# Abstract & problem statement

This project aims to find a set of features, feature transformations and models that will best predict the direction of the Dow Jones Industrial Average (DJIA).

Established in 1885, the DJIA, is a price weighted stock index that measures the performance of the stocks of 30 large companies listed on stock exchanges in the United States with a market cap of US$6.56 trillion. It is one of the most followed indexes for equities and if often used in benchmarking portfolios, predicting recessions, US economy performance among other indicators.

We have explored the usage of multiple features with different transformation with multiple machine learning models in order to achieve the highest prediction accuracy for the DJIA index.

We modelled the problem as a classification problem with 2 signals, Buy and Sell with daily frequency. They are defined as follows:

* St = BUY if Pt+1 ≥ Pt
* St = SELL if Pt+1 < Pt

Where St is the signal at time t and Pt is the DJIA closing price at time t.

We explored the relationship between DJIA and multiple other assets in order to narrow down a few to work with and then applied various feature transformation to improve our predictions results.

Next, in this report we will describe our base set of features and justify as to why we decided to use them, then we will describe the time series analysis we conducted on our dataset, followed by some financial technical analysis feature transformations used in the project followed by the models that we decided to use and why we used them and finally we will then jump in the results. Due to large amount of results collected from various feature and feature transformation sets we will be dividing the results by the feature and feature transformations that produced them.

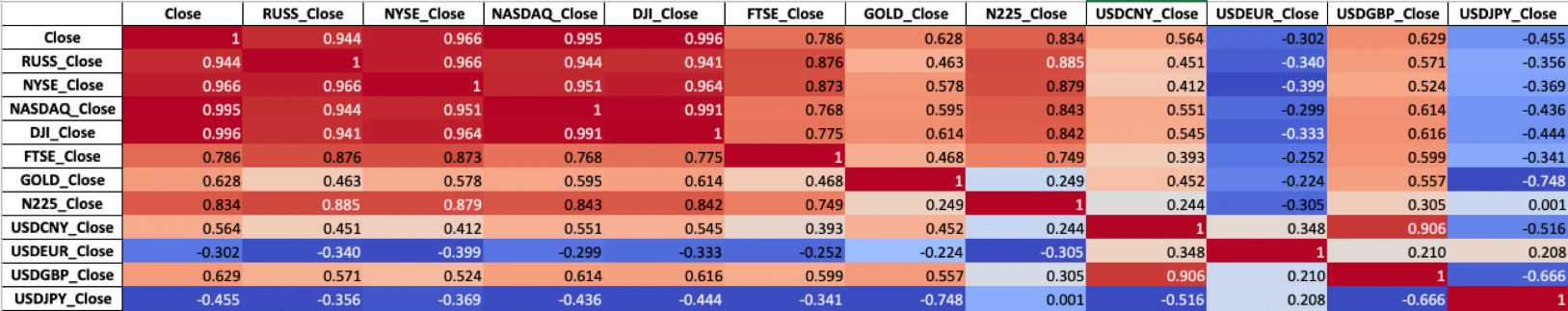
# The base features & feature transformation

In addition to the Open, High, Low and Close prices and daily Volume of the DJIA, we used daily Close prices other indexes, currencies as well as commodities. We used both 5 and 10 year historical data in an attempt to examine the effect on our prediction of simply adding more data with the same feature transformations and models.

\*\*EXPLAIN WHY THE FOLLOWING FEATURES ARE GOOD TO USE\*\*

The indexes used are NYSE, Russell, NASDAQ, S&P500, FTSE and N225. For commodities we used gold closing prices, for currency exchange rates we used the pairs USDCNY, USDEUR, USD GBP and USDJPY.

The below is a correlation matrix of our features against the Closing price of the DJIA



As the diagram above shows, there are strong correlations between the closing prices of the DJIA and the other assets. Clearly some assets are positively correlated and some are negatively which made use of as the report will later show. It is also worth noting that the above diagram is plotted based on the raw prices, which we later also transformed in order to make the data more uniform.

