CQF Exam Three Choice B

January 2022 Cohort

William Harris

CERTIFICATE IN QUANTITATIVE FINANCE
CQF

June 21, 2022

Problem 1

First, theory tasks:

- Mathematically specify three types of Loss Function for decision trees.
- Which data transformation can prepare the data for linear separation? (answer to score marks)

Solution. The loss function for classification decision tree's can be thought as the cost incurred for miss-classifying an observation. The algorithm attempts to find a criterion splitting branch structure which produces the lowest loss function and thus maximises the information gain for all observations over each branch node. Three types of classification loss functions for decision trees are:

- Entropy: $-\sum_{j=1}^{J} \log(p_j) p_j$
- Gini impurity index: $1 - \sum_{j=1}^{J} p_j^2$
- Miss classification error: $1 - \max_J p_j$

Here, J is the set of all classes with $j \in J$. p_j is the probability likelihood that the observation is class j at the particular branch node.

Feature scaling is an important data pre-processing step for machine learning pipelines which calculate distance measurements across features. Some examples of these include PCA or LDA feature reduction techniques or machine learning models which use linear separation to classify data. Support Vector Machines is one of these models, whereby the objective function attempt's to maximise the distance between observations and a separating hyperplane. If the dimensions are not scaled, then the distances between the observations and the hyperplane can be dominated by features which have variances that are higher than others. By scaling each feature to have a mean of zero and variance of one then the dimensional space of observations have a distance metric which is standardized.

Problem 2

Produce two models: (1) initial mapping model with many features, and (2) reduced in features model that aims to improve prediction of negative moves. Implementation tasks for models type (1), (2) as appropriate.

- Vary hyperparameters (min number to split, minimum number in leaf, and maximum depth) – can use ready search RandomizedSearchCV, GridSearchCV. While you are not required to generate surfaces, please.
- Formulate up to four principles/purposes of trees pruning. Illustrate with model type (2) vs (1) as appropriate.
- Provide attribution to splits (discuss if they are sensible), particularly for the model type (2) better negative moves classification.
- Provide plots for decision boundaries (surfaces), and name two specific issues for prediction quality of tree models. Investigate the prediction quality using area under ROC curve (each class) and confusion matrix. Do report on results for train/test split (typically, 50%/50% of observations).

Solution. The following steps were conducted to generate the two classification decision tree models. These models will attempt to predict the next days price trend for Bitcoin prices from the exchange Bitfinex.

Data Collection

Firstly, Bitcoin daily OHLCV data was retrieved from the Bitfinex API through the use of the python package `bitfinex-api-py` and saved to a CSV file. The series length included daily prices starting from the 2013-03-31 until 2022-05-27.

Explore Data

Bitcoin is traded 24/7 on Bitfinex, therefore we should expect prices for every date. To investigate this, a new dataframe was created which has the index set to the full date range of the pricing data. The prices downloaded from Bitfinex were joined onto the new full date range dataframe. The python package `missingno` was used to generate the Figure 1 below which identifies missing values in the dataframe.

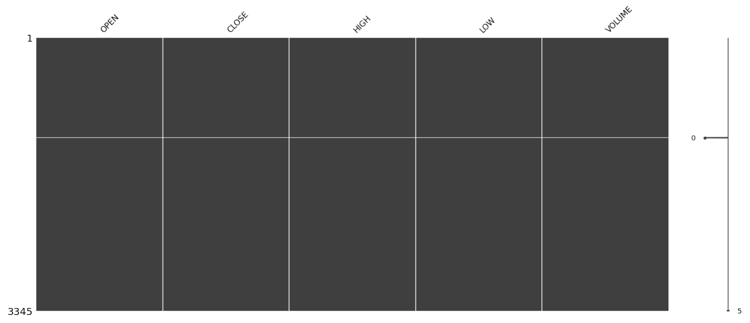

Figure 1: OHLCV Missing Data Matrix

There was a small sample of dates which had missing values ranging from 2016-08-03 to 2016-08-09. This can be is seen through the horizontal white line separating each column into two sections. The forward fill pandas function `fillna(method='ffill')` was applied to the dataframe to replace these missing values and keep the price series time consistent.

Features, Target Labelling and Transform Data

The following indicators were selected as features.

