Short-term digital asset trend prediction with Deep Learning

William Harris - CQF Jan 2022 cohort
August 22, 2022

1 Introduction

Predicting market prices for any asset class is a notoriously difficult task. With asset markets being driven by an astonishing number of factors, prices end up exhibiting behaviour which at least means and therefore can be modelled as random and chaotic in nature. As a result, historical price data is noisy, non-linear and seemingly inadequate to predict future prices [1]. This however has not deterred investors and traders from developing quantitative techniques to help predict expected returns.

A common technique for evaluating a stocks potential future performance is fundamental analysis. This analysis disregards historical price data and attempts to characterise favourable stocks through ratio analysis and factor screening. On the other hand, technical analysis only considers historical price performance and attempts to determine price patterns to predict future trends. In addition, an emerging form of price estimation is on-chain analysis. This method uses public blockchain information such as mining difficulty, block size or active addresses to determine future price trends of digital assets. With the recent emergence of machine learning, there has been a growing amount of research into how these quantitative techniques can be improved using the many developing fields within machine learning science.

2 Methodology

2.1 Deep Learning

Deep learning is a subset of machine learning which attempts to replicate the basic functionality of a human brain. It consists of representational layers that are made up of individual neurons. The most popular neural network is a model which consists of at least three layers, an input layer, hidden layers and an output layer. With a deep learning neural network being defined by having sets of hidden layers. The neural network is trained by propagating data through each neuron in the neural network starting at the input layer and finishing at the output layer. Neurons in each layer are connected to other neurons in adjacent layers through weighted connections. These connections are adjusted in training through the optimisation algorithm gradient decent and are altered to minimise the error between the result returned from the output layer and the targeted result.

2.2 LSTM

In 1997 Hochreister and Schmidhuber designed a neural network neuron which extended the traditional recurrent neuron by solving the long-term dependency problem.

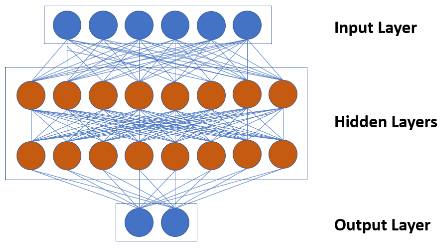

Figure 1: Neural Network

This added the ability to remember information in each neuron over long input sequences without reaching performance constraints. An LSTM neuron regulates the flow of information through the use of three gates, the input, forget and output gates. The input gate determines which current information is relevant to calculate the cells current state. While the forget gate accesses the prior state information and determines which prior state information is relevant. Both the input and forget gates contribute the necessary information to the output gate, which lastly regulates and controls the final state of the neuron. In essence, LSTM networks allow the processing of long sequential data which is useful with sequential datasets such as time series. [2]

2.3 Autoencoders

Autoencoders are unsupervised neural networks. They are thought to be a non-linear application of Principle Component Analysis (PCA). Autoencoders can be decomposed into two individual neural networks, the encoder and the decoder. Firstly, the encoder compresses the input data by having hidden layers which bottleneck the number of neurons. Secondly, the decoder attempts to reconstruct the compressed data back into the original input data. These algorithms can be valuable as the encoder is used separately from the autoencoder to transform the raw data into a lower-dimensional representations. This compressed data is often used in further supervised learning algorithms [3].

2.4 Self Organising Maps

Self organising maps (SOMs) are another type of unsupervised neural networks. Similar to autoencoders, they are used for dimensionality reduction. This makes the algorithm useful for clustering and visualise high-dimensional data. Their network's output layer is web of neurons that are evenly divided into a two-dimensional grid, with the output layer only activating for one of the output neurons in the grid. Unlike other types of neural networks, SOMs are trained through repetitively calculating the distance between weighted connections and the input. The neuron with the smallest distance is recorded as the activated neuron. This produces a system which maps high-dimensional data on to a two-dimensional plane [4].

3 Relevant Studies

One machine learning technique proven to yield high predictive ability in forecasting asset prices is deep neural networks. In 2015, Chen et al. presented a paper which predicted Chinese stock market returns using an LSTM deep learning neural network which successfully beat a random prediction method by 14.3% to 27.2%. This was achieved through a 10 feature and 60 time series look back period with an additional three-day earning rate labeling [5]. More recently, in 2022, Chhajer et al. studied the applications of three machine learning techniques for stock market prediction which included artificial neural networks (ANNs), support vector machines (SVMs) and LSTMs. From a comparison on an exhaustive list of studies, LSTM was described to be the best for stock market prediction [6].

More complicated neural network models have also been explored. One of which, Yan et al. investigated the use of an autoencoder network to first reduce the feature set before being used in an LSTM neural network. This produced a more robust feature set which had a higher associated relationship to the target and in addition helped combat the strong time correlation between data sequences [7]. Furthermore, Bao et al. used wavelete transforms, stacked autoencoders and LSTM networks together to predict stock prices. The proposed technique eliminated noise through the wavelete transformation, the autoencoders reduced the feature set before being used in an LSTM model which attempted to predict the next days closing price. Results showed that the new proposed model bet performance on a range of base line models [8].

Most notably, two studies which have returned exceptional results were developed through predicting future price trend classification (the next day move is either position or negative) rather than price regression (predicting the percentage change of the next day move). Firstly, Wu et al. used a combination of Convolution Neural Network (CNN) and LSTMs which were trained on historical and leading indicator data for ten stocks from the USA and Taiwan markets. Accuracy for these models was consistently evaluated at 90% across the selected stocks [9]. Secondly, Shen et al. performed feature engineering on the Chinese stock market dataset before training the LSTM model. This pre-processing step included recursive feature elimination, feature selection and principle component analysis dimension reduction. Outstandingly, results showed a 96% accuracy to predict up trends and 90% on down trends [10].

Looking outside of the traditional financial markets, deep learning neural networks have also been extensively researched in the digital assets market [11] [12] [13]. Two most relevant studies include one from Alonso-Monsalve et al. who discovered that Convolutional LSTM networks outperformed other types of neural networks for predicting one minute trends on six of the most popular digital assets. The networks were trained on 18 technical indicators on a short time frame of one minute [14]. The other is published work by Jagannath et al. who developed LSTM models through a comparison of three self-adaptive approaches to find the optimal hyper-parameters to predict the price of Ethereum accurately. This was conducted on LSTM models with on-chain metrics data and produced high predictability [15].

