Prediction of Stock Price Movement using K-Nearest Neighbours, Random Forest and Support Vector Machines Algorithms

Jamie Donnelly
MSc Data Science and Computational Intelligence
Faculty of Engineering, Environment and Computing
Coventry University
donnel39@uni.coventry.ac.uk

***Abstract* – The primary objective of this paper is to evaluate the effectiveness of various Machine Learning techniques to the problem of stock price prediction. I will be using the programming language Python to predict the future prices of Nvidia and Amazon stocks based on a dataset including features such as historical prices, technical indicators and the performance of common indexes. I will use classification techniques such Support Vector Machines, Random Forest and K-Nearest Neighbour to classify whether future stock returns shall be negative or positive.**

Keywords: Python, classification, SVM, random forest, KNN, prediction

I. INTRODUCTION

Stock price prediction, or predicting the direction of movement in stock prices, is an incredibly difficult, yet fiscally rewarding, challenge in Time Series modelling. This is in part a result of the sheer quantity of variables, both quantitative (e.g. economic, earnings or historical price data) and qualitative (e.g. political events or news stories), that can impact upon a stock's price. Furthermore, the stock market is essentially dynamic, non-linear, complex, non-parametric, and chaotic in nature [1].

Stock prices and, more generally, capital markets are highly volatile and for the most part unpredictable and thus for any parties operating in them there is an equal element of risk as well as reward. Therefore, the motivation for modelling stock price behaviour and/or making predictions about the future direction accurately is apparent. With greater information about future possibilities it could allow investors to earn higher rates of return or for financial institutions to more accurately forecast future risks and liabilities. The rest of this paper will be structured as follows. In section 2 I will review related relevant literature. Section 3 explains my research methodology. Section 4 will present my experimental setup. Section 5 will include my results and finally, Section 6 will include a discussion and conclusion.

II. LITERATURE REVIEW

The use of prediction algorithms to determine future trends in stock market prices contradicts a basic rule in finance known as the Efficient Market Hypothesis [2]. The Efficient Market Hypothesis is a theory that states that securities are priced accurately and reflect all available information. As part of this theory it's assumed that the movement of stock prices follow random walks and therefore are essentially unpredictable. Although the Efficient Market Hypothesis (EMH) states that it is not possible to anticipate market movements consistently, the use of computationally intensive systems that employ machine learning algorithms is increasingly common in the development of stock trading mechanisms [3].

Since the early 1990's, machine learning techniques such as Artificial Neural Networks have been used for research on the predictability of financial time series [4]. Numerous studies *(Phua et al. 2003, Chen et al. 2003, O'Connor and Madden 2006, Zhua et al. 2007)* have documented the successes and failures of ANNs in forecasting time series [5]. First published studies showed that ANNs were far superior to the traditional techniques used for forecasting such as ordinary least squares regression, logistic regression or discriminant analysis [5]. Artificial Neural Networks (ANNs) can model complex relationships between input and output data [6] which is most

likely the reason why they became popular in financial time series modelling.

Following this, alternative algorithms such as Support Vector Machines (SVM) have been used as well as ANNs. Recent studies tend to hybridize SVM with techniques such as robust feature selection and technical analysis factors [3]. *Kara et al. (2011)* conducted an experiment in which neural network and plain SVM models were compared for the prediction of stock price index movement with the extensive use of several technical indicators [3].

When it comes to the data used in predicting stock prices there are two main categories which can be used: *Fundamental data* or *Technical data*. Fundamentals usually related to a company's financial statements *(balance sheet, Income statement and cash flow statement)* or potentially wider economic data generally. While technical data usually relates only to historical price data or technical indicators.

III. METHODOLOGY

A. Dataset

My task is aimed with predicting whether the close price for a stock 30 days into the future on any given day is higher or lower than the previous day's close. My research focuses solely on using quantitative information for the features in my dataset. For each instance (1 day) of my data, as features I included data about the stock itself (previous day's open, close, high, low and volume. I have also included the previous 1-day and 5-day returns), data from indexes and *technical indicators*. All data has been sourced from *Yahoo Finance* and I have collected 15 years of data (for stocks and indexes) starting from September 2004 up to the present. On each day (dataset row), *i*, I will be trying to predict a target variable which is calculated as follows:

$$target_i = Sign(close_{i+30} - close_{i-1})$$

I'm trying to predict whether the closing price 30 days on from day i is higher or lower. When the value of $target_i$ is +1 (Class 1) this indicates the price has risen and -1 (Class 2) indicates a fall.