Feature	Formula	Time Span
O_C, H_L	Open-Close, High-Low	current time
MOMENTUM	$p_t - p_{t-k}$	$\forall k \in A, \forall k \in B$
MA	$\frac{1}{n}\sum_{i=0}^{n-1} p_{t-i}$	$\forall i \in B$
EMA	$EMA_{t-1} + \frac{2}{i+1}[p_t - EMA_{t-1}]$	$\forall i \in B$

where $A = \{i \in \mathbb{W} | 1 \leq i \leq 4\}$, $B = \{i \in 5\mathbb{W} | 5 \leq i \leq 55\}$

The target variable that the decision trees will attempt to predict is the binary classification that tomorrows percentage close return will be positive or negative. The values of +1 for a positive return and 0 for negative return was assigned, thus following the equation:

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

Here p_t is the current day close price and p_{t+1} is the next day close price.

This target labelling produced a mild imbalanced data set with 1738 upward trends and 1552 downward trends (after removing any rows with missing calculated features). This could cause issues with model training as the model might return a relatively high accuracy through solely predicting the majority class (upward trend). The under sampling technique to balance the training set was investigated, however better results were found by implementing model class weightings appropriately. Additionally, for the purpose of accommodating future problems in this exam, under sampling should not be used for decision trees where the internal probabilities are considered, as under sampling will skew the model's prediction class probabilities. With a training to testing separation of 20%, the following target labelling counters were:

Trend	Training	Testing
Up	1393	345
Down	1239	313

Table 1: Data Set Classification Breakdown

Model (1) Many Features

Model (1) was trained on all 37 features with the hyper-parameters for the model `DecisionTreeClassifier` being left as the preset values. The results below show that overfitting has occurred severely in this model with a training accuracy of 100% and test accuracy of 48.33%.

Data Set	Accuracy	Log Loss
Training	1	0.0
Testing	0.4833	17.8469

Table 2: Model (1) Results

Model (1)'s Confluence Matrix shown in Figure 2 was generated on the testing data set and gives a breakdown of prediction results per target label. It shows that Model (1) majority attempted to predict a downward move with 209 true negative predictions and 236 false negative predictions.

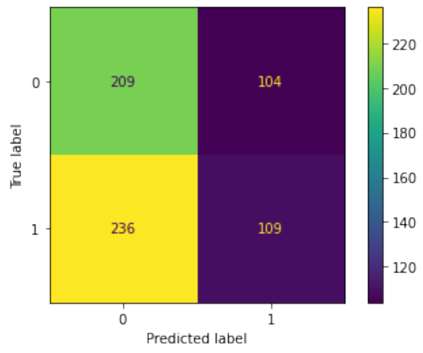

Figure 2: Model (1) Confusion Matrix

Model (1)'s tree plot shown in Figure 3 is a simplified version of the entire tree structure as it only shows the top four leaves. The preset hyper-parameter were not changed to reflect the data set and as a result the decision tree was constructed in a way which repetitively split leaves until each training data set was purely classified. Therefore, this has caused the model to overfit the training data as it has not learned any generic trends in the data but instead memorized each training set observation.

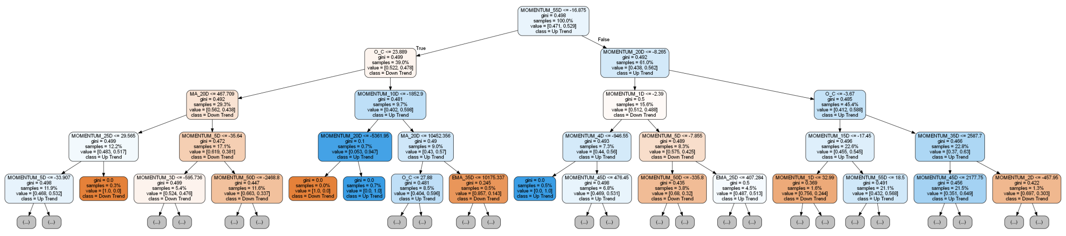

Figure 3: Model (1) Tree Plot

Model (2) Reduced Features and Improve Down Moves Classifications

Feature selection is an key part in machine learning as it helps to determine which features will be most crucial in creating a robust and effective algorithm. By only introducing favourable features to the algorithm it can decrease overfitting, increase its predictive ability and reduce training time. The following section gives an overview of the techniques used to select an improved feature set.

