UNSW

School of Software Engineering, Faculty of Engineering

The University of New South Wales

Predictions on Australian Horse Racing Results

COMP9417: Machine Learning and Data Mining

Group Project

Student ID	Student Name
z5075556	Cameron Horsley
z5185318	Chukwuemeka Eze
z3242416	Chung-Yu Liu
z5112589	Kevin Ly
z5345728	Muhamed Shafiq

Table of Contents

Section 1: Introduction

1.1 - Problem Description

Automated machine learning-based betting systems have become increasingly popular, operating profitably on sports betting exchanges by predicting the outcome of events and placing data-backed wagers over the internet. This growth has been spurred by data becoming increasingly descriptive, accurate and accessible in large volumes over the internet, coupled with exponential advances in computational power and data storage. Predictive analysis of sporting matches is a popular topic in recent research, particularly when applied to the problem of horse racing win classification.

Horse racing is a hugely popular sport for bettors around the world due to their frequent occurrence and the simplicity of event outcomes. Horse racing is widespread in Australia, with races occurring in every state and territory every day. A horse race involves 5-20 horses running in a row, with the fastest crossing the finishing line first. The growth of the Internet has allowed clean and comprehensive horse racing data from reliable sources to be readily and easily accessible for collection.

1.2 - Motivation for the Project

The primary motivation for this project was to attempt to build a predictive model that could eventually be used in an automated system to place profitable bets on sporting markets. Given the data we collect is available in full before races are run, there exists potential to apply the generated model in real-time before the race and place automated bets in an online betting market.

Disclaimer:
One of our group members (Cameron Horsley) is an undergraduate working on an independent thesis which is related to this topic - a proposed architecture for an automated system of machine learning-based decision making. The application for this thesis is an automated process of querying machine learning models for predictions on the outcome of horse races in Australia, and placing automated bets based on profit-seeking goals. Critically, the thesis does not focus on the generation of machine learning models, but rather applying predictions of models in a real-time and automated manner on live markets - leaving space for this report to explore the generation of models without any overlap. The resulting model generated in this project may be used in Cameron's thesis to generate predictions in order to test the functionality of the automated betting system architecture.

1.3 - Project Goals

The goal of this project was to apply various machine learning techniques and algorithms to solve the classification problem of predicting winning horses within a race. We trained and tuned several models on real data from Australian horse racing betting markets and compared their

results using standard model evaluation metrics. We conclude with a decision of the model which is best suited to the win classification problem and the selected dataset.

Section 2 will describe the horse racing data that was used and the way it was collected, as well as preprocessing and transformation techniques that were applied to the dataset. Section 3 will detail the methods of feature selection that were applied to reduce the dimensionality of the dataset into important features. Section 4 will provide detail into the models that were trained, including their loss functions and hyperparameter tuning. Section 5 will provide an evaluation of the models, including a decision of which model performed best. Section 6 will explore potential real-world applications for this project, as well as suggest future extensions to this work.

1.4 - Analysis of Existing Literature

Horse racing has become a popular topic for machine learning research due to the accuracy of the data and the simplicity of the events. As such, there have been various attempts in recent literature to generate models based on horse racing data that make predictions on the performance of horses and the outcome of races. Before making our attempt at creating models, we explored several reports detailing previous approaches to gather ideas on what techniques and algorithms we could apply to our models.

Neural networks were identified as being highly expressive to learn complicated non-linear relationships in the input data and is becoming the dominant approach for many problems [11]. Neural networks have been applied to many prediction problems in horse racing such as the win classification problem, and have been explored by several researchers to considerable success due to the unbalanced nature of horse racing datasets and the complexity of the data. Williams and Li [12] applied four neural network algorithms to the win classification problem on 143 races in Jamaica (Back Propagation, Quasi-Newton, Levenburg-Marquart and Conjugate Descent), with each method performing better than a 74% accuracy and Back Propagation being the most performant. Eight variables describing the horse's and jockey's performance were used as input neurons, with one neural network representing one horse. Cheng and Lau [13] also applied neural networks to the win classification problem and logistic regressions on the finishing time of horses, both realising positive net gains when applied to bets with a confidence threshold of 95%. Liu [2] suggested that binary classification of win or lose may be unfeasible as the "win" class is underrepresented with only 10% of data records and that problems may arise due to the asymmetric costs of misclassifying classes with normal classifiers. Based on this, Liu decided to instead attempt to predict the finishing time of horses. Similarly to the approach we will make in our paper, Borowski and Chlebus [14] trained and tested six classification models against a set of success criteria - Classification and Regression Tree (CART), Generalised Linear Model (Glmnet), Extreme Gradient Boosting (XGBoost), Random Forest (RF), Neural Network (NN) and Linear Discriminant Analysis (LDA). They concluded that there is considerable evidence that machine learning can be applied to horse racing predictions to operate profitably. Several papers suggested applying transformations such as Z-score normalisation to reduce feature sensitivity. Models were evaluated through several metrics such as loss (MAE, MSE), accuracy of bet decisions, and net gain (profit).