Table 1: Class distribution for stocks

Stock	*(%) Class 1 Results*
Nvidia	61.3
Amazon	64.2

Table 2: Market Index Features

Index	*Description*	*Symbol*
SP500	A market-capitalization weighted index of the 500 largest U.S. publicly traded companies	SPX
NASDAQ 100	A market index made up of 100 of the largest non-financial companies listed on the NASDAQ stock exchange	NDX
DAX	Is a stock market index consisting of 30 major German companies trading on the Frankfurt stock exchange	DAX
CBOE Volatility Index	Is a popular measure of the stock market's expectations of volatility implied by SP500 index options	VIX

Technical Indicators are important parameters that are calculated from time series stock data that aim to forecast financial market direction. They are tools which are widely used by investors to check for bearish or bullish signals [7]. I will now explain the technical indicators I have used and how they are calculated.

Moving Average

Simple moving averages (SMA) takes the arithmetic mean of a given set of prices over the past number of days, in this case 100 days [8]. It is used to smoothing out the price history.

$$MA = \frac{S_{t-1} + S_{t-2} + \cdots + S_{t-n}}{n}$$

Relative Strength Index

RSI is a popular momentum indicator which determines whether the stock is overbought or oversold. A stock is said to be overbought when the demand unjustifiably pushes the price upwards [7]. The formula for RSI is:

$$RSI = 100 - \frac{100}{1+RS}$$

$$RS = \frac{Average\ Gain\ Over\ past\ 14\ days}{Average\ Loss\ Over\ past\ 14\ days}$$

RSI values range from 0-100 with values below 30 deeming an asset *'oversold'* and above 70 as *'overbought'*.

Rolling Annualised Volatility

While this isn't a traditional technical indicator, I am including the stock's volatility in this category. For this I am using the 30-day rolling annualised volatility. This annualises (extrapolates to an annual

rate) the volatility of returns for the past 30 days. It is calculated as follows:

$$\sigma_{252} = \sqrt{252} \times \sqrt{\sigma^2} = \sqrt{252} \times \sqrt{\frac{\sum_{i=1}^{30} x_{t-i}}{30} - \mu^2}$$

Where,
$\mu = avg.return\ for\ 30\ days$

This value is always positive since it's the square root of the rolling-variance and so I do not encode it.

Stochastic Oscillator

Stochastic Oscillator follows the speed or the momentum of the price. As a rule, momentum changes before the price changes. It measures the level of the closing price relative to low-high range over a period of time [7]. The formula for calculating it is as follows:

$$\%K = 100 \times \frac{(S_t - L14)}{(H14 - L14)}$$

Williams %R

Williams %R values range from -100 to 0 and when the value is above -20 it's considered a sell signal and when it's below -80 a buy signal. It's calculated as follows:

$$\%R = \frac{(H14 - S_t)}{(H14 - L14)} \times -100$$

B. Feature Selection

My dataset consists of approximately 3800 observations with 16 features which is a reasonable size dataset. I therefore decided that it would be unnecessary to perform feature extraction to obtain a subset of only the most relevant features through a method such as Principle Component Analysis (PCA). When analysing the correlations between features it can also be found that there weren't enough significantly high coefficients ($\rho \geq 0.9$) to justify removing redundant features.

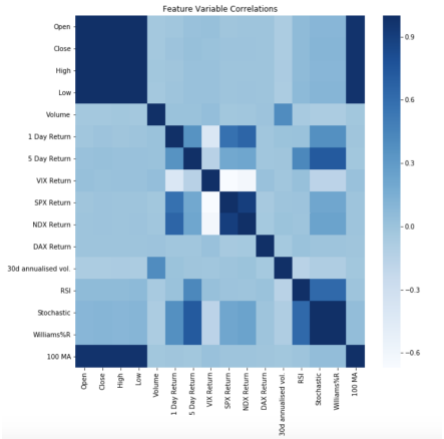

Figure 1: Correlation of feature variables

Furthermore, I have no issue with class imbalance for my response variable, so I have no need to make any adjustments in that respect.

C. Data Normalisation

I will be normalising the input data and the reason for doing this is so that all features are on the same scale and therefore aren't incorrectly weighted. Normalisation was performed using Python's StandardScaler class, performing the following on the data:

$$z = \frac{x - \bar{x}}{\sigma}$$

Where *z* is our new standardised data calculated by subtracting the mean value for that feature and dividing it by the standard deviation.

C. Prediction Algorithms – KNN

KNN is what we consider to be a *'lazy learning'* algorithm. Furthermore, it is a non-parametric technique and when you say a technique is *non-parametric*, it means that it does not make any assumptions on the underlying data distribution [9]. KNN works by calculating the *Euclidean distance* between all the instances and assigns a variable to a certain class based on the class of the K nearest neighbours, as measured by Euclidean distance. The choice of number of neighbours, K, will have an impact on the algorithm's effectiveness and the computational complexity. However, there is no accepted method of determining the optimal number for K. In terms of the *bias-variance* trade-off, we can say that low K results in high variance and low bias and vice versa for high K.