Feature Importance

Feature importance provides an insight into the average amount each feature has reduced the impurity of the tree leaves. The feature importance for Model (1) was investigated with a bar chart for the top 10 features shown below.

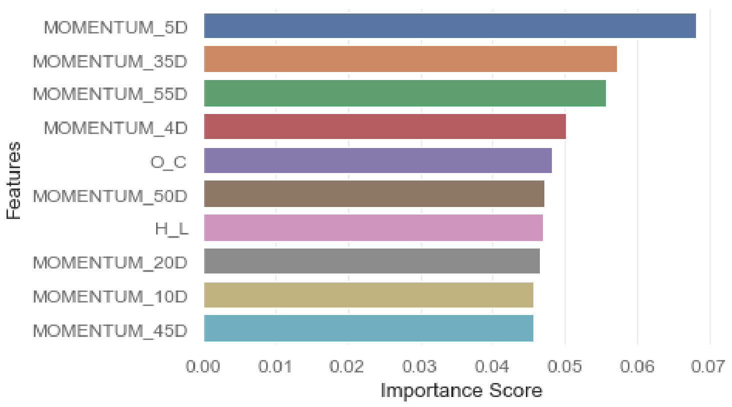

Figure 4: Model (1) Feature Importance

Scikit-Learn Feature Selection Modules

The following three feature selection techniques from Scikit-learn were run and compared in Table 3. The table is sorted by most important features to least important, with only the top 10 shown in the table.

- Variance Threshold: A variance threshold of 0.1 was used to identify which features have the highest variance. As expected, the Open-Close, High-Low and smaller time-frame features were ranked the highest.

- Mutual Information Classification: This technique helps to identify features which have a high dependency to the target data set. Interestingly, higher time-framed momentum features had the highest.
- RFE: Recursive feature elimination identifies the most favourable features by repeatable training and evaluating the most important features through recursively decreasing the feature set. Open-Close returned the highest importance ranking with small to mid ranged time-frames for the momentum indicator also being important.

Variance Threshold Ranking	Mutual Information Ranking	RFE Ranking
O_C	MOMENTUM_35D	O_C
H_L	MOMENTUM_40D	MOMENTUM_2D
MOMENTUM_1D	MOMENTUM_25D	MOMENTUM_4D
MOMENTUM_2D	MOMENTUM_50D	MOMENTUM_10D
MOMENTUM_3D	MOMENTUM_55D	MOMENTUM_15D
MOMENTUM_4D	MOMENTUM_1D	MOMENTUM_20D
MOMENTUM_5D	MA_15D	MOMENTUM_30D
MA_5D	MOMENTUM_45D	MOMENTUM_35D
EMA_5D	H_L	MOMENTUM_45D
MOMENTUM_10D	MA_55D	EMA_55D

Table 3: Scikit Learn Feature Selection Rankings

SHAP

Lastly, the python package `SHAP` was used to generate a `beeswarm` plot (Figure 5) which was run on a decision tree with preset parameters. Similar to the feature importance bar chart, the beeswarm plot provides a breakdown of each features contribution to the model. However, it also shows how the impact on the model changes depending on the particular feature's value. The highest ranking features include the Open-Close, High-Low and a range of varying time-framed momentum indicators.

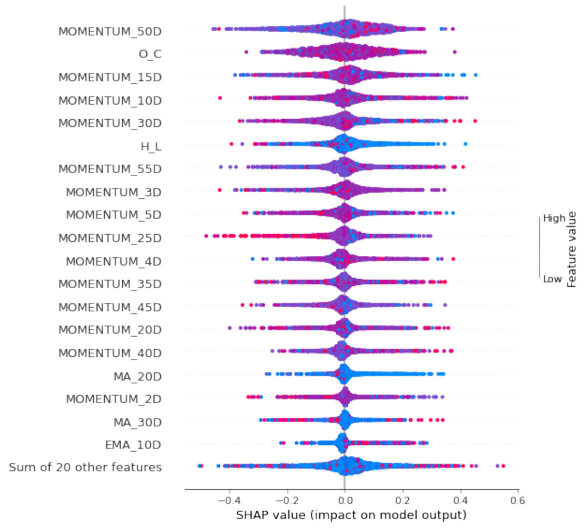

Figure 5: Decision Tree SHAP Beeswam

Feature Selection

Table 4 specifies the features which were selected through referencing the discussed feature ranking technique results. It is important to note that MA and EMA indicators were not selected as they were not found to have high rankings in any feature selection technique. This is due to the fact that these features are not scaled, with their values being subject to the current price. To make them more relevant, the indicator could be divided by the periods current price. This would produce a new indicator that is relatively comparable irrespective of the current observations price value.

