MPhil in Advanced Computer Science

Stock market prediction

Matteo Bettini
Sidney Sussex College

UNIVERSITY OF CAMBRIDGE

A project report submitted to the University of Cambridge for the course: L330 Data Science: principles and practice

Word count: 4959

TeXcount was used as word-counting tool

Matteo Bettini - mb2389@cam.ac.uk - December, 2020

Contents

List of Figures

1 Introduction

This is the report for the final project of the course: L330 Data Science: principles and practice.

The task for this project is stock market prediction using a diverse set of variables. In particular, given a dataset representing days of trading in one stock market, our aim is to predict the daily movement of the market up or down conditioned on the values of variables in the dataset over the previous N (trading) days. Therefore, our prediction task is binary classification.

The datasets we use for this purpose are: "Stock market prediction using a diverse set of variables[1]". The repository contains data of five major U.S. stock markets. For this project we just focus on the NASDAQ Composite dataset.

This work is based on insights and considerations gained from the paper "CNNPred: CNN-based stock market prediction using several data sources" [1], where the authors build two CNN models for predicting the movement of stock markets.

A Jupyter notebook illustrating all the work done has been submitted with this report. It provides a step by step walkthrough of the code and the algorithms used in the project.

This report is structured in the following way: in **Section 2** we briefly describe the importance of stock market prediction and some existing work in this area, in **Section 3** we start exploring the dataset, looking at the features and their correlation to then prepare the dataset for the machine learning algorithms, **Section 4** explores dimensionality reduction as a technique to visualise our data, **Section 5** investigates different models and approaches to the classification task (e.g., traditional machine learning, neural networks) comparing their performance and introduces a novel model that uses convolutional and long short-term memory (LSTM) layers, finally, **Section 6** concludes this report.

2 Background

Stock markets are at the heart of today's economy. Every day billions of dollars are exchanged through trading all over the world in various markets. Given this context, stock markets' index movement prediction is a fundamental challenge. Predicting the movement of a market could help millions of investors make more informed decisions when trading. However, achieving

[1]Dataset repository `https://archive.ics.uci.edu/ml/datasets/CNNpred%3A+CNN-based+stock+market+prediction+using+a+diverse+set+of+variables`

good prediction quality for stock markets is a major challenge. This is because the financial time series of stock market indices is intrinsically dynamic, nonlinear, complex, noisy, and chaotic [2, 3, 4].

According to the Efficient Market Hypothesis (EMH) [5] stock price is just a function of all the current information we have about stocks and markets and, therefore, it is impossible to predict future stock prices accurately. In the beginning of the 21st century, however, some economists indicated that future stock prices are at least partially predictable [5]. This has enabled researchers to apply prediction techniques form the field of machine learning to financial time series.

Stock market index movement is influenced by many factors: from economic indicators to currency exchange rates, from the price of commodities to the indices of other markets. Therefore there has been a lot of research in investigating which features to use and how to structure the prediction task [6]. Artificial Neural Networks (ANN) are a very popular model used for this purpose [7], they have been applied to the NASDAQ market to perform regression [8].

Different ANN architectures have been explored. Recently Convolutional Neural Networks (CNNs) have been proven to efficiently perform feature selection and prediction [1]. This has been done by using data from different markets and developing custom convolutional layers specifically designed for the stock market features. In [9] the authors compare CNN, Recurrent Neural Networks (RNN) and Multi Layer Perceptrons (MLP) for stock market movement prediction. The results show that CNNs outperform the other architectures.

In this work we are going to compare different traditional machine learning techniques and various neural network architectures for predicting the movement of the NASDAQ stock market. We will also introduce a novel architecture that puts together CNN and LSTM layers.

3 Dataset exploration

The dataset used for our task contains days of trading in the NASDAQ Composite stock market from 2010 to 2017. It is composed of 1984 entries each representing a working day. Every entry is indexed by date and is characterised by 83 features which are grouped in the following way:

- **Primitive features**: these are the close price of the market and the day of the week
- **Technical indicators**: historical data of stocks used by analysts in stock market research

- **Economic data**: economic data of the country
- **World stock markets**: returns of stocks markets all over the world
- **The exchange rate of U.S. dollar**: exchange rate of various foreign currencies with the U.S. dollar
- **Commodities**: information relating to the price of commodities
- **Big U.S. Companies**: stock market value of big U.S. companies
- **Futures contracts**: features describing future contracts about trades that are expected to take place

The authors of [1] have made available five datasets, each representing a different stock market. The available markets are: S&P 500, NASDAQ Composite, Dow Jones Industrial Average, RUSSELL 2000, and NYSE Composite. In this work we explore and analyse the NASDAQ Composite dataset, but all the insights we are going to gain are be valid for all datasets. The primitive features and the technical indicators are unique for each dataset, while all the other features are common among datasets. A tabular description of the features is also reported in the appendix of the paper [1]. We note that with respect to that description we have and additional feature representing the relative change of wheat price.

After uploading the dataset, we start by reordering the features to group them into the described categories. We index the data by date and add to the dataset the categorical feature representing the day of the week. For this purpose we use a natural number. We do not use a one-hot encoding as we want our models to learn that numerical values that are close to each other encode similar concepts.

The features in which we are interested the most are "Close" and "mom", representing respectively the close price and the return of the market. Where the return of a market at time t is computed in the following way:

$$Return_t = mom_t = \frac{Close_t}{Close_{t-1}} - 1$$

These features represent the trend of the market, we will use them to gain some insights on correlations among the data and to build our classification target.

3.1 Time series analysis

Differently form the other datasets seen during the course, this dataset represents a time series.

A time series is a sequence of data sampled in a discrete time interval. In

our case the series represents data at a daily granularity. The data is indexed by time as its order is very important. Data is correlated with time, features at a certain time t are closely influenced by themselves from previous days. We will look into recognizing trends, periodic behaviors and correlation using time series analysis techniques.

Our dataset represents a multivariate time series. This means that it contains more than one time-dependent feature. Each feature not only depends on its past values but also on other features and their past values.

The goal is to predict the movement direction at a certain time t using information from the previous N time steps in the series. We choose $N = 60$ in accordance with [1] after some trial and error tuning.