D. Random Forest

Decision trees are a technique commonly utilised in machine learning where the data is continually partitioned based on a given parameter. Decision trees have a low bias and a high variance since a slight noise in the data may cause the tree to grow in a completely different manner [7]. Random forest is a technique built off the idea of decision trees however it overcomes the bias-variance problem of trees by training multiple decision trees on different subspaces of the feature space at the cost of slightly increased bias [7]. The model then works on a 'voting' system whereby the final classification is determined by the most common classification among each tree in the model.

E. Support Vector Machines

Support vector machine (SVM) were first introduced by *Vapnik (1999)*. There are two main categories for support vector machines: support vector

classification (SVC) and support vector regression (SVR) [10]. The idea behind SVMs is to map data onto a high dimensional space via a *kernel function* where data can then be classified with linear decision surfaces. SVM then performs classification by identifying a maximum margin of separation between the classes, a hyperplane or a line (for a two-dimensional case) [10]. Once a hyperplane has been established then instances are classified by means of assigning them to one of two disjoint half spaces, either in the pattern space or in a higher-dimensional feature space [10].

IV. Experimental Setup

Before presenting and commenting upon my results I will explain a bit about how I put the chosen algorithms and my data into practical use. When it came to test the efficacy of my algorithms, I decided to use the *cross-validation*, or *k-fold cross validation*, method of testing as an alternative to simply splitting my data into 2 parts – training data and testing data. This is where the dataset is randomly split in *k* subsets of equal size (*In my case I am using k=10)*. The data then uses 1 'fold' of the data for testing and the remainder for training. Iterating *k* times changing the test fold each time and then averaging the performance of each iteration:

Figure 2: Illustration of Cross-Validation [11]

Additionally, I feel I should explain the choice of parameters I have used in my respective algorithms since the parameters will ultimately effect performance. In the case of KNN I decided to iterate and measure the classification performance over the range $k = [1,2,\dots,100]$

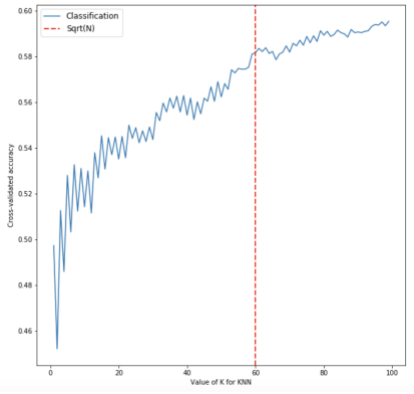

Figure 3: KNN Classification Rate

As this plot shows, the algorithm's accuracy improves as the number of neighbours considered increases. However, beyond a point, approximately around 65 onwards, we notice diminishing returns in the increases in accuracy. A common practice is to use $k = \sqrt{n}$, where *n* is the amount of data and so I have indicated where that is in this plot. I have decided to settle to use this value of 60 since above that I am only increasing the computational complexity for marginal increases in accuracy.

Next, in the case of the Random Forest algorithm I chose to optimise the parameters for the number of trees and the maximum depth of the tree. I have used a range of 1-100 for number of trees and 1-3 for the maximum depth. Then using the *GridSearchCV* class in Python to find the best performing parameters out of those options, I obtained that 20 trees with a maximum depth of 1 performed best.

Finally, in the case of my Support Vector Classifier I needed to decide on the choice of kernel function, value for the penalty parameter, C, and the margin, γ. I tested *rbf* and *linear* kernel functions and used the better performing one. I would then trial values C = [1, 5, 10, 20, 100] and $\gamma = [0.0001, 0.001, 0.01, 0.1, 1]$. The choice of the parameter *C* decides whether model will aim to maximise the margin or minimise misclassification. The margin gamma, γ, is defined as the distance of the closest sample from the hyperplane [12]. Again, using *GridSearchCV* class, the optimal choice was to employ the use of *rbf* kernel function and set *C* and *gamma* to *1* and *0.001*, respectively.

V. Results

When assessing the algorithms, I am only going to use the *out-of-sample* or *testing* classification rate as a measure of accuracy:

Table 3: Classification Rates

Classification Algorithm	*Nvidia Accuracy*	*Amazon Accuracy*
KNN	58.9%	60.6%
Random Forest	59.7%	61%
Support Vector Machine	61.3%	64.2%

These results show that in the case of both stocks, the Support Vector Machine algorithm is the most accurate as correctly classifying future stock direction with 61.3% accuracy for Nvidia and 64.2% accuracy for Amazon. In both cases KNN performs the worst however, the differences between that and Random Forest are negligible.

