tree* based classifier to determine which features are most important. By using this classifier, we will be able to determine which features are most important when classifying the QR or stock.

```python
# Importing the necessary libraries
from sklearn.ensemble import ExtraTreesClassifier

# Instatiating the classifier
forest = ExtraTreesClassifier(n_estimators=200)

# Setting the corresponding variables for our classifier
X = final_df.drop(['Decision'], 1)
y = final_df.Decision

# Fitting the classifier
forest.fit(X, y)

# Determining the important features
importances = forest.feature_importances_
```

```

# The standard deviation among the trees for the important features
std = np.std([i.feature_importances_ for i in forest.estimators_], axis=0)

# Indexing and sorting the important features
indices = np.argsort(importances)[::-1]

# Assigning the top 10 features as a new DF
top10_df = final_df[X.columns[indices][:10]].join(final_df.Decision)

# Exporting the top 10 features DF
with open("top10_df.pkl", "wb") as fp:
    pickle.dump(top10_df, fp)
```

class_feat_imp.py hosted with ❤ by **GitHub** **view raw**

Finding the feature importances

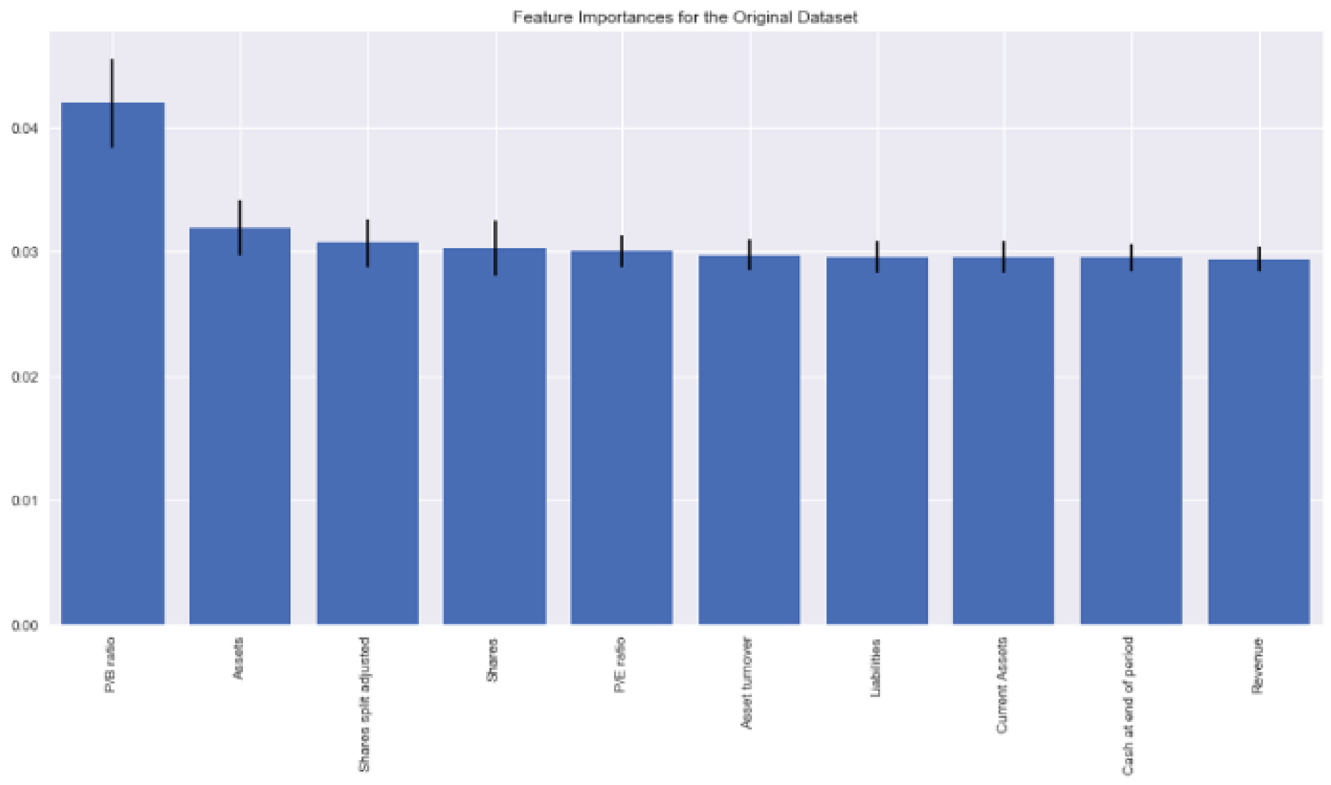
Next, with our important features all sorted out, we'll visualize them to see exactly which features were determined to be most important. Just to get an idea of which features were selected.

```
# Matplotlib style to use
plt.style.use('seaborn')

# Printing out the different features as a list
print("Feature Rankings:")

# Showing the top 10 features
for i in range(10):
    print(f"{i+1}. {X.columns[indices[i]]}: {importances[indices[i]]}")

# Plotting the top 10 features
plt.figure(figsize=(14,7))

plt.title("Feature Importances for the Original Dataset")
plt.bar(range(X.shape[1]), importances[indices], yerr=std[indices], align='center')

plt.xticks(range(X.shape[1]), X.columns[indices], rotation=90)
plt.xlim([-1, 11.5])
plt.show()
```

class_feat_imp_plot.py hosted with ❤ by **GitHub** **view raw**

Visualizing our features

Running this code will give us:



According to our `forest.feature_importances_` method, these are the Top 10 most important features:

1. P/B Ratio
2. Assets
3. Shares split adjusted
4. Shares
5. P/E Ratio
6. Asset turnover
7. Liabilities
8. Current Assets
9. Cash at end of period
10. Revenue

The last line in our code exported the DF of the 10 most important features. Now we have two Top 10 DFs with some different and some same features represented in both.

We can use both of these DFs to find out which performs better with our machine learning classifiers.

Next up, Machine Learning Classifiers

With our data cleaned, formatted, and selected, we can move on to classifying stocks based on their Quarterly Reports! In the next part, we will be coding out multiple different machine learning classifiers and finding out which classifier performs the best with our data.

Click below for Part 2:

Teaching a Machine to Trade Stocks like Warren Buffett, Part II

Utilizing Machine Learning for Stock Fundamental Analysis

medium.com

*Follow me on Twitter: @_Marco_Santos_*

Resources:

marcosan93/Stock-Performance-Predictor-2.0

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

Machine Learning Python Artificial Intelligence Data Science Finance