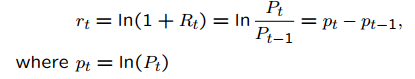
Our features set is not uniform in multiple ways. Not all of our features assets trade on the save trading venue which means that they are traded in different currencies e.g. N225 is a Japanese index and thus is traded in the Japanese Yen (JPY) while the FTSE is a British index that trades in GBP while the DJIA trades in USD. In order to take out the effect of the currencies in the prices we calculated the log returns of each of the assets and used them instead of the raw closing prices.

## Log returns \*\*REPHRASE THIS SECTION\*\* <KABIR>

Reference: <https://quantivity.wordpress.com/2011/02/21/why-log-returns/>

Given log returns are beyond the scope of this course, we will go ahead and explain it here.

Log returns are calculated as follows:



Where rt = return at time t, Pt is the price at time t

Benefit of using returns, versus prices, is normalization: measuring all variables in a comparable metric, thus enabling evaluation of analytic relationships amongst two or more variables despite originating from price series of unequal values. This is a requirement for many multidimensional statistical analysis and machine learning techniques.

Several benefits of using log returns, both theoretic and algorithmic.

First, log-normality: if we assume that prices are distributed [**log normally**](http://en.wikipedia.org/wiki/Log-normal_distribution) (which, in practice, may or may not be true for any given price series), then log(1 + r_i) is conveniently [**normally distributed**](http://en.wikipedia.org/wiki/Normal_distribution), because:

1 + r_i = \frac{p_i}{p_j} = \exp^{\log(\frac{p_i}{p_j})} 

This is handy given much of classic statistics presumes normality.

Second, approximate raw-log equality: when returns are very small (common for trades with short holding durations), the following approximation ensures they are close in value to raw returns:

\log(1 + r) \approx r , r \ll 1 

Third, time-additivity: consider an ordered sequence of n trades. A statistic frequently calculated from this sequence is the compounding return, which is the running return of this sequence of trades over time:

\displaystyle (1 + r_1)(1 + r_2)  \cdots (1 + r_n) = \prod_i (1+r_i)

This formula is fairly unpleasant, as probability theory reminds us the product of normally-distributed variables is not normal. Instead, the sum of normally-distributed variables is normal (important technicality: only when all variables are uncorrelated), which is useful when we recall the following logarithmic identity:

\log(1 + r_i) = log(\frac{p_i}{p_j}) = \log(p_i) - log(p_j) 

Thus, compounding returns are normally distributed. Finally, this identity leads us to a pleasant algorithmic benefit; a simple formula for calculating compound returns:

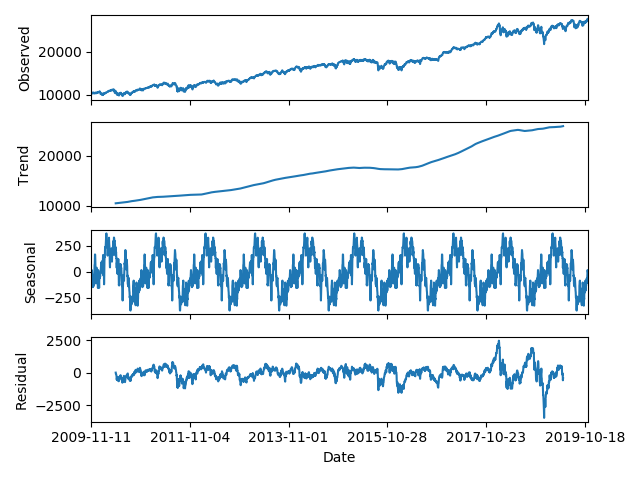
\displaystyle \sum_i \log(1+r_i) = \log(1 + r_1) + \log(1 + r_2)  + \cdots + \log(1 + r_n) = \log(p_n) - \log(p_0)

Thus, the compound return over n periods is merely the difference in log between initial and final periods. In terms of [**algorithmic complexity**](http://en.wikipedia.org/wiki/Big_O_notation), this simplification reduces O(n) multiplications to O(1) additions. This is a huge win for moderate to large n. Further, this sum is useful for cases in which returns diverge from normal, as the [**central limit theorem**](http://en.wikipedia.org/wiki/Central_limit_theorem) reminds us that the sample average of this sum will converge to normality (presuming finite first and second moments).