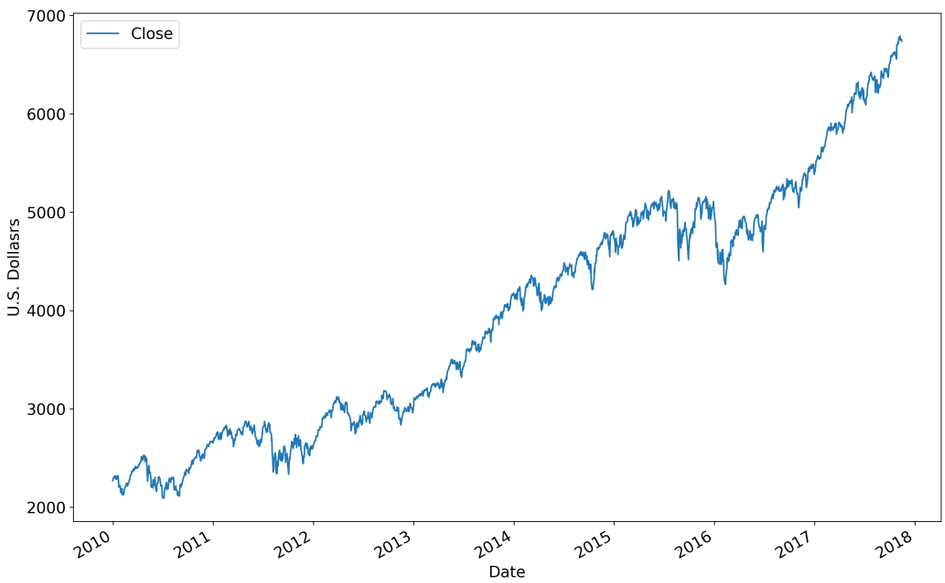


Figure 1: The time series of NASDAQ Close price

It is interesting to see the evolution of some features through time. In Fig. 1 we can see the evolution over the years of the close price of the NASDAQ market. Another interesting plot to look at is the comparison between the return series from the four main U.S. stock markets, this is shown in Fig. 2. As we can see the returns are closely correlated and share a similar trend. One example is the big oscillation we observe between 2011 and 2012.

We also want to look into the comparison of NASDAQ return among different years. This is shown inf Fig. 3. As for different years we have a different number of data points, here we compare only the years that have the same amount of data points and for which the amount corresponds to the maximum. As we can see in the figure, there is a slight correlation of return among different years. This can be seen, for example, between 2013 and 2014. But this correlation is not clearly marked and the plot suggests that return does not strongly depend on the time of the year.

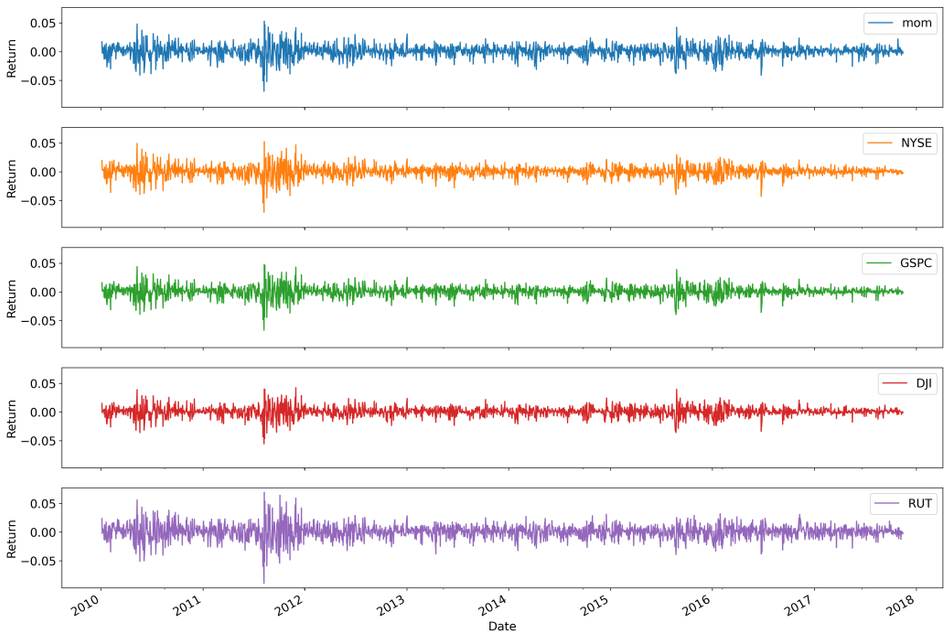


Figure 2: The return of the main U.S. stock markets from 2010 to 2018. From top to bottom: NASDAQ Composite, NYSE Composite, S&P 500, Dow Jones Industrial Average and RUSSELL 2000

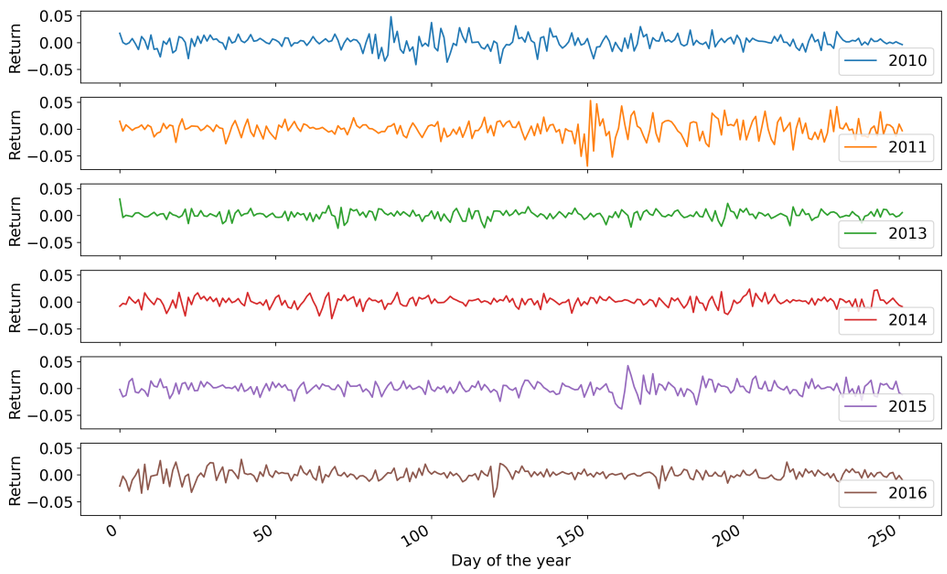


Figure 3: The return of the NASDAQ stock market compared among different years between 2010 and 2018, the x axis, labeled as "Day of the year", refers to working days

We report also a whisker plot for the median return of the NASDAQ market over different years. This is shown in Fig. 4. The drawn box encloses data between the 25^{th} and 75^{th} percentiles, capturing the middle 50% of the observations. The middle line represents the median and whiskers are drawn above and below the box to outline the extent of the observations. Dots outside the whiskers represent outliers.

Finally, let's plot the distribution of the features. As we can see in Fig.

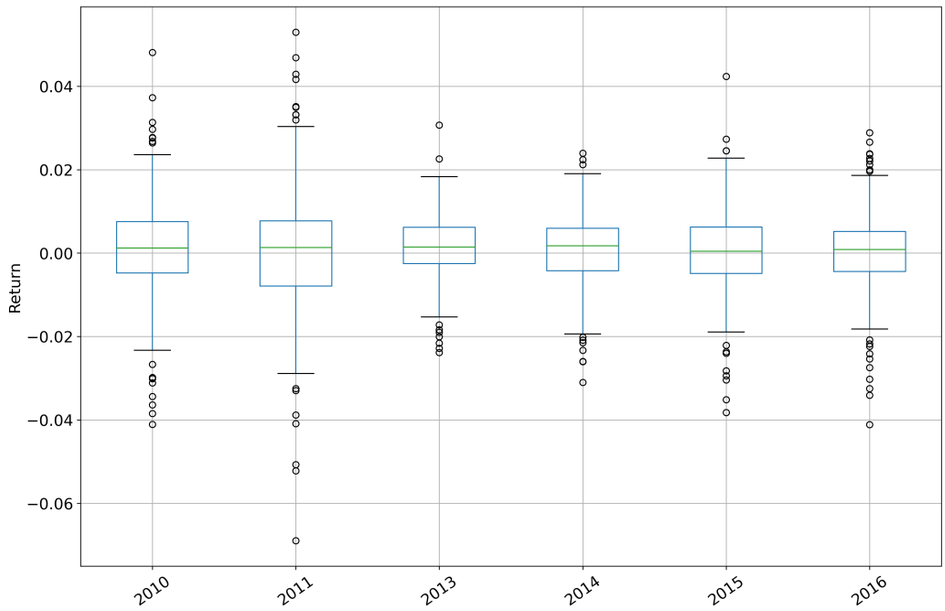


Figure 4: The average return of the NASDAQ stock market in different years between 2010 and 2018

5, the great majority of features is distributed as a Gaussian centered in 0, with some exceptions in the technical and economic indicators.