Features
O_C
MOMENTUM_1D
MOMENTUM_2D
MOMENTUM_15D
MOMENTUM_25D
MOMENTUM_40D
MOMENTUM_45D
MOMENTUM_50D
MOMENTUM_55D

Table 4: Model (2) Features

Training and Testing

A grid search cross-validation on a five time series split was performed to find the best hyper-parameters for Model (2). `balanced_accuracy` was chosen as the cross-validations scoring metric as the best model should be selected on the basis of correctly predicting upward and downward moves with equal importance. This will also help to select the best model with respect to using the unbalanced data set. The search space parameters were selected to reflect the data set specified in Table 1.

Hyper-parameter	Search Space	Best Parameters
max_depth	{3,4,5}	3
min_samples_leaf	{25, 50, 100}	25
min_samples_split	{150, 200}	150

Table 5: Cross-Validation Parameters and Results

As seen in Table 6, Model (2)'s overfitting has improved compared to Model (1), with the difference between training and testing accuracy decreasing to 3.949%. This has been driven by the training accuracy decreasing and testing accuracy increasing relative to Model (1)'s performance. Figure 6 shows that the downward predictions have increased slightly compared to Model (1), however more importantly the upward predictions have significantly increased with 149 correctly upward predictions compared against Model (1)'s 109.

Data Set	Accuracy	Log Loss
Training	0.5866	0.6737
Testing	0.5471	0.7033

Table 6: Model (2) Results

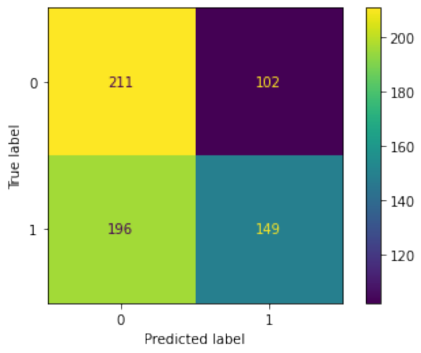

Figure 6: Model (2) Confusion Matrix

Pruning

Pruning is the process whereby trees which are being built are stopped before their leaves are either purely separated or are unable to be separated any further. This is done by specifying appropriate hyper-parameters which limit the tree from building entire trees. Implementing pruning effectively can add multiple benefits which include:

- Reduce overfitting: By restricting the number of leaves the model is constructed in a way that reflects general patterns in the data set.
- Increase performance metrics: Out-of-sample testing data will perform better on trees which are not tailored to memorizing the training data set.
- Decrease complexity: Trees that have been pruned are easier to understand as they do not have long complicated leaf structures.
- Decrease training time: It takes less time to train the model since the entire tree is not being constructed.

The cross-validation grid search for Model (2) has helped to identify the most appropriate hyper-parameters. These new hyper-parameters have implemented pruning which has forced the decision tree to learn more generic trends in the data, rather than creating a long and complicated tree structure which memorized the training data set. This affect can be seen by comparing the different tree plots between Model (1) (Figure 3) and Model (2) (Figure 7). For Model (1), by the third leaf from the top there are four splits which are purely separated leaves and have approximately 0.3% of the total samples in each leaf. Since there are very small samples in each leaf, these cases are most likely specific to the training data set and they will not occur in the testing data set. This becomes the root cause for Model

(1) to overfit the training data set. In comparison, Model (2)'s tree plot does not have these small pure leaves and instead has more evenly distributed samples in each leaf node.