Most of the results we will show later used log returns instead of raw prices unless otherwise indicated.

# Time series analysis <DANISH>

Given that we are working with time series, we went ahead and visualized each of our features and the DJIA for a better understanding. The below is a diagram of the decomposed time series of the DJIA close price:



Next we will introduce some concepts that are beyond the scope of the course but were used for feature transformations in our project.

## Stationarity

## Autocorrelations and partial autocorrelation

## Seasonality

As we can observe, there is clearly a seasonal factor in the prices of the past 10 years of the DJIA.

# 

# Technical analysis & feature transformation <KABIR>

# The models\*\*ADD BRIEF DESCRIPTIONS OF EACH MODELS WITH DIAGRAMS\*\* <YUKI>

### **Random Forest Classifier**

A picture containing text, map

Description automatically generated  
<https://www.globalsoftwaresupport.com/random-forest-classifier/>

Random Forest was an ensemble model made of a multitude of decision trees, generated through the use of bootstrapping. Predication was often made from average voting.

Number of Iteration: 100

Cross-validation: 3 folds

Parameter grid:

Depth of tree: 10 to 100 with a step of 10, and unrestricted

Minimum number of leafs: 1, 2, and 4

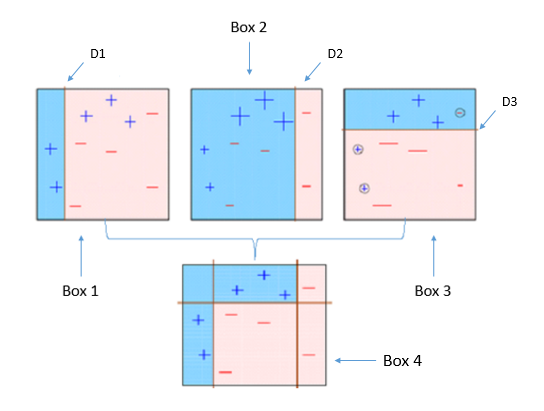
Minimum number of samples for a node split: 2, 5, 10

Number of trees: 600 to 2,000, with a step of 200

Number of features to consider for best split: square root

Use bootstrap for tree generation: both

### **Adaboost Classifier**



<https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe>

Adaboost was another ensemble classifier, which combined weak classifier algorithms to form a strong classifier. The model relied on dataset previously trained, and the weights generated according to the accuracy it returned. The model then determined the updates of dataset using these weights for the input of next sub-model.

Cross-validation: 3 folds

Epoch: 10

Parameter grid:

Loss function: Linear, Square, Exponential

### **Support Vector Machine**

A close up of a logo

Description automatically generated  
<https://en.wikipedia.org/wiki/Support-vector_machine>

Support Vector Machine applied kernel function to perform discriminative classification by finding the hyperplane that maximised the margin between the two classes.

Parameter grid:

Penalty parameter: 0.001 to 1,000 with a step of 10

Kernel: RBF kernel

Kernel coefficient: 0.001& 0.0001

### **K Nearest Neighbors**

K Nearest Neighbors was a supervised learning algorithm; classification was based on the distance between the data points and its nearest neighbors.

Cross-validation: 2 folds

Parameter grid:

Number of neighbors: 1 to 30

Power parameter for the Minkowski metric: 1 to 9

### **Gradient Boosting**

A close up of a piece of paper

Description automatically generated  
<https://www.researchgate.net/figure/A-simple-example-of-visualizing-gradient-boosting_fig5_326379229>

Gradient boosting produced a prediction model that ensembled weak decision trees. Unlike Random Forest, it combined the trees at the beginning of the process. In the iterative processes the decision tree considered the residual error, instead of updating the weight of data points produced in the previous iteration.