4 Research

This paper investigates the use of LSTM neural networks to predict one hour price trends of two popular digital assets Bitcoin and Ethereum. The feature dataset comprises of fifty-three technical indicators and six on-chain analytic metrics recorded in hourly time format. The optimal look back period used to generate the dataset sequences is evaluated and two types of feature extraction techniques are compared. The first technique uses manual feature selection through evaluating feature importance with Shaply Additive Explanations (SHAP) and SOMs to identify and minimise non-linear correlation between features. The second uses recurrent neural network autoencoders to compress the features in an attempt create a translated feature set with reduced noise in the data. This is conducted through the use of a developed object-orientated python pipeline which leverages KerasTuner to hyper parameterise models.

5 Implementation

5.1 Data Exploration

For both Bitcoin and Ethereum, all available hourly Open, High, Low, Close and Volume (OHLCV) price data was retrieved from the centralized exchange Bitfinex through the use of their public API. In addition, hourly Block Size, Hash Rate, Difficulty, Transaction Rate, Active Addresses and New Addresses on-chain data was retrieved from Glassnode through the use of a paid subscription API. These particular on-chain metrics were chosen due to their high Spearman rank correlation coefficient (equation 1) to price stated in N. Jagannath et al. paper *An On-Chain Analysis-Based Approach to Predict Ethereum Prices* [15]. This measure is an appropriate comparison for trend prediction as it explains the monotonic relationship between the price and the metric.

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{1}$$

The frequency of one hour was decided due to a limitation of data in the one day time frame. The inception date for Bitcoin and Ethereum is relatively new and as a result there is a scarcity of available pricing and on-chain data. It was decided that the one day time frame was not appropriate as the recommended number of 10000 sequential data points could not be supplied.

After collating the data, the python package MissingNo [16] helped to identify missing data entries. Figures 2 and 3 are the raw data MissingNo matrix plots for the Bitcoin and Ethereum respectively. The matrices show representations of each data column with values depicted as the colour black. Due to a small number of missing entries it was appropriate to apply a forward fill to replace any missing values. After which, the raw datasets were saved to a CSV file.

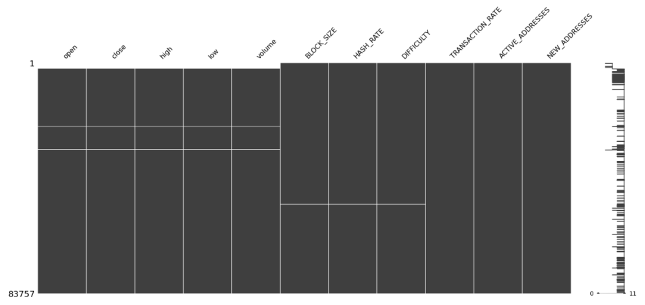

Figure 2: Bitcoin Missingno Matrix

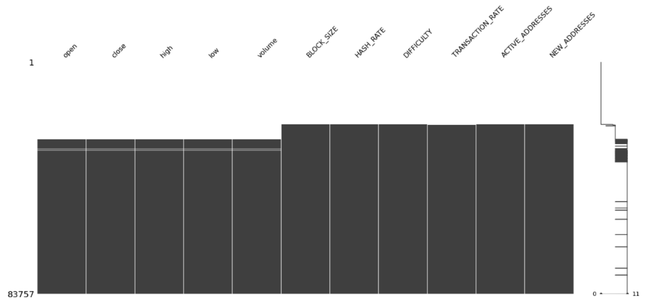

Figure 3: Ethereum Missingno Matrix

5.2 Pipeline

The proposed deep learning models were developed through the pipeline represented in figure 4. This framework was implemented in python as a object-programming package, which allowed for a fast and reliable way to develop neural networks and compare evaluation results. The pipeline consists of three sections, pre-processing, training and testing.

5.2.1 Pre-processing

The pre-preprocess step holds the functionality to manipulate the raw dataset and stage it into the three input datasets. This is encapsulated in the abstract class `BaseData` and additionally requires the raw dataset and target classification to be defined. After which, the data

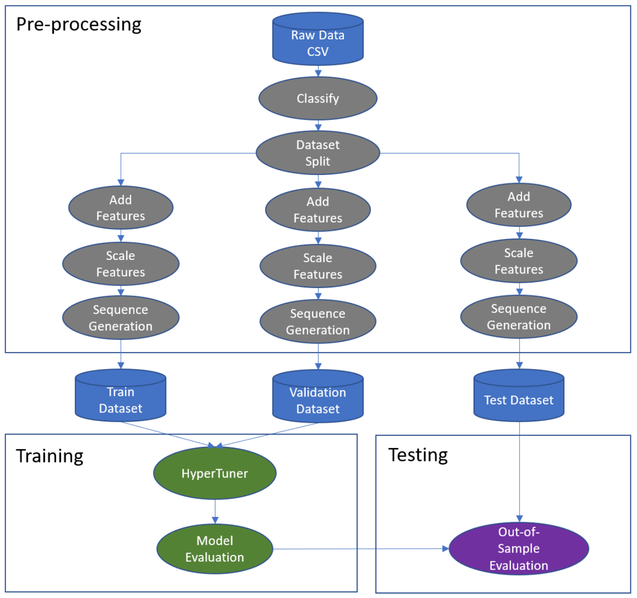

Figure 4: Machine Learning Pipeline

is split into three sets with consideration to maintaining the sequential time series order. Each dataset is further processed individually by calculating the defined features, scaling the features and generating the input data time series sequences. It is important to note that the features were applied to the raw dataset after separation to minimise possibilities of data leakage. This procedure has reduced the potential of data leakage for the following two reasons.

Firstly, input data is packaged into sequences of rolling time periods. Therefore, if rolling time period sequences are built first and then separated, then there will be occurrences where time periods are present in sequences which are in both datasets. For example figure 5 describes a simplified situation with a two day look back period. Using separation after sequence generation then the time period 01-03-2022 becomes present in both the test and validation datasets.

The second risk mitigated is similar to the first, however it stems from the technical indicators being calculated over varying time periods. If technical indicators are calculated before separating the data, information used to calculate an indicator in a subsequent dataset is created from information in the preceding dataset. If data leakage is not managed properly then model evaluation results can be overly optimistic and produce models which are not functional in production.