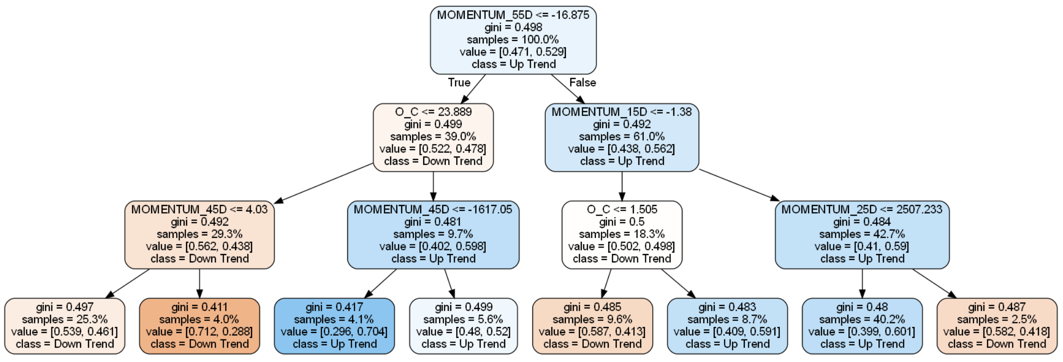

Figure 7: Model (2) Tree Plot

ROC Curves

The ROC Curves for both Model (1) and Model (2) are shown below. In summary, Model (1) ROC curve shows that the model does not have any predictive ability better than random selection, with it infact being slightly worst than 50/50 chance. Model (2) on the other hand, has a predictive ability slightly better than random selection, with the AUC curve equaling 0.55.

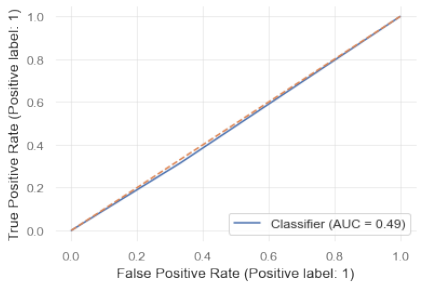

Figure 8: Model (1) ROC Curve

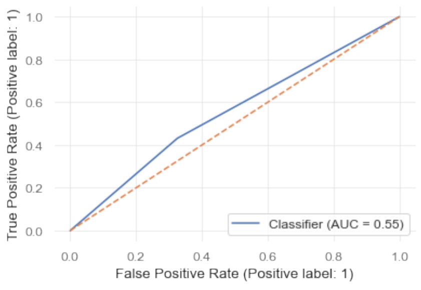

Figure 9: Model (2) ROC Curve

Attribution Splits

As seen in Model (2)'s Tree Plot (Figure 7). Two interesting attribution splits for Model (2) are:

- Third row, last leaf: MOMENTUM 25D less than or equal to 2507.233. This split is attempting to classify price action which is in a strong upward mid term trend. If true, this split produces a 60% probability that the next day is an up trend. While if false produces a 58% probability that the next day is a down trend. This describes that if the mid term momentum trend is overly extended then there is a high likelihood that there will be pullback in price. The leaves that precede this one include, MOMENTUM 15D is larger than -1.38 and MOMENTUM 55D is larger than -16.875. These classifications are describing price action which is neutral to an upward momentum on both the long term and short term time horizons. Therefore, if the mid term momentum is not classified as being very strongly upward trending, then there is a likelihood that the trend will continue as upward.

- Second row, first leaf: Open-Close less than or equal to 23.889. This leaf is one below from the top of the tree with it being preceded by the MOMENTUM 55D indicator which has to be smaller than -16.897. These two splits describe price action which is on a downward on the long term horizon. While on the short term horizon, if the Open to Close price increases by more than 23.889 then there is a 60% chance that the next move will be upwards. However, if the Open to Close price is less than 23.889 then there is a 60% that the price decrease. Intuitively, these two splits makes sense as a long term downward move with a strong upward move in the short term, insinuate that there will be a relief rally. On the other hand, if the short term horizon is still negative, then the down trend will most likely continue downward.

Decision Boundaries

Model (2) decision boundary plots were investigated through the use of the python package `mlxtend`. Below are two decision boundary plots that were found to be the most interesting as they help to describe the relationships found in the attribution splits explored above. It is important to note however that these plots also show how difficult it is to classify the target data set. Firstly, the data points are mainly clustered around a single point with no noticeable separation between the two target classes. Secondly, the target points relative to the features have high variance, this is shown through the scattered points throughout the decision boundaries. These issues affect the prediction quality of the tree models as there is no straight forward classification between the target data sets.