3.2 Splitting the data: training, validation and test

As already mentioned, our dataset represents a time series. Therefore, random or stratified train/test split strategies cannot be used as it would not enable the machine learning algorithm to benefit from the fact that the data are temporally correlated with each other. Test and validation indices must always be greater than training indices.

Given this considerations, the we choose to split the dataset with a traditional splitting strategy:

- The first 60% of the entries will form the **training set**
- The subsequent 20% will form the **validation set**
- The last 20% will form the **test set**

Before performing the split we also compute the target vector. For each entry, the target is 1 if there is an increase in the Close price in the following entry and 0 otherwise. Note that the target value is also equivalent to the sign of the return on the next day.

$$Target_t = \begin{cases} 1, & \text{if } Close_{t+1} > Close_t \\ 0, & \text{otherwise} \end{cases}$$

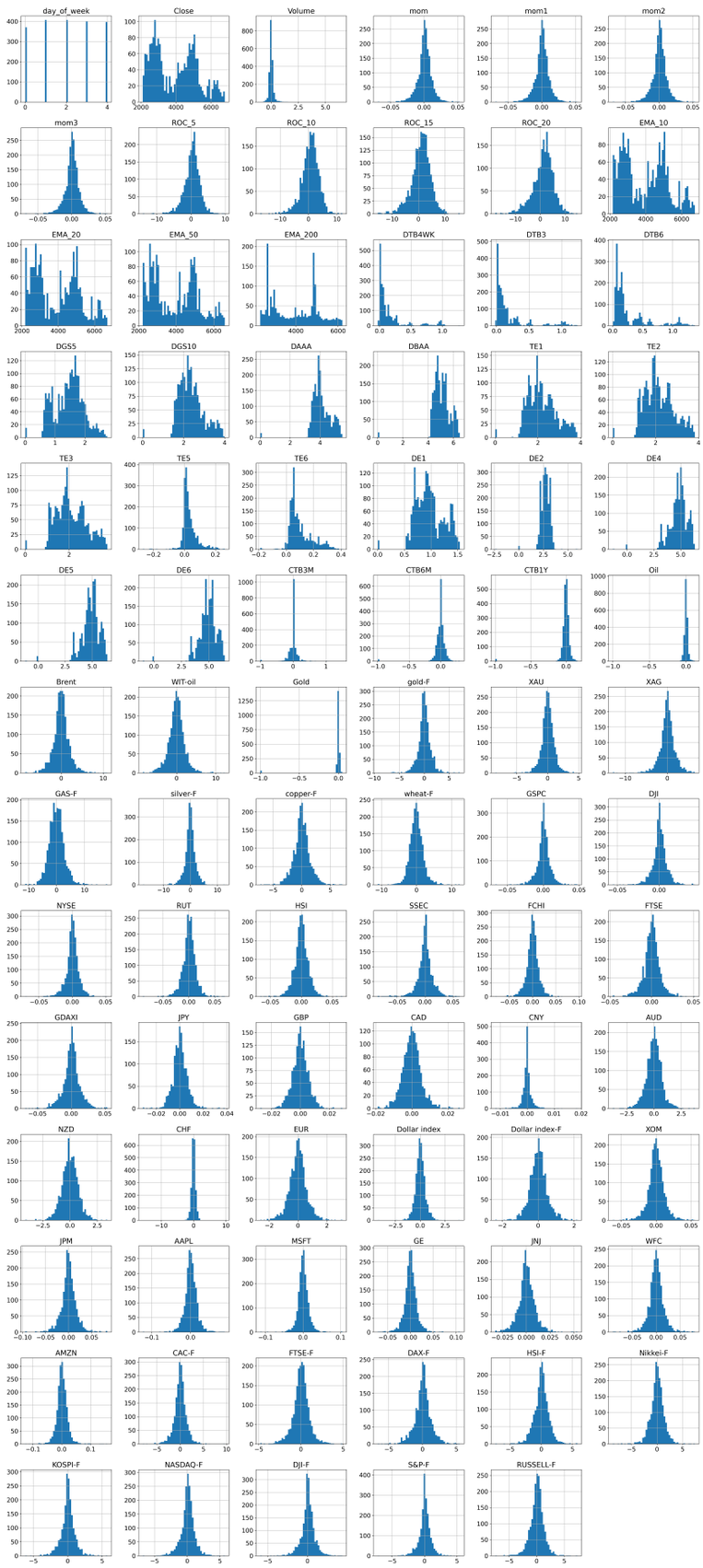


Figure 5: Histogram representing the distribution of the features in the dataset

The sets are still in the form of a time series. We will proceed to make them suitable for a supervised learning task in section 3.4.4.

Given the nature of the data it is very difficult to split the dataset properly. This is because the data in the time series is closely correlated with each other and with time. Furthermore, certain features are calculated using past training days and go back even 200 days in the data (like EMA_200). Therefore, it is impossible to ensure that the training, validation and test set will be independent from each other. Our evaluations will always be a bit biased. In particular, the validation set will be dependent on training data and the test set will be biased by the other two sets. This is caused both by the fact that the data we are analysing is a time series and by the fact that some features take into account trading days in the past.

To ensure the least interdependence between the sets, we treat them in the following way: after the split, we discard the last entry in the test set as we have no target for it. We also remark that the last target in the training set is computed using information from the validation set, but the bias that this introduces is very little. Then, when we modify the sets for our task (in section 3.4.4), for each set, we will discard the first $N-1$ targets (where N is the number of previous training days considered for the prediction), as those are the targets that come from input that is partially belonging to other sets or not available.

After the split we obtain a training set of 1189 entries and a validation and test set both containing 397 entries. We will also keep a training/validation set (composed by the union of the two) of 1586 entries that we will use for studying correlations and for cross-validation.

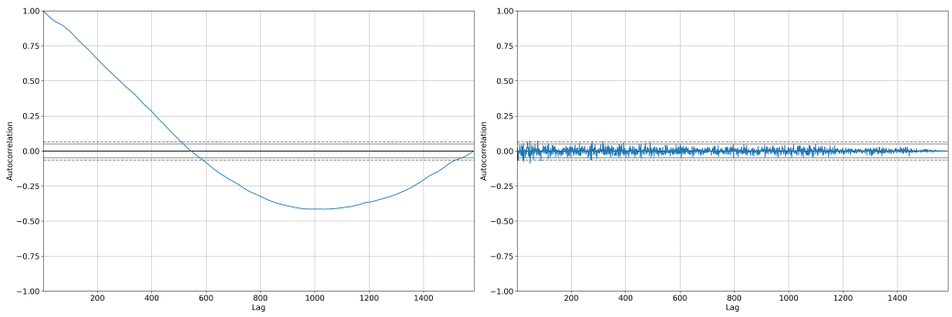
3.3 Correlations

In this section we investigate correlations between the data. For this purpose we use the training/validation set. By doing this we ensure no data leakage between the training/validation set and test set. The results on the test set will thus be a fair evaluation the algorithm's performance.