5.2.2 Training

Three datasets are output from the pre-processing pipeline, these include the train dataset, the validation dataset and the test dataset. These independent datasets are crucial for developing reliable machine learning models. Without such a split, models can give inaccurate and bias results. The train and validation datasets are used to

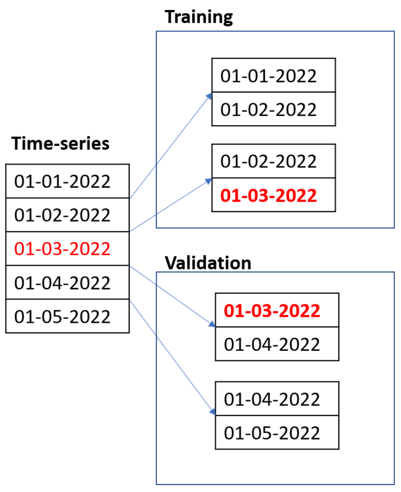

Figure 5: Data Leakage Sequence Packing

develop the model. The main goal of the train dataset is to teach the machine learning algorithm features and patterns to predict the target, while the validation dataset evaluates the model performance in training and helps to optimise hyperparameters.

The python packages TensorFlow [17] and KerasTuner [18] were used to develop the neural networks and search for model hyperparameters. A few abstract classes were developed to support the training of the models, these include: 1. An abstract class `BaseModel` defines model specific behaviour such as calculating model evaluation metrics and feature importance. 2. A child abstract method of `BaseModelFromScratch` extends the `BaseModel` class and allows for users to implement manual hyperparameter tuning. 3. The abstract class `BaseHyperTuner` allows users to define tuners to autonomously experiment with hyperparameter tuning.

5.2.3 Testing

The best hypertuned models (determined by the train and validation dataset metrics) are then evaluated on the test dataset. This dataset is important as it provides an unbiased final evaluation of a model. Since both training and validation datasets were used to develop the algorithm an out of sample dataset is required to confirm the model has successfully identified trends which are also present in a production setting. The test dataset is especially important for noisy data as the model might learn patterns which are specific to the train and validation datasets. Ideally a reliable model should produce evaluation metrics which are similar to all datasets. If the test datasets do not meet the same evaluation metrics as the others then the model has a degree of bias towards the train and validation datasets.

6 Experiment

6.1 Classifier

The target was defined by the binary classification that the specified future price is either below or above the current periods price, thus following the equation 2. This study uses a one hour future price to generate the target datasets. This classification resulted in a very slight class imbalance with more up trend classifications than down trends. To provide a balanced importance weighting between the classifications, class weights were calculated on the train dataset for each hyper tuning run and applied to the neural network in training.

$$y_t = \begin{cases} 0 & \text{if } p_t > p_{t+i} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

6.2 Dataset Split

The default dataset split percentage allocations are 60% to train, 20% to validation and 20% to testing. This allocation gives the approximate start and end date time series shown in tables 1 and 2. These dates are subject to change depending on the particular features selected.

	Start	**End**
Train	2013-04-02	2018-10-31
Validation	2018-11-02	2020-09-10
Test	2020-09-12	2022-07-22

Table 1: Bitcoin - Split Dates

	Start	**End**
Train	2016-03-11	2020-01-04
Validation	2020-01-05	2021-04-13
Test	2021-04-14	2022-07-22

Table 2: Ethereum - Split Dates

6.3 Features

The exhaustive set of technical indicators and on-chain metrics are shown in table 3. The python package FinTA [19] was used to generate the variation of volatility and momentum technical indicators.

6.4 Feature Scaling

Feature scaling is a data pre-processing step that can improve the neural networks ability to optimise connection weights. If features are not scaled, then convergence for gradient decent can be impacted making model performance worse. The scaling attributes are calculated on the train dataset only and subsequently the transform is applied to each dataset. One of two scalers are chosen for each feature with them being allocated based on features attributes. The scaling function StandardScaler from the

Type	Feature	Settings (Hours)
Technical Indicator	ROC	period 3, 6, 12, 24
	RSI	period 3, 7, 14, 32
	ATR	period 3, 7, 14, 32
	Williams %R (WILLIAMS)	period 3, 7, 14, 32
	Mass Index (MI)	period 3, 9, 18
	Commodity Channel Index (CCI)	period 5, 10, 20
	BASP	period 10, 20, 40
	Bollinger Bands (BB)	period 5, 10, 20
	Kaufman Efficiency Indicator (ER)	period 5, 10, 15, 20
	MACD	fast/slow/signal 6/12/4, 12/26/9, 19/39/9
	ADX	period 3, 7, 14, 21
	Stochastic Oscillator (STOCH)	period 3, 7, 14, 21
	StochRSI (STOCHRSI)	rsi/stoch 3/3, 7/7, 14/14, 21/21
	Bandwidth (BBWIDTH)	period 5, 10, 20
	Percent B (PERCENT_B)	period 5, 10, 20
On-Chain Metric	Block Size	
	Hash Rate	
	Difficulty	
	Transaction Rate	
	Active Addresses	
	New Addresses	

Table 3: Features

popular python package Scikit Learn [20] was applied to features which closely follow a normal distribution. The StandardScaler normalises each feature by removing the mean and scaling the variance to one, this is shown in equation 3. While, the MinMaxScaler function was applied to features with broadly defined upper and lower bounds. This scaler transforms the features into a zero to one range shown in equation 4.

$$StandardScaler = \frac{x - \mu}{\sigma} \tag{3}$$

$$MinMaxScaler = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4}$$

The scaling choice for each feature is outlined in table 4. In addition, since the models are predicting binary trend predictions, the ATR, Bollinger Bands and all on-chain metrics were altered to a percentage change base.

Feature	StandardScaler	MinMaxScaler
ROC		x
RSI		x
ATR	x	
Williams %R (WILLIAMS)		x
Mass Index (MI)		x
Commodity Channel Index (CCI)		x
BASP		x
Bollinger Bands (BB)	x	
Efficiency Indicator (EI)		x
MACD		x
ADX		x
Stochastic Oscillator (STOCH)		x
StochRSI (STOCHRSI)		x
Bandwidth (BBWIDTH)		x
Percent B (PERCENT_B)		x
Block Size	x	
Hash Rate	x	
Difficulty	x	
Transaction Rate	x	
Active Addresses	x	
New Addresses	x	

Table 4: Normalisation Types

6.5 Metrics

Three metrics which were used to evaluate model performance are binary accuracy, log loss and f1 score.