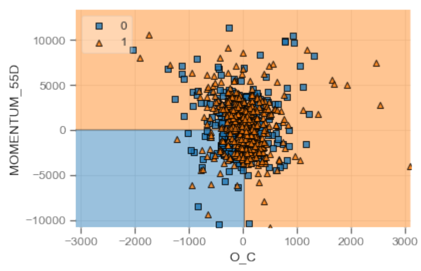

Figure 10: Open-Close and Momentum 55D Decision Boundary

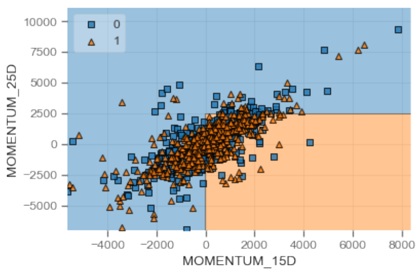

Figure 11: Momentum 15D and Momentum 25D Decision Boundary

Problem 3

Ensemble learning

- Experiment whether to use AdaBoost or XGBoost, with which inputs: tree regressor or classifier, which specific tuned hyperparameters, pruned or non-pruned. Do not present all runs but write a brief empirical study design for (3) a boosted model of your choice – three paragraphs of text minimum.

Solution. Four different boosted models were investigated with their hyper-parameters tuned from a time series split grid search cross-validation. Similar to Problem 2, the same feature set (Table 4) and cross-validation scoring `balanced_accuracy` was used for classification models while `r2` was used for regression models. For some, the time series split was reduced from five to three to decrease the calculation time of the cross-validation. The results for the best hyper-parameter tuned models for each boosted model are outlined in Table 7. For this analysis and for the purpose of comparing discrete classifications accuracy, the regression predictions have been converted into buckets with upward predictions classified by probabilities over 50% and downward predictions classified by probabilities equal or under 50%. Additionally, log loss was calculated for the classifier models by using the `predict_proba` function and taking the probabilities for the true (upward) prediction.

Boosted Model	Train Accuracy	Train Log Loss	Test Accuracy	Test Log Loss
XGBoost Classifier	0.5764	0.6931	0.5517	0.6931
AdaBoost Classifier	0.5889	0.6709	0.5471	0.6934
XGBoost Regressor	0.5931	0.6872	0.5669	0.6905
AdaBoost Regressor	0.5646	0.6856	0.5350	0.7017

Table 7: Boost Model Results

XGBoost vs AdaBoost

The XGBoost models were found to have a higher test accuracy and lower test log loss compared against their AdaBoost counterparts. The XGBoost model's were also found to be less prone to overfitting. As seen and discussed in Problem 2, predicting financial time-series price trends have a high degree of noisy data. As a result, models predicting future trends have a tendency to overfit the training data. XGBoost is able to handle noisy data more efficiently than AdaBoost as it is better at generalizing the data. This is mainly due to the fact that XGBoost models have regularization and a larger range of hyper-parameters that reduce overfitting.

Firstly, the regularization hyper-parameters gamma and lambda were used in the XGBoost models to help penalise pruning of the decision trees. Gamma contributes to the objective function by introducing a higher penalty when splitting terminal nodes. Lambda on the other hand, penalises the objective function by the score of each leaf. These two regularisation hyper-parameters have created models which are simpler and thus overall lowered the overfitting compared against the AdaBoost models.

Secondly, the two XGBoost models also improved their overfitting through tuning the following other hyper-parameters, `colsample_by_tree, min_child_weight, subsampling` and `early_stopping_rounds`. The `subsampling` parameter allowed for the internal decision trees to only be trained on a randomly selected sample data. This forced the model to find alternative classification of the data and reduced the variance of the model. Furthermore, the parameter `early_stopping_rounds` significantly helped to reduce overfitting as the model stopped training after the evaluation metric increase (or decreased depending on the metric used) by three consecutive rounds on the testing data set. As a note, it was found that XGBoost `scale_pos_weight` helped to re-correct target class weightings for the imbalanced data set, as a result 0.9 `scale_pos_weight` (which was calculated by $\frac{\sum(\text{negative instances})}{\sum(\text{positive instances})}$ referred from Table 1) was used in the XGBoost Classifier model.