3.3.1 Autocorrelation

The autocorrelation plot is a way of measuring and explaining the internal association between observations in a time series. We can check how strong an internal correlation is in a given amount of time. The autocorrelation plot shows how much a value in the time series is correlated with itself n timesteps in the past. n is called lag. The dotted lines are just above and below the first quartile, or within the 95% confidence interval. If the line is above or below the dotted line we can say that the correlation is significant.

The autocorrelation plots of close price and return are reported in Fig. 6.



(a) Autocorrelation of close price *(b) Autocorrelation of return*

Figure 6: Autocorrelation

In Fig. 6a we can see that the close price is highly correlated with itself in the first part of the graph. The correlation decreases with the increase of the lag as we can expect. This is intuitive as the close price is a variable that we expect to be correlated with itself for small lags as it cannot drastically oscillate. There is an interesting negative correlation in the second half of the plot. This could suggest a trend of stock markets to rebalance themselves every three-four years. It could also be an effect of the economic crisis. Fig 6b shows that there is no strong correlation between the return of NASDAQ and time.

3.3.2 Correlation between the features

We are also interested in analyzing the correlation between the features. In particular, we look into how close price and return correlate with the other features and highlight the strongest correlations.

We have identified a high correlation between close price and the exponential moving average features, depicted in Fig. 7. This is not surprising as the moving average is used to smooth out short term fluctuations, it is an indicator of the trend of the close price.

Another important correlation worth highlighting is the one between the return of different markets, shown in Fig. 8. As we can see NASDAQ return correlates strongly with its futures and with the returns of other stock markets. This suggests that returns of other stock markets are important features for our prediction task.

3.4 Data transformation

Now that we have explored the attributes and their correlations, we can transform our data to prepare it for the machine learning algorithms.

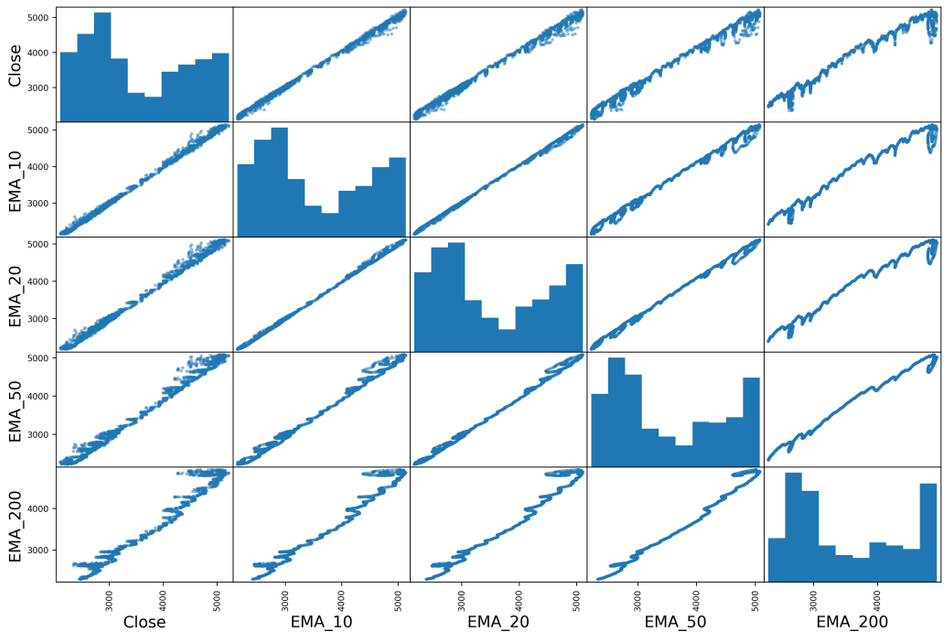


Figure 7: Correlation between NASDAQ close price and exponential moving average (EMA)

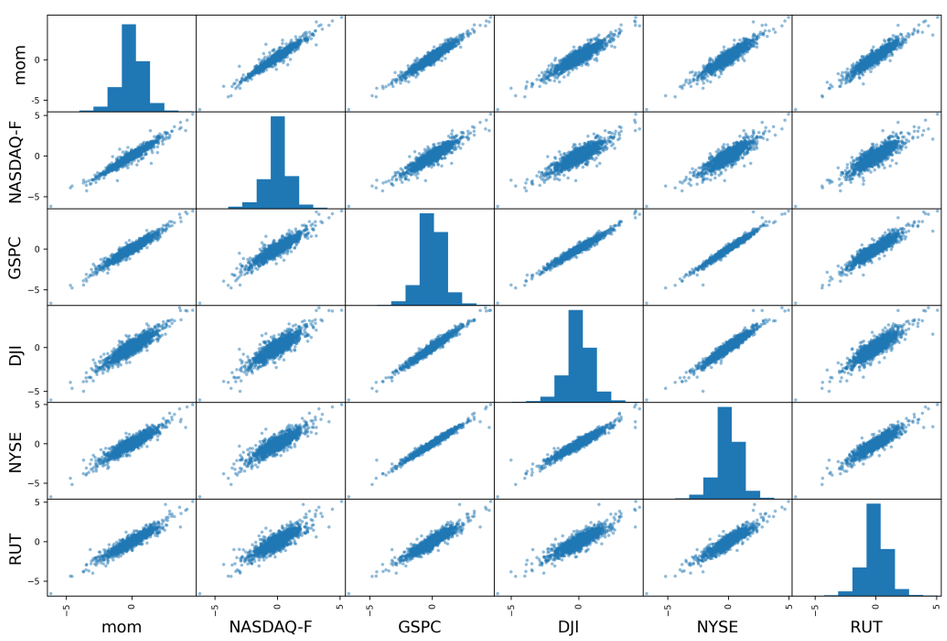


Figure 8: Correlation between NASDAQ return and NASDAQ futures with return of other stock markets

3.4.1 Missing values

In our dataset there are some missing values. Therefore we need to define a strategy to treat them. We do not want to throw away valuable data, we instead want to fill the NaN values with reasonable guesses. Given the fact that we are working with a time series, filling them with statistics (e.g., median) computed over all the values would not be reasonable as our data is time-dependent. Therefore, we decide to interpolate the missing values

using the time index. Then, to fill the remaining missing values in the first rows of the dataset, we perform backwards filling. This way we propagate backwards the first valid value to fill the remaining missing values.

3.4.2 Categorical features

In the dataset we have just one categorical feature, which is the day of the week. It is already expressed as a natural number, with the following mapping:

- $0 \rightarrow Monday$
- $1 \rightarrow Tuesday$
- $2 \rightarrow Wednesday$
- $3 \rightarrow Thursday$
- $4 \rightarrow Friday$

In this case we do not need to map it to a one-hot encoding, as we want the algorithm to benefit from the similarity between subsequent days given by this encoding. Note that we never have data on Saturdays and Sundays, as in those days the stock market is closed.