6.5.1 Binary Accuracy

Equation 5 defines the binary accuracy metric with TU, TD, FU and FD representing the number of true up trend, true down trend, false up trend and false down trend predictions, respectively. It is important to remember that the binary accuracy metrics can be missleading with imbalanced data. This is due to the fact that the metrics can return relatively high values if the majority classification is consistently predicted.

$$\text{Binary Accuracy} = \frac{TU + TD}{TU + TD + FU + FD} \tag{5}$$

6.5.2 Log Loss

Logistic loss is defined as the negative log-likelihood that a model returns accurate probabilities for classifications. Defined in equation 6, it provides a way to quantify the models confidence in predicting a certain classification. The base line log loss for a balanced binary classification is approximately 0.693 with values higher than this indicating worse predictive ability. In trend prediction, log loss is a pertinent measure to evaluate as prediction probabilities can be used in a tailored trading strategy.

$$\text{Log Loss} = y\log(p) + (1 - y)\log(1 - p) \tag{6}$$

6.5.3 F1 Score

The f1 score is an appropriate metric to use for imbalanced datasets as it takes into consideration both the number of correct and incorrect predictions. A higher f1 score indicates that a model has improved both its precision and recall on a harmonic mean basis. This measure also helps to evaluate a models ability to predict up and down trends with an equal importance. This is applicable for this study because trend prediction desires minimising both false up (FU) trend and false down (FD) trend classifications.

$$\text{Precision} = \frac{TU}{TU + FU} \tag{7}$$

$$\text{Recall} = \frac{TU}{TU + FD} \tag{8}$$

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{9}$$

6.6 Hypertuner

In the training pipeline, the Bayesian Optimisation algorithm was used to optimise hyperparameters. This was conducted through using the tuner class BaysianOptimization in the python package KerasTuner. The objective for the tuner was to maximise the validation binary accuracy metric with a maximum trial run set to fifty. The lower and upper bounds of the hyperparameter search space is listed in table 5. Other parameters which remained constant for all models include Batch size of 512

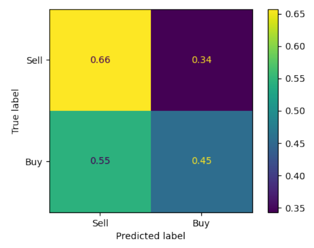

Figure 6: Bitcoin Best 10hr look back AFM - Validation

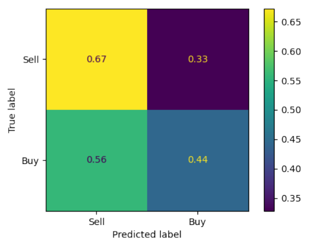

Figure 7: Bitcoin Best FSM - Validation

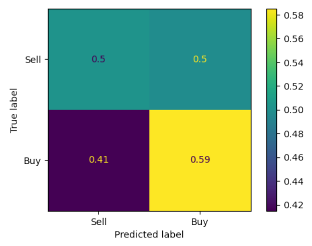

Figure 8: Bitcoin Best AEM - Validation

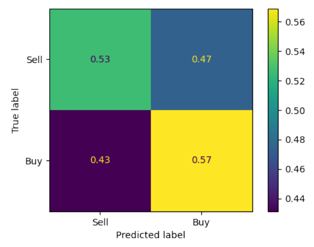

Figure 9: Ethereum Best 10hr look back AFM - Validation

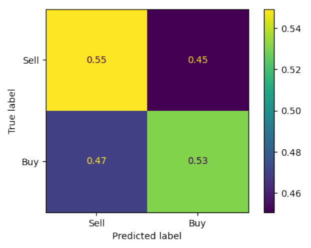

Figure 10: Ethereum Best FSM - Validation

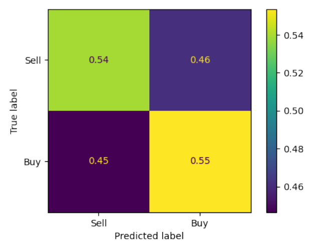

Figure 11: Ethereum Best AEM - Validation

and number of Epochs 100. In addition the hyperparameter Early Stopping was defined to monitor validation loss and stopped training if the loss did not improve within the last ten Epochs. The hyperparameter search space and static parameters were decided through manual hyperparameter testing. This was supported with TensorBoard to visualise results and determine appropriate variables though comparing model metrics. The best model from a hypertuner was evaluated by their performance relative to other models in the hypertuner. This was determined by both the final validation binary accuracy metric and verifying that the model was trained appropriately through visualised training and validation metric plots in TensorBoard.

Hyperparameter	**Min**	**Max**
LSTM input layer	32	320
# of extra hidden layers	1	6
LSTM hidden layer 1	32	320
LSTM hidden layer 2	32	320
LSTM hidden layer 3	32	320
LSTM hidden layer 4	32	320
LSTM hidden layer 5	32	320
LSTM hidden layer 6	32	320
LSTM last hidden layer	32	320
LSTM Dropouts	0	0.4
Dense Layer	32	128
Dropout Layer	0.1	0.4
Nadam Learning Rate	1e-4	1e-2

Table 5: Hyperparameter Search Space

7 Results and Analysis

Three model architectures were developed, the *all feature model* (AFM), *feature selection model* (FSM) and the *autoencoder model* (AEM). These models were created through the outlined pipeline with the main difference between models being the alterations to feature datasets.

7.1 All Feature Model (AFM)

The AFMs were developed to investigate the best look back period for the sequential data, diagnose feature importance and provide a comparable result to future model architectures. For both Bitcoin and Ethereum, three pipelines were run to evaluate the performance on a ten, thirty and sixty hour look back period. The train and validation evaluation metrics for the best performed model in each pipeline run are shown in table 6, 7.