Lastly, the AdaBoost model is built through aggregating weak learner decision trees with the focus on improving the largest inaccurate predictions. This improves outlier prediction which can be beneficial for low noisy data sets. However, for noisy data sets, this could be counter productive because outlier points in the training data set are learned by the model causing higher variance. Due to the combination of the discussed topics, the XGBoost model was found to be a better fit for predicting price trends rather than AdaBoost.

Regressor vs Classifier

The main advantage of using a Regressor model for binary classification is that it allows for values between zero and one to be observed which can be interpreted as probabilities. However, since the Regressor model should be built on the basis that the correct probabilities are predicted, then the data set should not be rebalanced and particular hyper-parameters should not be used to balance the data set internally. If these techniques are implemented then the Regressor models probability predictions will not be a correct representation of the data set. Because of this, the `scale_pos_weight` hyper-parameter used in the XGBoost Classifier model was not used in the Regressor model. References from the XGBoost documentation, instructed the use of the hyperparameter `max_delta_step` to be used instead. This regularization parameter applies a cap to terminal leaf weights which may help with model convergence on data sets which are imbalanced.

Considering only the XGBoost models, the Regressor model can be argued to have better prediction metrics rather than the Classifier model. When evaluating the best machine learning model it is important to examine the bias-variance trade off between the models. For this case when solely looking at the models probability predictions, the XGBoost Regressor model has lower bias but higher variance than the XGBoost Classifier model. This is evident by a comparison between the log-loss metrics with the Classifier model's training and testing loss being the same at 0.6931 and the Regressor model's training loss at 0.6872 and testing loss at 0.6905. Therefore, on a probability prediction basis, the Regressor model would be expected to perform better.

Additionally, when evaluating the models on a discrete classification accuracy, the Regressor model had a lower bias and higher variance than the Classifier model. This can be seen from the Classifier model's training accuracy recorded at 0.5764 and testing accuracy at 0.5517. Alternatively, the Regressor model's accuracy for training was 0.5931 and testing was 0.5669. Interestingly, both models are still overfitting the data with the Classifier model being slightly better. The confusion matrix's for both models can be seen below, the Regressor model has performed better on improved predictions on true negatives (downward trends) and false positives, while, false negatives predictions performed worse.

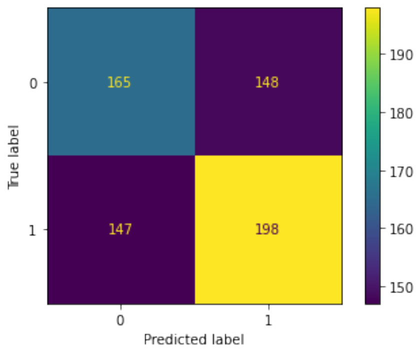

Figure 12: XGBoost Classifier

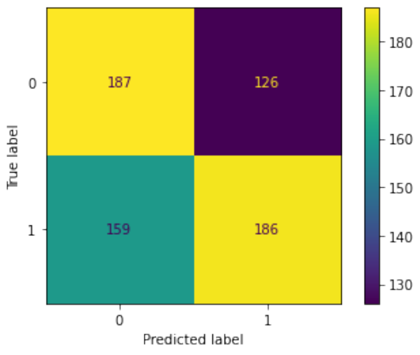

Figure 13: XGBoost Regressor

Summary

In conclusion, the AdaBoost models were not found to be better for predicting price action trends compared against the XGBoost models. The main cause of this was due to the AdaBoost model not being well equipped to handle noisy data sets, such as the finance time series data set being used. On the contrary, XGBoost performed better with both the Classifier and Regressor models performing well on a balanced accuracy stature. The XGBoost Regressor model has a higher variance than the XGBoost Classifier as the differences between the training and testing performance metrics are slightly smaller for the XGBoost Classifier model. However, it should be considered that this higher variance is outweighed by the XGBoost Regressor model performing better on a bias metric. It can be argued that the increased accuracy and decreased loss in the XGBoost Regressor model is better overall, even with the incremental increase in variance. Considering the bias-variance trade off, the XGBoost Regression Model can be thought as a slightly better model. The XGBoost Regression model hyper-parameters are listed in Table 8.