Cross-validation: 5 folds

Minimum number of samples for a node split: 2

Minimum number of leafs: 1

Number of features to consider for best split: square root

Parameter grid:

Maximum depth of tree: 2 to 7

Number of trees: 100, 250 to 1,750, with a step of 250

Learning rate: 0.001, 0.005, 0.01, 0.05, 0.1, 0.15

### **Logistic Regression**

Logistic regression was a binary regression model that used a logistic function to model/describe a dichotomous dependent variable

Parameter grid:

Inverse of regularization strength: 0.001 to 1000 with a step of 10

Optimisation solver: Newton-conjugate gradient, L-BFGS, Lib-linear, SAG, SAGA

### **Forward Feed Neural Networks**

A close up of a logo

Description automatically generated  
<https://www.researchgate.net/figure/Multilayer-Perceptron-with-two-hidden-layers_fig1_274084844>

A Forward Feed Neural Networks was the simplest neural network constructed with layers of hidden units called perceptions. Information moved only in one forward direction from input nodes to the output nodes.

Epoch: 100

Sequential Model:

1. Fully Connected Layer
   1. Output size: 16
   2. Input size: number of features
   3. Activation function: Rectified Linear Unit
2. Fully Connected Layer
   1. Output size: 32
   2. Activation function: Rectified Linear Unit
3. Fully Connected Layer
   1. Output size: 1
   2. Activation function: Sigmoid function

Loss Function: Binary Cross-entropy

Early Stopping applied, but allowing 10 epochs with no improvement

### **Long short-term memory**

A screen shot of a computer

Description automatically generated  
<https://en.wikipedia.org/wiki/Long_short-term_memory>

LSTM was a variation of RNN, which was invented for predicting sequential data. It implemented Input, Output, Forget gates and internal memory across consecutive states, which allowed valuable information be retained until a signal triggered the model to discard it.

Epoch: 15

Steps per epoch: 10

Optimiser: Adam

Loss function: Mean Absolute Error

LSTM Model:

1. LSTM Layer
   1. Output size: 64
   2. Input size: (4, number of features)
2. Fully Connected Layer
   1. Output size: 32
   2. Activation function: Rectified Linear Unit
3. Fully Connected Layer
   1. Output size: 16
   2. Activation function: Rectified Linear Unit
4. Fully Connected Layer
   1. Output size: 1
   2. Activation function: Linear activation

### **Autoregressive integrated moving average**

### **ARIMA - Autoregressive integrated moving average was a regression analysis that predicted future values of a series based entirely on its own inertia, lagged or prior values.**

p, d, q Parameters :

p, number of time lags: 2

d, degree of differencing: 1

q, order of moving average model: 2

# Results \*\*SHOW RESULTS CATEGORICALLY\*\* <CLIFF>

## **Training/Testing Datasets**

Daily closing prices of the mentioned currencies and indices across 10 years were used for making the prediction for the next trading action. This dataset produced the 80/20 training/testing split where various transformations were applied, including: First-order differencing, which subtracted the previous day value for the current one; Second-order differencing, which subtracted the value two days before the current one. The differencing help stabilise the means of consecutive values in time series data.

Similar to stock price, DJIA was also influenced by market sesonality, therefore, seasonal adjustment was applied to the dataset for producing Stationary values as one of the feature transformations.

Instead of using features of a single day for prediction, data was reconstructed to include features in the previous n days, where n is the window size. This experiment exploited the signal/pattern potentially hidden in a series, rather than a data point.

These transformations of dataset were fitted into models described in the previous section, and the results were explained below:

A screenshot of a cell phone

Description automatically generated

## **Value Differencing**

A screenshot of a cell phone

Description automatically generated

The results indicated that half of the eight models performed better with second order differencing and the average improvement is 0.90%. Among these, Random Forest gave a 2.61% of improvement with second order differencing. The remaining models showed an average of 1.16% decrement in their accuracy.