Look back	**10 Hours**		**30 Hours**		**60 Hours**	
Dataset	**Train**	**Validation**	**Train**	**Validation**	**Train**	**Validation**
Binary Accuracy	0.561	0.554	0.568	0.556	0.571	0.556
Log Loss	0.682	0.686	0.702	0.724	0.677	0.688
F1 Score	0.517	0.51	0.534	0.518	0.54	0.525

Table 6: Bitcoin - Best Performing AFMs

Look back	**10 Hours**		**30 Hours**		**60 Hours**	
Dataset	**Train**	**Validation**	**Train**	**Validation**	**Train**	**Validation**
Binary Accuracy	0.558	0.548	0.556	0.548	0.542	0.544
Log Loss	0.685	0.69	0.685	0.69	0.69	0.69
F1 Score	0.582	0.566	0.586	0.568	0.578	0.567

Table 7: Ethereum - Best Performing AFMs

For the comparison between look back periods, the results show that a ten hour look back period is the most favourable parameter. The ten hour look back period is appropriate for both assets as they balance the trade

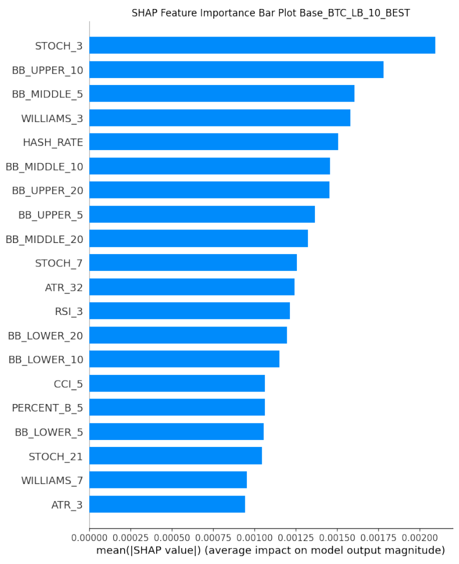

Figure 12: Bitcoin Feature Importance

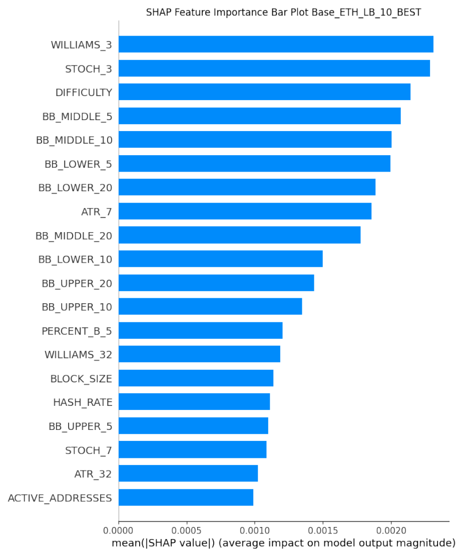

Figure 13: Ethereum Feature Importance

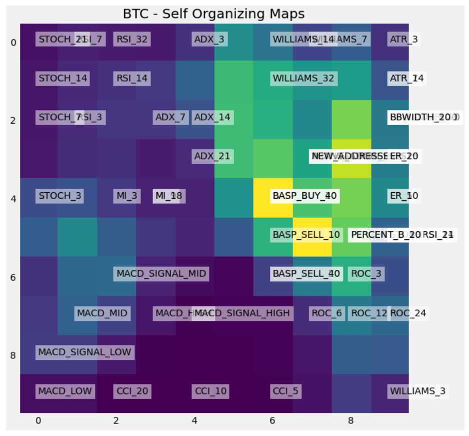

Figure 14: Bitcoin Feature SOM

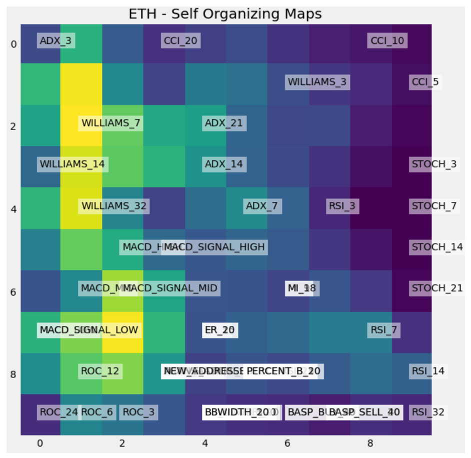

Figure 15: Ethereum Feature SOM

off between bias and variance appropriately. This conclusion was drawn from the following observations of results. Firstly, overfitting is more prevalent in the Bitcoin AFMs as the look back period increases. This increase in overfitting however is not outweighed with higher validation metrics. Secondly, the opposite occurs for the Ethereum AFMs, with underfitting occurring in the sixty hour AFM. This makes the ten hour AFM more attractive as the validation metrics are higher with a manageable amount of increased overfitting. As a result, the look back period of ten hours was implemented for the FSMs and AEMs and all comparisons between model architectures will be done with respect to the ten hour AFMs.

On aggregate the Ethereum models performed better than the Bitcoin models. This is supported through comparison of their f1 scores and confusion matrix's (figures 6 and 9 . The best Ethereum ten hour look back AFM was successful in predicting up and down trends more consistently than the best Bitcoin ten hour look back AFM. This is reflected in the f1 score being considerably higher and the confusion matrix showing 58% up trend predictions and 55% of down trend predictions. On the other hand, the Bitcoin confusion matrix shows a high 66% of down trend predictions however this is outweighed with only 45% of up trend predictions. The low up trend prediction impacts the evaluation metrics, binary accuracy and f1 score as the up trend predictions were lacking.

7.2 Feature Selection Model (FSM)

Feature selection can create more robust and effective machine learning algorithms. By introducing only features which have high predictive ability, the algorithm can be less prone to overfitting, produce higher overall out of sample results and reduce the time it takes to train the model. In addition, it can be beneficial to remove highly correlated features from the dataset. In most cases this can improve performance, however it does expose the model to losing information found in the interaction between correlated features.

Features for this model were selected based on what they contributed to the AFMs performance and their correlation to other features. SHAP provided insight into feature importance for each asset's best ten hour AFM. Figures 12 and 13 show the top twenty contributors to the neural networks predictive ability. Bollinger Bands across multiple calculation time period lengths were found to have high predictive ability. The on-chain metrics, Mining Difficulty, Block Size, Active Addresses were also found to contribute significantly to the Ethereum AFM. On the other hand, only one of Bitcoin's on-chain metrics Hash Rate was in the top twenty of contributors.