Hyper-Parameter	Value
colsample__bytree	0.7
gamma	1
lambda	1.5
learning__rate	0.01
max__delta__step	2
max__depth	3
subsample	0.7
early__stopping__rounds	3
eval__metric	logloss
objective	reg:logistic

Table 8: XGBoost Regressor Hyper-Parameters

Problem 4

Provide P&L backtesting plots (cumulative returns) for Kelly optimal bets vs. 100% bets. Kelly allocation to the asset computed with probability p taken from predict proba() is as follows:

$$p - (1 - p) = 2p - 1$$

Therefore, you remain in cash for $1 - (2p - 1) = 2(1 - p)$ percentage, which reduces your gain (and loss).Worked example: allocation 75% into a risky asset that moves up 5%, while 25% remained in a brokeraccount earning 0%, gives the total gain of $0.75 * 0.05 + 0.25 * 0 = 3.75\%$.

To obtain sensible probabilities from trees method you might need to use Regressor or non-optimal min samples leaf. Assume daily betting on price direction. Use the realised return value with PREDICTED sign to compute profit (loss) for end of the day. P&L can have a more profitable path than the asset price evolution (long-only strategy) because of successful bets on negative moves.

Solution. The python package `quantstats` was used to evaluate the performance metrics on the XGBoost Regressor model in Problem 3. Figure 14 and Figure 15 show the cumulative return of the 100% Strategy and Kelly Optimal Strategy respectively against the benchmark of Bitcoin returns. Without taking into consideration exchange fees, both strategies were better than the long only strategy of holding bitcoin.

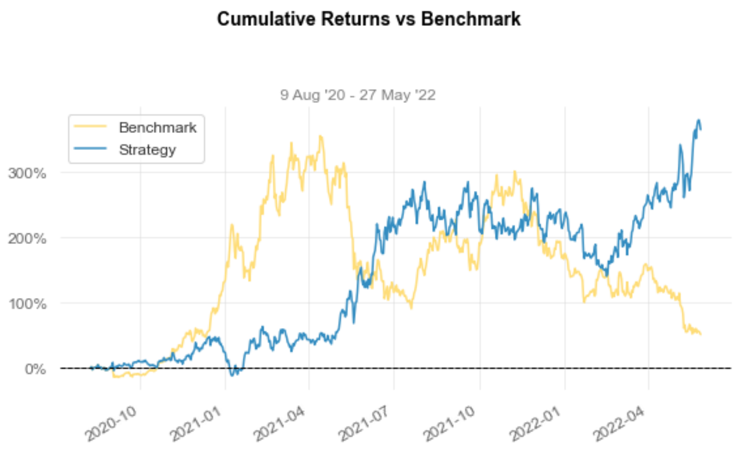

Figure 14: Cumulative Returns for 100% Strategy

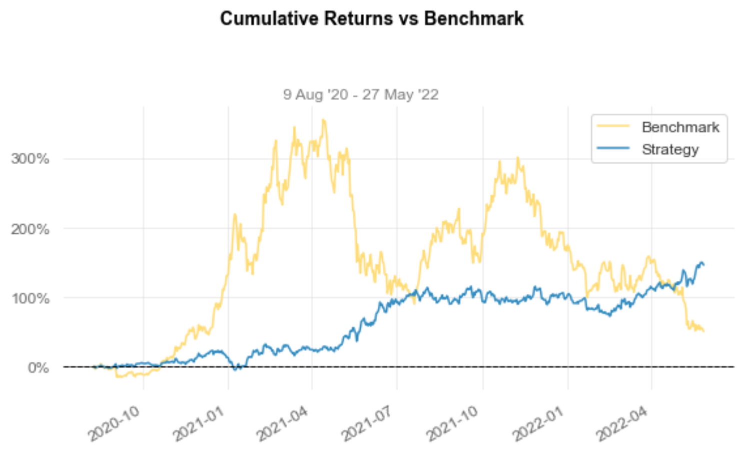

Figure 15: Cumulative Returns for Kelly Optimal Strategy

As expected the 100% Strategy was found to have a higher return while the Kelly Optimal Strategy had lower risk. On a risk-to-reward basis however, both strategies were matched with an approximate average Sharpe ratio of 1.27. This indicates that the Kelly Optimal Strategy on the XGBoost Regression model does not add any risk-to-reward benefit compared against a 100% Strategy. After analysing the probability outputs from the XGBoost Regression, it was evident that the probabilities do not move further than 5% either side of 50%. Therefore, this causes there to be hardly any benefit incurred from using the probability as the position sizing metric.