3.4.3 Feature scaling

Finally, we want to standardise our features as they have different ranges. To map them to similar ranges we preform *standardisation*, which subtracts the mean value (so the standardised values have a zero mean) and divides by the variance (so the standardised values have unit variance). This is more suited to our task than *MinMaxScaling* as the majority of our features are distributed as Gaussians.

3.4.4 Transforming the dataset for the prediction task

Now that we have prepared our data, we can build the dataset for our prediction task. We want to predict our targets using $N * 83$ values as input, where N is the previous number of days considered and 83 is the number of features. To do this, we have to build a new dataset using a sliding window of dimension N. We implemented a function that takes as input the time series dataset and outputs a new dataset containing for each row $N * 83$ values. These represent the values of the features for all the N previous training days of each target. As mentioned in section 3.2, for each dataset we drop the first $N - 1$ targets, corresponding to the number of windows that do not contain all the features of the N previous days.

We convert the training, validation and test set. We also convert training/validation set. This set is bigger than the union of the converted training

and validation sets. This is because when we convert it we do not need to throw away the first $N-1$ entries of the validation set.

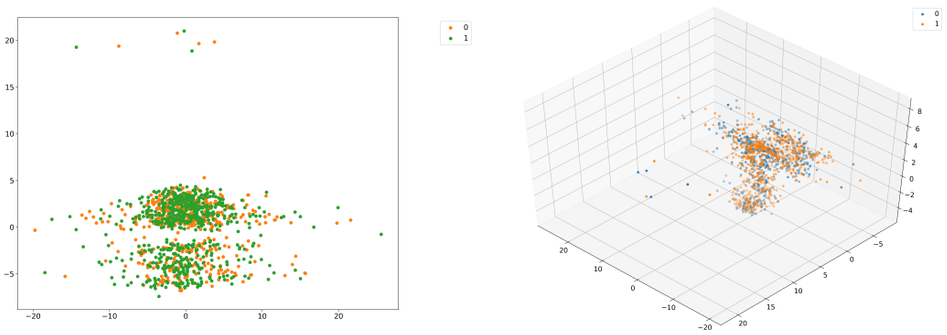
For the detailed implementation of this crucial step please refer to the attached notebook.

4 Visualisation and dimensionality reduction

Now we look into visualising the data. Having 83 features, the data has a relatively high dimensionality. Thanks to Principal Component Analysis (PCA) and t-SNE we can visualise and plot the data in less dimensions. We could also use the results obtained from the PCA to train our models on less features and save training time. However, the training time of our models is relatively short and we obtained the best results by using all the original features.

4.1 PCA

In Fig. 9 we plot the principal 2 and 3 components obtained by applying PCA to the training set.



(a) Principal 2 components

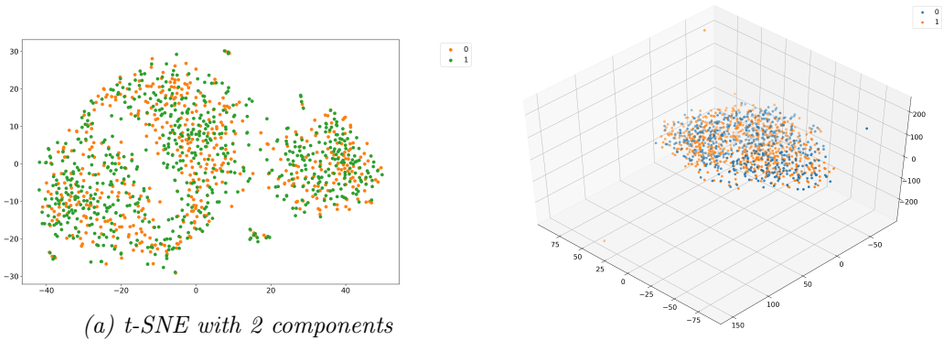
(b) Principal 3 components

Figure 9: Principal component analysis

4.2 t-SNE

In Fig. 10 we plot the principal 2 and 3 components obtained by applying t-SNE to the training set with a perplexity of 30.

More investigation on this methods is reported in the attached notebook. The reported visualisations show data that is still grouped randomly. Therefore we will run our learning algorithms on the dataset with all the features. Feature extraction will be performed by the neural network models.



(a) t-SNE with 2 components

(b) t-SNE with 3 components

Figure 10: t-SNE

5 Machine learning algorithms

Now that we have prepared the data for our machine learning algorithms, we are going to train different classifiers and see how they perform. We look both into traditional classifiers such as: logistic regression, naive bayes and perceptron and neural networks such as: convolutional neural networks (CNN), recurrent neural networks (RNN) and multi layer perceptron (MLP). For the traditional classifiers we use an input vector with $60 * 83 = 4980$ features while, for the neural networks, we will have an input tensor of shape $60 * 83$ for each datapoint.

We start by defining the metrics used for the evaluation of our algorithms.

5.1 Evaluation

Important: the metrics we will show for the neural networks are all computed as an average after initialising and training the same model 20 times. This is done to provide reliable results by partially factoring out the stochasticity introduced by the random initialisation of the weights in the models. The only exception is the ROC curve which will be shown for the best performing models (5.8).

We will evaluate our models on the following metrics:

- **Accuracy** represents the proportion of correct classifications our model makes over all the samples. In order to maximise accuracy, you'll need to maximise on the number of true positives and true negatives.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision** measures how reliable or trustworthy your classifier is. It tells you how often when the classifier predicts that the stock market close price will increase in the next day (class 1) it actually increases.

It relies on the number of TP's and FP's:

$$P = \frac{TP}{TP + FP}$$

- **Recall** measures the coverage of your classifier. It tells you how many of the actual increases your classifier can detect at all. It relies on the number of TP's and FN's:

$$R = \frac{TP}{TP + FN}$$

- **F_1-score** combines the two measures above to give you an overall idea of your classifier's performance. F_1-score is estimated as follows:

$$F_1 = 2 \times \frac{P \times R}{P + R}$$

In the paper [1] the authors select Macro-Averaged-F_1-Measure as their evaluation metric. This is equal to the unweighted mean of the F_1 scores between the two classes. Their choice is guided by the fact that accuracy may be biased towards models that tend to predict the more frequent class. We agree with this statement. For all our models we use the metrics just described in their *weighted* version. *Weighted* is similar to macro, with the difference that the resulting metric is a weighted average of the metric for the two classes. For the final comparison we use accuracy and Weighted-F_1-Score.

To see how our models perform, we also look into:

- the **Confusion matrix**, that represents the number of TP, TN, FP, FN in a matrix, this can be found in the attached notebook:

	predicted class 0	predicted class 1
actual class 0	TN	FP
actual class 1	FN	TP

- the **Receiver Operating Characteristic (ROC)** (when possible to plot) that shows the true positive rate (TPR) against the false positive rate (FPR) at various discrimination thresholds. Reported in section 5.8.

5.2 Logistic regression

We train a logistic regression classifier using an adaptive[2] learning rate with a starting value of 0.4. The classifier is able to correctly predict all the

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html`

instances in the training set.