Feature	**Bitcoin**	**Ethereum**
Bollinger Bands - period 10	x	
Difficulty		x
Block Size		x
Hash Rate	x	
ATR - period 3	x	
Williams %R - period 3	x	x
Willaims %R - period 7	x	
Willaims %R - period 32		x
CCI - period 5	x	
RSI - period 3	x	
Percentage B - period 5		x
Stochastic Oscillator - period 3	x	x
Stochastic Oscillator - period 7	x	x
Stochastic Oscillator - period 21	x	

Table 8: Selected Features

These top twenty features were filtered again by analysing the results from the feature SOMs. The 10x10 SOMs shown in figures 14 and 15 provide graphical view of the non-linear relationship between features. The filtering was conducted by analysing each features grid location and only selecting the highest predictive contributor from each grid. Even though Bollinger Bands produced high predictive power, only the Bitcoin ten hour frequency was added to the feature set. This was due to them having high auto-correlation and in addition being correlated with some ATR indicators. As a result, all were characterised into the same grid. Interestingly, were the Stochastic Oscillator and Williams % R indicators as they recorded high contributions to predictive power but were also fairly separated in the SOM graphs. Thus the final dataset of selected features are listed in table 8.

Asset	**Bitcoin**		**Ethereum**	
Dataset	**Train**	**Validation**	**Train**	**Validation**
Binary Accuracy	0.558	0.555	0.543	0.54
Log Loss	0.691	0.692	0.693	0.695
F1 Score	0.509	0.504	0.554	0.545

Table 9: Best Performing FSMs

Evaluation metrics for the best performing models in the FSM hypertuners are displayed in table 9. The results show that both models fit the data appropriately as the train and validation metrics are broadly in-line. This makes an improvement in variance for the FSMs compared to the best ten look back AFMs. However, it also produces an increase in bias, as all AFM validation metrics, except the Bitcoin's binary accuracy, is worse than the FSMs validation metrics.

A similar analysis can be seen with respect to a comparison between the AFM and FSM confusion matrices. The best Bitcoin FSM's confusion matrix 7 has improved 1% on down trend predictions but this has also caused a 1% decrease in up trend predictions, resulting in the f1 score metric to be worse due to the slight class imbalance to up trends. The same conlcusion is drawn for the best Ethereum FSM's confusion matrix 10 with an increase in down trend predictions but a drastic decrease in up trend predictions.

The conclusions of this comparison need to be interpreted with care as the bias-variance trade off will become more noticeable in out of sample testing. This stems from the nature of asset price data consisting of high amounts of noise. As a result, models which more accurately find general trends in the data will perform better on out of sample datasets.

7.3 Autoencoder Model (AEM)

Two LSTM recurrent neural network autoencoders were developed to compress all features into a ten feature x ten hour look back data sequences. The architecture outlined in table 10 was used for the development of both the Bitcoin and Ethereum autoencoder. These networks were constructed through manual hyperparameter tuning. It is important to note that all features were scaled by MinMaxScaler. This is because overfitting was occurred on the normalisation technique defined in section Feature Scaling. In addition, the unsupervised machine learning technique t-Distributed Stochastic Neighbour Embedding (t-SNE) was leveraged to produce visualisation of the compressed data. As expected, small local clustering was observed, but due to the dataset having a high degree of noise, there was no clear separation between trends.

Model	**Layer (type)**	**Output Shape**	**Param #**
encoder	lstm (LSTM)	(None, 10, 126)	99792
encoder	lstm_1 (LSTM)	(None, 10, 10)	5480
decoder	lstm_2 (LSTM)	(None, 10, 64)	19200
decoder	time_distributed (TimeDistributed)	(None, 10, 71)	4615

Table 10: Autoencoder Architecture

The AEM hypertuners were run with respect to the compressed feature dataset. The best performing models evaluation metrics for each asset are outline in table 11. The log loss metric is relatively consistent between the train and validation comparison. However, the same comparison for the Bitcoin binary accuracy and f1 score indicates a slight overfit model. The best Ethereum AEM metrics shows that the model has underfit on binary accuracy but overfit on the f1 score. Significant improvement was made for both assets relative to the best FSMs. This is evident in comparison to the log loss and f1 scores metrics, of particular note the Bitcoin AEM validation f1 score

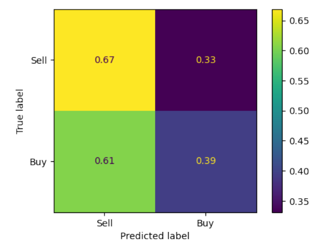

Figure 16: Bitcoin Best 10hr look back AFM - Test

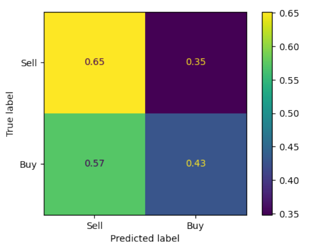

Figure 17: Bitcoin Best FSM - Test

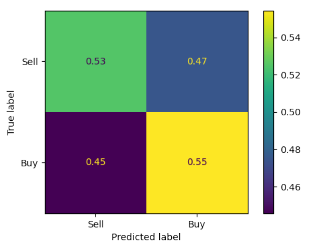

Figure 18: Bitcoin Best AEM - Test

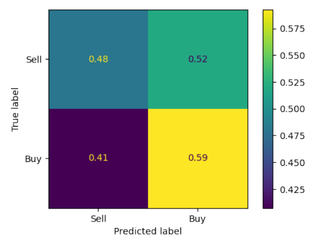

Figure 19: Ethereum Best 10hr look back AFM - Test

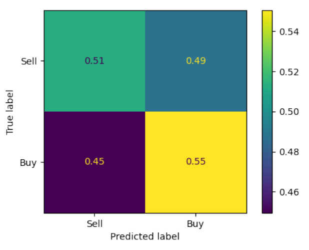

Figure 20: Ethereum Best FSM - Test

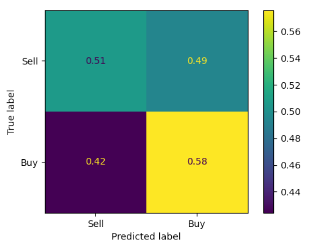

Figure 21: Ethereum Best AEM - Test

recording 56.7% while the Bitcoin FSM validation f1 score recorded 50.4%.