A close up of a logo

Description automatically generated

Considering datasets with NASDAQ and all currencies removed, the average improvement of taking a 2nd order differencing dropped to 0.80%; however, Random Forest gave a higher improvement of 3.01% with second order differencing. A stronger negative influence of 1.86% on Adaboost, KNN and Logistic Regression was observed. Among them, Adaboost’s performance was largely deteriorated with a 5.03%.

## **Features Removal**

Certain features (“X Features”) were removed from the dataset for, or with a view to reducing the potential “white noise” that reduced model accuracy; in this subsection, the NASDAQ and closing price of all currencies were excluded. A column that  emphasised the orderal property of the records was appended, since the date column was removed.

A close up of a logo

Description automatically generated

While taking first order differencing on the values, the removal of X Features did not show much impact on most models; only Adaboost and LSTM Regressor were reported to have +2.61% and -2.02% influence on their accuracies.

A close up of a logo

Description automatically generated

While applying data value with second order differencing, accuracies of Adaboost and KNN dropped by 2.62% and 1.81% respectively. Other models remained relatively stable against the removal of X Features.

Models behaved differently to feature transformations. From the result, features transformations appeared to have little influence on Forward Feed Neural Networks and Gradient Boosting, their variances in accuracy percentage were 0.00% and 0.06% respectively. Adaboost was found very sensitive to feature transformations, and the variance is approx. 4.22%.

## **Stationary Features**

A screenshot of a cell phone

Description automatically generated

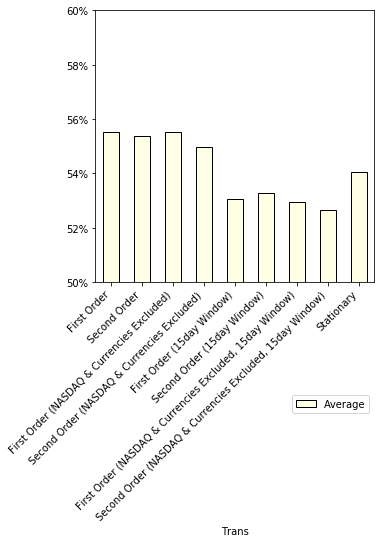
A close up of a logo

Description automatically generated

By fitting stationary features into the models, the accuracies obtained were lower than the average of other transformations. Adaboost achieved the highest accuracy of 59.80%, but the score was still 2.37% lower than the top runner. The removal of seasonality did not appear to help the prediction.

## **Additional Experiment**

Instead of fitting data of a particular day into the models, each data point was restructured to include data of the past n consecutive days. The number of features considered would become n times the original feature size, e.g. with a 15-day window, a data point expanded to contain 240 features from the original 16. From this experiment, however, the accuracies of the models dropped significantly; it was suspected that a larger window size may represent pattern for a long run better. Tuning of window size was the next topic to explore in the future.



# Conclusion \*\*FROM EACH RESULT CATEGORY PICK A WINNER MODEL AND DESCRIBE WHY IT WON\*\*<CLIFF>

The figure below summarised the accuracies each model obtained. Only the 5 transformation categories with significant accuracy were compared.

A close up of a logo

Description automatically generated

Adaboost outplayed other models across all data transformation categories. Even considering it’s worst prediction using data with second order value differencing, NASDAQ and all currencies column removed, the accuracy of 57.14% was still above all other models.

In this study, the predicted labels were discrete values of either BUY or SELL action - this value was disjointed and unordered; since Adaboost was an ensemble of multiple decision trees, this dichotomous nature matched the capability of such classifiers. The high accuracy of the model also indicated the potential drawback of overfitting; as one key objective of Adaboost was to minimise training error.

LSTM was designed to make predictions based on sequential data but it did not perform very well in our experiments. It was suspected that only daily closing values may not be sufficient for the neural network to predict the next trading action; and the sampling interval may overgeneralise meaningful signal and the volatility of the index throughout the day.

# Teamwork

|  |  |
| --- | --- |
| **Member** | **Contribution** |
| Danish Alsayed |  |
| Yuki Ng |  |
| Cliff Ng |  |
| Kabir Rajput |  |