5.2.1 Cross-validation

We then run cross-validation using the training/validation set. Given the fact that our input is a time series, we cannot perform stratified K-fold or normal K-fold cross-validation as they would not preserve the time order (test indices must always be greater than training indices), instead we just split the time series incrementally in 5 groups[3]. On cross-validation our model achieves an accuracy of 0.496 and a weighted-F_1-score of 0.492. The poorness of this result is somewhat explained by the cross-validation split. Time series splits yield training sets with an increasing number of datapoints. Therefore, some models are trained on very few data.

Finally, in Table 1 we report the results of this model on the validation set. Note that we evaluate and compare our models on the validation set and only in section 6 we see their performance on the test set.

Metrics	Accuracy	F_1 weighted	Precision weighted	Recall weighted
Validation	0.5740	0.5723	0.5726	0.5740

Table 1: Logistic regression validation results

5.3 Voting classifier

Now we try to use an ensemble of 4 classifiers, the used classifiers are:

- Gaussian Naive Bayes
- Logistic Regression
- the Perceptron
- Support Vector Machine (SVM)

We use hard voting as the SGD classifier used to model the perceptron does not provide probability estimates. All the models are trained using default sklearn parameters apart from logistic regression which uses the parameters described in section 5.2. For naive bayes we infer the prior over the classes' probability distributions directly from the data. The classifier is able to correctly predict all the instances in the training set.

On the cross-validation explained in section 5.2.1 it achieves an accuracy of 0.498 and a weighted-F_1-score of 0.493, slightly better than logistic regression.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html`

When looking at the metrics on the validation set we see that only logistic regression and the perceptron obtain fairly good results. However, even if they are all trained on the same set, the ensemble yields a significant improvement. We report the results of our model on the validation set in Table 2.

Metrics	Accuracy	F_1 weighted	Precision weighted	Recall weighted
Validation	0.5947	0.5922	0.5934	0.5947

Table 2: Hard voting classifier validation results

5.4 2D-CNNPred

In this section we try to recreate the 2D-CNNPred model from [1]. We have made autonomous choices for the parameters not described in the paper. Our loss function is the binary cross-entropy. We use early stopping on the validation loss with a patience of 4 epochs[4] and the maximum number of training epochs is 20. For all the other parameters' values please refer to [1]. In Fig. 11 we report the architecture of the network.

With respect to the authors' model, our model is trained only on one stock market. Therefore, comparison with their results would be pointless.

In Table 3 we report the results of this model on the validation set.

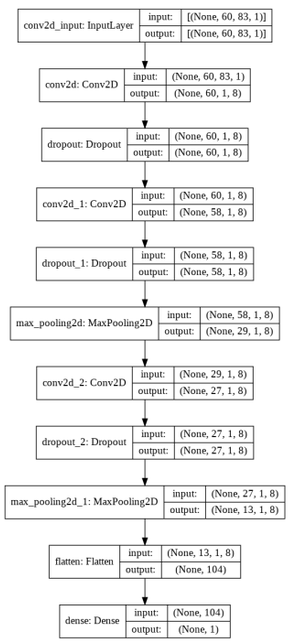


Figure 11: 2D-CNNPred architecture

Metrics	Accuracy	F_1 weighted	Precision weighted	Recall weighted
Validation	0.5260	0.4906	0.5159	0.5260

Table 3: 2D-CNNPred validation results, averaged between 20 trained models

[4] `https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping`

5.5 LSTM

Long short-term memory cells are able to learn what to remember and what to forget from previously seen data. They are good for predicting the future. For this reason, it seems reasonable to apply them to time series prediction. They are able to model the correlations in the time series and predict the next step. The LSTM layer is fed a $N * 83$ vector, with N the number of days. It analyses incrementally each of the N days and provides one final output. The output is then converted in a vector containing the probabilities of classes 0 and 1. The loss function used is sparse categorical cross-entropy. This configuration of LSTM cells is called sequence-to-vector.

As we can see in fig. 12, the network is composed of a first layer of 61 LSTM cells with "*tanh*" as activation function and a second dense layer composed by two neurons that output the probabilities of the binary classes. For this reason, the activation function of the second layer is "*softmax*". During training, we use early stopping on the validation loss with a patience of 2 epochs and 20 as the maximum number of training epochs. We use a batch size of 32. The optimizer is stochastic gradient descent (SGD) with a learning rate of 0.08. The optimizer, learning rate and number of LSTM cells have all been chosen using hyperparameter search.

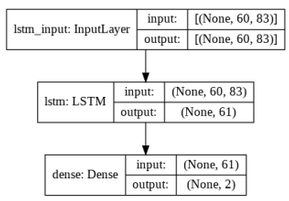


Figure 12: LSTM architecture

5.5.1 Hyperparameter search with HParams

For this model we perform hyperparameter search using the HParams plugin of tensorboard[5]. Tuning is performed by looking at the accuracy on the validation set. For training we use the parameters explained above and we vary the hyperparameters performing a grid search.

In the first round of tuning we investigate different optimisers: Adadelta, Adam, SGD and RMSprop. The best results are achieved with SGD. In the second round we investigate different number of LSTM units in the first layer as well as different learning rates. The resulting best model has 61 LSTM units and 0.08 as learning rate.

In Table 4 we report the results of this model on the validation set.

Metrics	Accuracy	F_1 weighted	Precision weighted	Recall weighted
Validation	0.5260	0.5110	0.5208	0.5260

Table 4: LSTM validation results, averaged between 20 trained models

[5] `https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams`

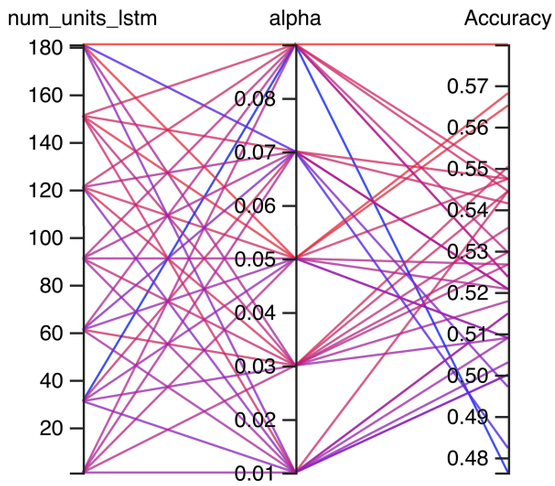


Figure 13: HParams dashboard in tensorboard

5.6 MLP

Multi layer perceptron (MLP) is a neural network architecture composed of dense fully connected layers. The MLP implemented for this task is composed by two layers: the first containing 181 neurons and the second containing 61. For both layers we use dropout in order to prevent the network from memorizing the training data and overfitting. We use a dropout rate of 0.4. The last layer is composed by two neurons outputting the probabilities for the two classes. The activation for the first two layers is "*tanh*" while the activation for the output layer is "*softmax*". The architecture of the network is shown in Fig. 14.