In regards to the ten hour look back AFM comparison, the Bitcoin AEM can be argued to be a better model. This is because the increase in variance seen from the higher training and lower validation metrics is a smaller impact then the improved lower bias seen in much larger f1 scores and comparable log loss metrics. This is also evident in the confusion matrix comparison, with figure 8 having the same predictive power in down trends but 59% in predicting up trends.

A much smaller improvement is seen in the Ethereum AEM compared against the Ethereum ten hour AEM. The binary accuracy metric for the Ethereum AEM is lower in the training dataset than the validation dataset. This indicates that the model is underfit and should be trained for longer. However, this is a better outcome than overfitting which is found in the Ethereum AFM binary accuracy. The f1 score comparison also favours the Ethereum AEM as the f1 score is 0.8% worse than the Ethereum AFM but is 1.9% better in terms of overfitting difference. The Ethereum AEM confusion matrix 11 shows this decrease with down trends improving by 1% however up trends decreased by 2%.

Asset	**Bitcoin**		**Ethereum**	
Dataset	**Train**	**Validation**	**Train**	**Validation**
Binary Accuracy	0.552	0.544	0.539	0.546
Log Loss	0.688	0.687	0.691	0.691
F1 Score	0.575	0.567	0.565	0.558

Table 11: Best Performing AEMs

7.4 Hypertuner Parameters

The tuned hyperparameters for each best performing model are displayed in table 12.

Asset	**Bitcoin**			**Ethereum**		
Model Type	**AFM**	**FSM**	**AEM**	**AFM**	**FSM**	**AEM**
LSTM input layer	32	32	167	32	320	279
# of extra hidden layers	1	2	1	1	1	1
LSTM hidden layer 1	32	32	167	32	320	279
LSTM hidden layer 2	NaN	32	NaN	NaN	NaN	NaN
LSTM last hidden layer	32	320	32	32	320	320
LSTM Dropouts	0	0.1	0.4	0	0	0.4
Dense Layer	32	128	72	128	128	67
Dropout Layer	0.4	0.2	0.4	0.4	0.1	0.1
Nadam Learning Rate	0.0001	0.01	0.0004	0.00535	0.00192	0.01

Table 12: Hypertuned Model Architectures

The following comments can be made about the results. Both AFMs constructed architectures with hyperparameters towards the lower bound of the search space for the number of neurons and extra hidden layers. This is as expected since the models would be highly exposed to overfitting due to the significant number of features. The tuner also found that the highest last dropout layer value helped to improve the overfitting. The FSM and AEM models show more refined tuned hyperparameters as these models were less exposed to overfitting due to more specific feature sets. The Ethereum FSM had the upper bound of the search space for the number of neurons in each layer. This shows that the model was trying to capture larger amounts of information through the small set of seven features. On the other hand, the AEMs fluctuated around the middle to upper bounds of neutrons in each layer. The AEMs were the only models to implement LSTM dropouts with the maximum values of 40%. Therefore, there was an advantage to implementing more neutrons per layer and offsetting the additional overfitting with a high dropout applied to each LSTM layer.

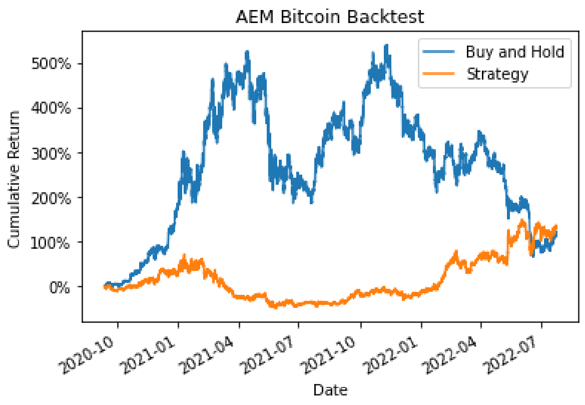

Figure 22: Bitcoin Backtest

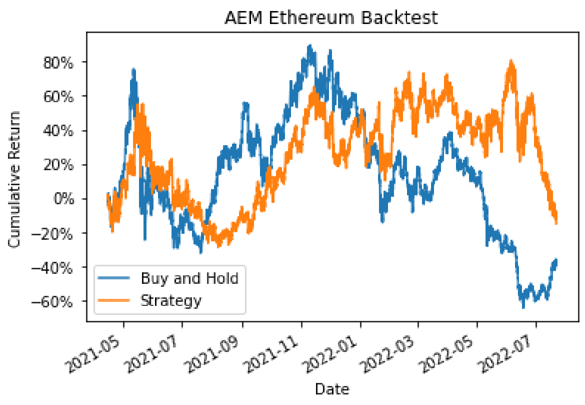

Figure 23: Ethereum Backtest

7.5 Out Of Sample Evaluation

Out of sample testing is a crucial part in evaluating deep learning models as it provides a clearing view on the bias-variance trade off between models. The test dataset evaluation metrics are displayed in table 13.

Asset	**Bitcoin**			**Ethereum**		
Model Type	**AFM**	**FSM**	**AEM**	**AFM**	**FSM**	**AEM**
Binary Accuracy	0.529	0.538	0.54	0.537	0.531	0.541
Log Loss	0.693	0.703	0.69	0.697	0.701	0.691
F1 Score	0.459	0.485	0.550	0.564	0.542	0.558

Table 13: Out Of Sample Testing

For the Bitcoin models, the results clearly shows that the AEM model performs the best with all three metrics returning substantially better numbers than the other two models. This confirms the comparison seen in the training pipeline with an increase in f1 score being more important that managing the small increase in variance. In addition, the FSM model returned more favourable results than the AFM as the decrease in variance for the FSM provided a larger benefit than the decreases in bias. This can also be seen in analysis of the confusion matrices, 16, 17 and 18. The AEM confusion matrix shows an advantage over both the AFM and FSM as it predicted down and up trends with a fairly equal weighting. This demonstrates that the model was able to successfully determine up and down trends as it was not predicting one class consistently.