While not a measure of the accuracy of an algorithm at prediction, I thought it would still be informative

to measure and compare the runtimes of each algorithm. All programming was completed in Python (using *Jupyter Notebook)* with the following computer specifications: 2.3 GHz Intel Core i5 processor, 128GB SSD and 8 GB 2133 MHz LPDDR3 RAM. The times are as follows:

Table 4: Computation speeds

Classification Algorithm	*Algorithm Speed (seconds)*
KNN	0.46
Random Forest	0.73
Support Vector Machine	4.2

This shows that the time taken to for the SVM algorithm is considerably greater than the others, with nearly a 10x longer runtime than KNN, the fastest. While SVM is the most accurate, in this case it's also the most computationally complex, as demonstrated by runtime, and may be very slow performing in cases with extremely large datasets in comparison to algorithms with lower complexity.

VI. Discussion and Conclusion

As previously stated, the prediction of stock prices or the direction of price movement is one of the hardest challenges in time series modelling. Feature selection poses a big problem, and is one I faced, as the number of potential variables that can influence the price of a stock are effectively unlimited – there is no predefined, dataset for the problem like there might be with other problems that Machine Learning aims to solve. Furthermore, many theorists will argue that financial markets at their foundation are ultimately random, most prices are simply random walks or that returns data is essentially *white noise* and hence unpredictable. Whether this is true or not, there definitely does exist a large element of unpredictability – the inevitable result of the actions of thousands of individuals and corporations driven by varying motivations and beliefs.

In my task I managed to achieve ~60% accuracy in the task of predicting whether the price of a stock would be higher or lower in 30 days' time for any given data. Which, while not objectively that impressive, in the context of the problem should be evaluated as a success.

If I were to pursue this line of work further, I would look to include a wider array of very different feature variables such as fundamental information about the underlying stock or the economy as a whole. Perhaps with a larger dataset such as that I could identify the most important variables in the task of stock prediction or obtain more accurate predictions.

VII. Appendix

The Python code and the original datasets can be found at: https://livecoventryac-my.sharepoint.com/:f:/g/personal/donnel39_uni_coventry_ac_uk/Ehvikk_-oAJDkpV38YUfodkBfRZkM6bEBlVLZLXtbC0FCQ?e=eclwOV

VIII. References

1. Tan, Tuan Zea & Quek, Chai & Ng, Geok. (2005). *Brain-inspired genetic complementary learning for stock market prediction*.
2. Malkiel, B. G. and Fama, E. F.(1970). *Efficient capital markets: A review of theory and empirical work*. The Journal of Finance, 25, 383-417.
3. Henrique, Bruno & Sobreiro, Vinicius & Kimura, Herbert. (2018). *Stock Price Prediction Using Support Vector Regression on Daily and Up to the Minute Prices*. The Journal of Finance and Data Science.
4. Lachiheb, Oussama & Gouider, Mohamed. (2018). *A hierarchical Deep neural network design for stock returns prediction*. Procedia Computer Science. 126. 264-272.
5. Olson, Dennis & Mossman, Charles. (2003). Mossman, C.: *Neural Network Forecasts of Canadian Stock Returns using Accounting Ratios*. International Journal of Forecasting. 19, 453-465. International Journal of Forecasting. 19. 453-465
6. Jabbari, Aida & Bae, Deg-Hyo. (2018). *Application of Artificial Neural Networks for Accuracy Enhancements of Real-Time Flood Forecasting in the Imjin Basin*. Water. 10. 1626. 10.3390/w10111626.
7. Khaidem, Luckyson & Saha, Snehanshu & Basak, Suryoday & Kar, Saibal & Dey, Sudeepa. (2016). *Predicting the direction of stock market prices using random forest*.
8. Hayes, A. (2019). *Understanding Moving Averages (MA)*. [online] Investopedia. Available at: https://www.investopedia.com/terms/m/movingaverage.asp [Accessed 4 Oct. 2019].
9. Thirumuruganathan, S. (2010). *A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm*. [online] God, Your Book Is Great !!. Available at: https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/ [Accessed 4 Oct. 2019].
10. J. Patel, S. Shah and P. Thakkar, *Predicting stock and stock price index movement using*

Trend Deterministic Data Preparation and machine learning techniques, Expert Systems with Applications, vol. 42, pp. 259-268, 2015.
11. Rosaen, K. (2016). *Scikit-learn Pipeline gotchas, k-fold cross-validation, hyperparameter tuning and improving my score on Kaggle's Forest Cover Type Competition | ML Learning Log*. [online] Karlrosaen.com. Available at: http://karlrosaen.com/ml/learning-log/2016-06-20/ [Accessed 4 Oct. 2019].
12. Negrini de Araujo, J. (2018). Application of machine learning techniques for stock price direction forecasting.