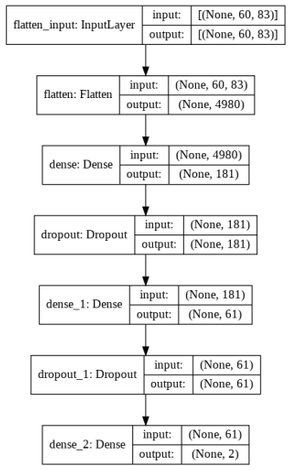


Figure 14: MLP architecture

We use sparse categorical cross-entropy as the loss function. During training we use early stopping on the validation loss with a patience of 3 epochs. The batch size for training is 32 while the maximum number of epochs is 20. The optimizer is SGD with a learning rate of 0.08, taken from the insights gained in section 5.5.1.

We perform hyperparameters tuning using the HParams plugin. The hyperparameters we investigate are:

- Number of units in the first hidden layer
- Number of units in the second hidden layer
- Dropout rate for the two layers

In Table 5 we report the results of the model on the validation set.

Metrics	Accuracy	F_1 weighted	Precision weighted	Recall weighted
Validation	0.5500	0.5478	0.5489	0.5500

Table 5: MLP validation results, averaged between 20 trained models

5.7 CNN + LSTM

Finally, we look into an hybrid architecture specifically engineered for this task. We want to benefit both from the feature extraction power of convolutional layers and from the time prediction power of LSTM cells. Our architecture, shown in Fig. 15, is composed by a first layer of 8 convolutional filters of size $1 * 83$ which summarise all the features of one day into one single feature. It is followed by a dropout layer with rate 0.2. After, we reshape our input with a lambda custom layer and we feed it to a LSTM layer with 8 units. The last layer is connected to a dense layer with two neurons representing the classes' probabilities. The LSTM cells provide their output after having processed the features of the 60 days. Between the last two layers we use a dropout of 0.1. The activation function is "*tanh*" for all the layers except the last which uses "*softmax*".

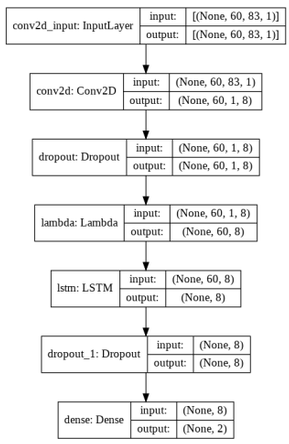


Figure 15: CNN+LSTM architecture

We use sparse categorical cross-entropy as the loss function. During training we use early stopping on the validation loss with a patience of 6 epochs. The batch size for training is 6 while the maximum number of epochs is 20. The optimizer is SGD with a learning rate of 0.08, taken from the insights gained in section 5.5.1.

5.7.1 Hyperparameter search with sklearn

The number of convolutional filters, the number of LSTM units and the batch size are obtained through hyperparameter tuning using sklearn. To use hyperparameter search in sklearn, we wrap our model using keras[6]. Then we specify the dictionary of the hyperparameter of interest and their domains.

When using grid search, sklearn automatically performs cross-validation. Thus, we use the splitting strategy described in section 5.2.1 with 2 time series splits. Differently from the other hyperparameters searches that we performed, with scikit-learn we can choose different metrics for evaluation during cross-validation. Therefore we use F_1 weighted as our evaluation

[6] `https://www.tensorflow.org/api_docs/python/tf/keras/wrappers/scikit_learn/KerasClassifier`

metric.

In Table 6 we report the results of the model on the validation set.

Metrics	Accuracy	F_1 weighted	Precision weighted	Recall weighted
Validation	0.5207	0.4821	0.5022	0.5207

Table 6: CNN+LSTM validation results, averaged between 20 trained models

5.8 Receiver operating characteristic

In figure 16 we can see the ROC curve of the various models computed on the validation set, with the area underlying each curve reported in Table 7.

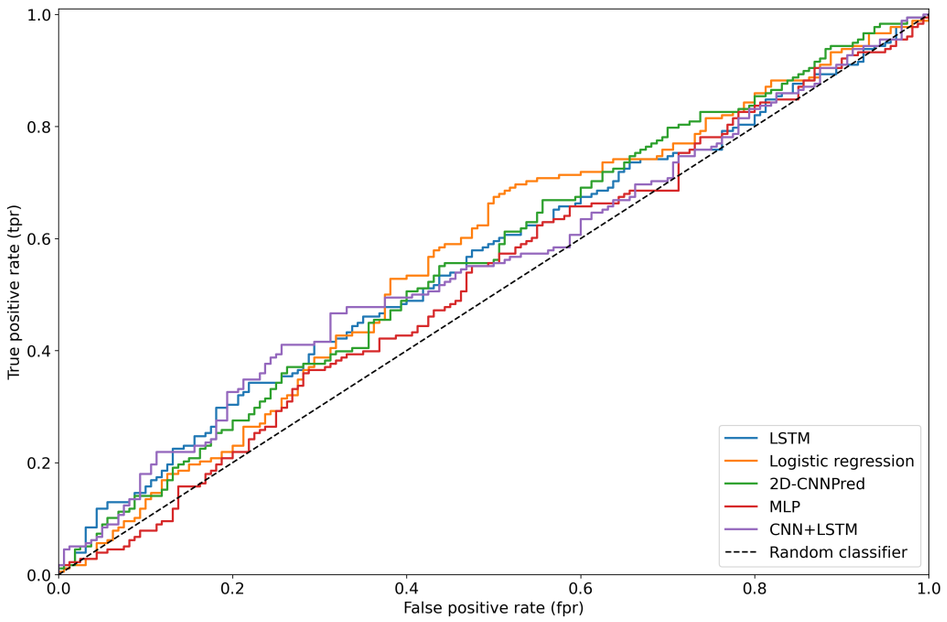
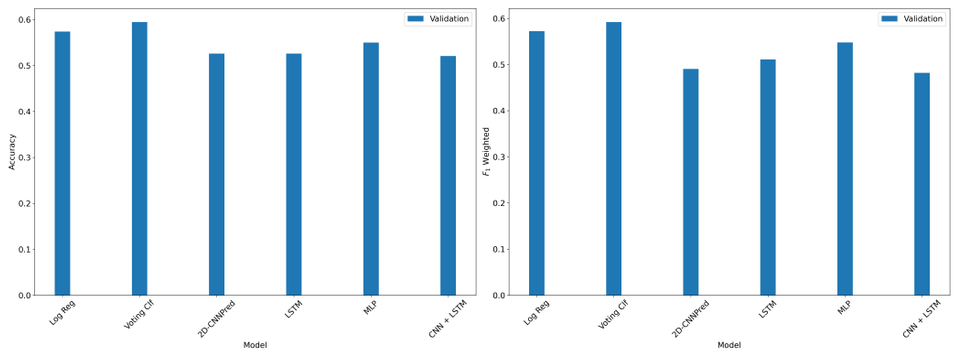


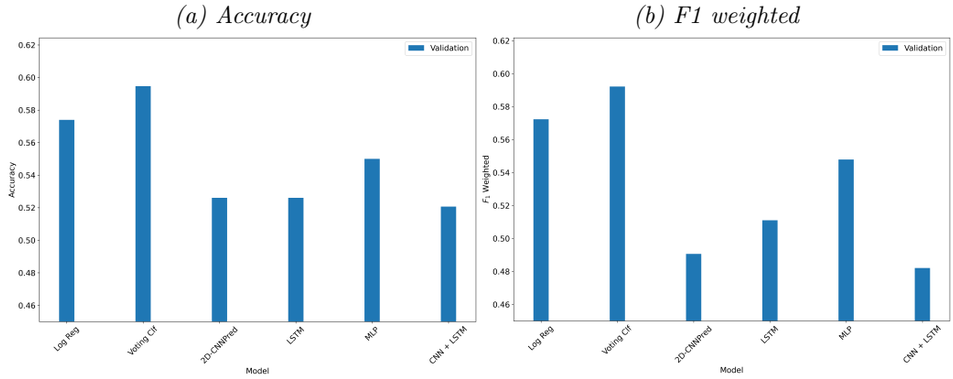
Figure 16: ROC curve computed on the validation set