Less obvious than the Bitcoin models, the Ethereum AEM model also performs best. The binary accuracy is the highest with log loss also being the lowest. The f1 score shows better performance then the FSM, while also lower than AFM. The reason for this is seen in the confusion matrices (figures 19, 20 and 21) with up trend predictions for the AEM decreasing by 1% but having a 2% increase in down trend predictions. This produces the AEM recall to decrease while the precision increases relative to AFM. Thus the f1 is lower with increased false up trends having a bigger impact than the increase in true down trends. However, it can be argued that a model which is able to predict both up and down trends with an equal weighting is more valuable than a single sided prediction model.

Both AEM models show adequate comparable results between datasets, while the same can not be said for the other two models. This adds additional evidence towards that the AEM models have successfully found general trends. On the other hand, the Ethereum FSM performed the worst. Showing that manual feature selection can be detrimental to model performance. In this situation too many features were removed from the feature dataset as many of the top contributors to prediction ability were correlated. The removal of correlated features could also negatively impacted the model since information gained through interactions between correlated features is removed. For example, when the correlation between two features is momentarily stopped it may provide indication of future behaviour.

Lastly, a crude backtest was conducted on the test datasets for both the best Bitcoin and Ethereum AEMs (figure 22 and 23). A buy and hold benchmark was used to evaluate the models performance. Additionally, this backtest does not incorporate any transaction fees or buy-sell spreads, thus the results should be considered lightly as these could affect the returns significantly. The figures show that both strategies outperformed the benchmark. The Bitcoin AEM struggled to maintain performance at inception with a large drawndown impacting most of the first and second quarters of 2021. The model regained performance after the third quarter of 2022 to outperform the buy and hold benchmark by approximately 10%. The Ethereum AEM was highly correlated with the benchmark from the inception to the first quarter of 2022. After which the model outperformed the buy and hold strategy by returning 20% higher at the end date.

8 Conclusion

This study has investigated the application of LSTM neural networks to predict short term Bitcoin and Ethereum price trends. A range of technical indicators and on-chain metrics were explored and utilized as the dataset and three models were explored to determine appropriate network architectures and hyperparameters. All network architectures produced prediction significant models with an recurrent netural network autoencoder architecture returning the highest results. The study demonstrates the benefit of autoencoders to compress technical indicator and on-chain metrics over manual feature selection. It also demonstrated the risks of manual feature selection through the removal of too many features. Further research could be conducted to 1. explore and identify the best autoencoder architectures which optimises the LSTM networks trend prediction ability and 2. refine the feature selection processes through investigating other feature scaling and filtering processes.

References

[1] Ahangar R.G., Yahyazadehfar M., Pournaghshband H. (2010). The Comparison of Methods Artificial Neural Network with Linear Regression Using Specific Variable for Prediction Stock Price in Tehran Stock Exchange. International Journal of Computer Science and Information Security Vol. 7, No. 2

[2] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[3] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. Science, 313(5786), 504–507. https://doi.org/10.1126/science.1127647

[4] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43(1), 59–69. https://doi.org/10.1007/bf00337288

[5] Chen, K., Zhou, Y., & Dai, F. (2015). A LSTM-based method for stock returns prediction: A case study of China stock market. 2015 IEEE International Conference on Big Data (Big Data). https://doi.org/10.1109/bigdata.2015.7364089

[6] Chhajer, P., Shah, M., & Kshirsagar, A. (2022). The applications of artificial neural networks, support vector machines, and long–short term memory for stock market prediction. Decision Analytics Journal, 2, 100015. https://doi.org/10.1016/j.dajour.2021.100015

[7] Yan, Y., & Yang, D. (2021). A Stock Trend Forecast Algorithm Based on Deep Neural Networks. Scientific Programming, 2021, 1–7. https://doi.org/10.1155/2021/7510641

[8] Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PLOS ONE, 12(7), e0180944. https://doi.org/10.1371/journal.pone.0180944

[9] Wu, J. M.-T., Li, Z., Herencsar, N., Vo, B., & Lin, J. C.-W. (2021). A graph-based CNN-LSTM stock price prediction algorithm with leading indicators. Multimedia Systems. https://doi.org/10.1007/s00530-021-00758-w

[10] Shen, J., & Shafiq, M. O. (2020). Short-term stock market price trend prediction using a comprehensive deep learning system. Journal of Big Data, 7(1). https://doi.org/10.1186/s40537-020-00333-6

[11] Cavalli, S., & Amoretti, M. (2021). CNN-based multivariate data analysis for bitcoin trend prediction. Applied Soft Computing, 101, 107065. https://doi.org/10.1016/j.asoc.2020.107065

[12] Awotunde, J. B., Ogundokun, R. O., Jimoh, R. G., Misra, S., & Aro, T. O. (2021). Machine Learning Algorithm for Cryptocurrencies Price Prediction. Studies in Computational Intelligence, 421–447. https://doi.org/10.1007/978-3-030-72236-4_17

[13] Aggarwal, A., Gupta, I., Garg, N., & Goel, A. (2019). Deep Learning Approach to Determine the Impact of Socio Economic Factors on Bitcoin Price Prediction. 2019 Twelfth International Conference on Contemporary Computing (IC3). https://doi.org/10.1109/ic3.2019.8844928

[14] Alonso-Monsalve, S., Suárez-Cetrulo, A. L., Cervantes, A., & Quintana, D. (2020). Convolution on neural networks for high-frequency trend prediction of cryptocurrency exchange rates using technical indicators. Expert Systems with Applications, 149, 113250. https://doi.org/10.1016/j.eswa.2020.113250

[15] Jagannath, N. et al. (2021) 'An On-Chain Analysis-Based Approach to Predict Ethereum Prices', IEEE Access, 9, pp. 167972–167989. doi:10.1109/access.2021.3135620.

[16] Missing data visualisation (2022) MissingNo, Software avaiable from `https://pypi.org/project/missingno/`

[17] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, & Others (2015) TensorFlow: Large-scale machine learning on heterogeneous systems, Software available from `https://tensorflow.org`

[18] O'Malley, Tom and Bursztein, Elie and Long, James and Chollet, François and Jin, Haifeng and Invernizzi, Luca and others (2019) KerasTuner, Software available from `https://github.com/keras-team/keras-tuner`

[19] Financial Technical Analysis (2022) FinTA, Software available from `https://pypi.org/project/finta/`

[20] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and others (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research Vol 12 Pg 2825–2830. Software available at `https://scikit-learn.org/`