Model	Area under ROC
Logistic regression	**0.5670**
Voting classifier	NA
2D-CNNPred	0.5635
LSTM	0.5596
MLP	0.5237
CNN+LSTM	0.5538

Table 7: Area under the ROC curve computed on the validation set. Note that this is not available for the voting classifier as it does not provide output probabilities



(a) Accuracy *(b) F1 weighted*



(c) Accuracy zoomed *(d) F1 weighted zoomed*

Figure 17: Evaluation metrics on validation set, averaged between 20 trained models

Validation	Accuracy	F_1 weighted	Area under ROC
Logistic regression	0.5740	0.5723	0.5670
Voting classifier	0.5947	0.5922	NA
2D-CNNPred	0.5260	0.4906	0.5635
LSTM	0.5260	0.5110	0.5596
MLP	0.5500	0.5478	0.5237
CNN+LSTM	0.5207	0.4821	0.5538

Table 8: Metrics recap on the validation set, averaged between 20 trained models

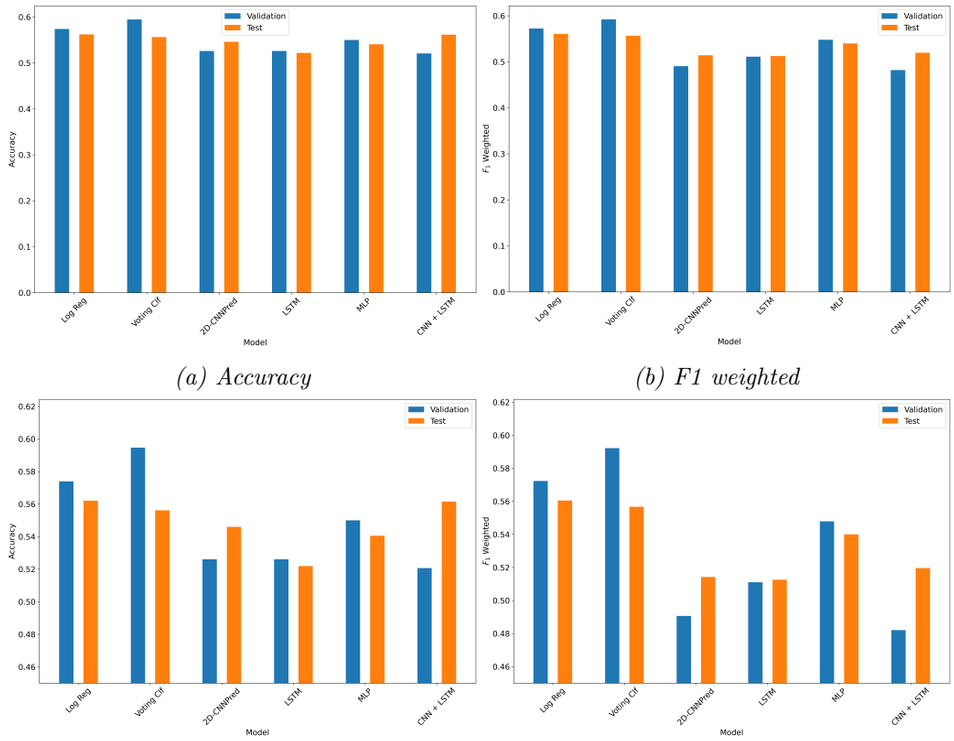
6 Conclusion

In Fig. 17 and in Table 8 we show the results of our models on the validation set. From the evaluation results we can see that the best performing models are the two traditional classifiers. The worst results are obtained by the convolutional and recurrent neural networks with our custom architecture obtaining the worst result. These results, however, are not totally truthful of the model quality. Being the validation set temporally close to the training set, models that may have overfitted the training set will still achieve a good result during validation. This may have happened for the MLP and the two traditional classifiers.

Looking for the first time at the results on the test set, shown in Fig. 18 and Table 9, we can confirm our hypothesis. Indeed, we observe a worsening

result for the two traditional classifiers, which are now failing to generalise on the test data. On the other hand, we see a strong increase in the performance of CNN-based models. Furthermore, our custom architecture outperforms 2D-CNNPred on both metrics, almost obtaining the best accuracy, showing that combining the power of CNN and LSTM for stock market prediction we can achieve a good solution.

We also test our models on the NYSE dataset. The test set for the NYSE dataset will be composed just by the last 20% of the data as the return of the stock markets is highly correlated and it would be biased to test on the first 80% given that we used very similar data for training and validating our models. For brevity, results on this part are available in the notebook.



(a) Accuracy *(b) F1 weighted*

(c) Accuracy zoomed *(d) F1 weighted zoomed*

Figure 18: Evaluation metrics on validation and test set, averaged between 20 trained models

Test	Accuracy	F_1 weighted
Logistic regression	0.5621	0.5604
Voting classifier	0.5562	0.5567
2D-CNNPred	0.5460	0.5142
LSTM	0.5219	0.5126
MLP	0.5405	0.5400
CNN+LSTM	0.5615	0.5196

Table 9: Metrics on the test set, averaged between 20 trained models

References

[1] Ehsan Hoseinzade and Saman Haratizadeh. Cnnpred: Cnn-based stock market prediction using several data sources. *arXiv preprint arXiv:1810.08923*, 2018. URL: `https://arxiv.org/pdf/1810.08923.pdf`.

[2] Andrew W Lo and A Craig MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *The review of financial studies*, 1(1):41–66, 1988.

[3] Yaser S Abu-Mostafa and Amir F Atiya. Introduction to financial forecasting. *Applied intelligence*, 6(3):205–213, 1996.

[4] Min Deng. Pattern and forecast of movement in stock price. *Available at SSRN 217048*, 2006.

[5] Eugene F Fama. Efficient capital markets: Ii. *The journal of finance*, 46(5):1575–1617, 1991.

[6] Bjoern Krollner, Bruce J Vanstone, and Gavin R Finnie. Financial time series forecasting with machine learning techniques: a survey. In *Esann*, 2010.

[7] Yakup Kara, Melek Acar Boyacioglu, and Ömer Kaan Baykan. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange. *Expert systems with Applications*, 38(5):5311–5319, 2011.

[8] Amin Hedayati Moghaddam, Moein Hedayati Moghaddam, and Morteza Esfandyari. Stock market index prediction using artificial neural network. *Journal of Economics, Finance and Administrative Science*, 21(41):89–93, 2016.

[9] Luca Di Persio and Oleksandr Honchar. Artificial neural networks architectures for stock price prediction: Comparisons and applications. *International journal of circuits, systems and signal processing*, 10(2016):403–413, 2016.