Section 2: Data, Preprocessing and Transformation

2.1 - Data Source and Collection Method

The data used in this project was collected from Punters.com.au through the use of web scraping. Punters is the fastest growing racing site in Australia, primarily being popular for the comprehensive form guide they offer free to download. Their form guide is available for every race occurring across every race track in Australia and is available in CSV format, containing a row for each horse in the race and features that summarise the recent performance of the horse, jockey and trainer. Historic races from the horse which match the features of the current race are also summarised, such as those with the same distance, track condition or weather. Crucially, the data is updated after the race is concluded with the finishing position of each of the horses, providing us with a target variable.

The data was collected through the use of an automated process using a Selenium webdriver which scrapes the ID of each of the races occurring in the day, and then proceeds to download the form guide CSV.

2.2 - Preprocessing

2.2.1 - Cleaning

Firstly, columns in the dataset that do not contain useful information are dropped, such as columns for the names of the horse, jockey and trainer and their profile URL links on Punters.com.au. The target column, “Finish Result” contained integers with the place each horse finished and needed to be converted into binary for “winning” and “losing” horses - 1 for horses that finished first and 0 for second place or worse. The “Apprentice” column indicating jockeys-in-training also needed to be cast to binary - 1 for Yes and 0 for No. The categorical “Gender” column was one-hot-encoded, where new columns were created for Filly, Mare, Gelding, and Colt. Missing values were imputed into columns largely with the mean of the column, however some columns where missing values held a special meaning needed to be handled differently. Missing values in “Average Prize Money”, “Last Start Prize Money”, “Last Start Distance” and “Last Start Finish” indicate a horse that hasn’t raced before and was filled with zero so that the performance of these horses were not inflated to the mean of the dataset.

2.2.2 - Feature Engineering

Additional features were generated from the initial dataset to draw out information that was not immediately present in the data. New columns were created to indicate which horses were racing for the first time, as well as first time jockeys and trainers. Additionally, new columns were created for each feature which ranks the horse within its race, for example the horse with the highest career winnings in a race of ten horses was assigned a value of 1 and the horse with the lowest career earnings was assigned a value of 10. The reasoning behind generating a

ranking column for each feature was to capture the strength of a horse relative to other horses in their race.

2.2.3 - Normalisation

Since the goal of this project was to attempt to classify winning horses and losing horses within a race, it was crucial to capture some relativity of their performance within the race groupings, as any model trained on the data will treat each horse's row independently. Given the initial features of the dataset contained only summarised data for the historic performances of each horse, there was no information about how competitive their current race was. Following the same methodology as generating rankings, each feature was normalised within their race grouping through two methods - z-score and chi-squared score. Both methods allow some information to be gained about how far away each of the horse's features are from the expected value (mean) of the race.

$$\textbf{Z\_score} = (X - \bar{X}) / \sigma \qquad \textbf{Chi\_sq} = (X - \bar{X})^2 / \bar{X}$$

X: The feature for the current horse
$\bar{X}$: The mean of the feature within the current horse's race
Σ: The standard deviation of the feature within the current horse's race

Given the initial quantity of features within a race (129), generating a rank, z score and chi-squared score for each feature quadrupled the number of features in our dataset. These were then reduced during feature selection to avoid issues related to the curse of dimensionality.

2.2.4 - Train Test Split and Standardisation

Splitting the data into test and train sets had to be handled carefully due to the potential future application of the model being implemented in a trading strategy. Thus, we had to ensure that race groupings were maintained during the split to allow comparisons within a race during evaluation of test predictions. Instead of splitting by row count, we split the data by race ID to ensure that all horses in a race were either entirely in the test set or entirely in the train set.

Finally, once we had split the data into test and train sets, a standard 0-1 scaler was fitted to the train data, and both the test and train data were transformed.

Section 3: Feature Selection

3.1 - Feature Selection

Several problems can arise when there are too many features in a dataset. As the dimension of the data grows, the number of samples required for a good model increases exponentially as the data becomes sparse relative to the space. Intuitively, the larger the dimension, the more

samples are needed to cover enough different combinations of features to allow for modelling. The curse of dimensionality is particularly relevant to this project, as after feature engineering was applied, our dataset consists of over 400 features. Feature selection can be used to address these problems by reducing the number of input features, with the additional benefit of improving computational cost. There are three different types of feature selection methods: filter, wrapper, and embedded.

Filtering methods choose subsets of the features based on their correlation or dependence to the target variable. Pearson, Spearman, and Kendall's tau are the three types of filter methods given in a pandas DataFrame. The Pearson correlation coefficient can be used to measure the correlation between the features and the target variable. It is used for linear relationships and requires variables to be normally distributed and not ordinal. Spearman's rank correlation coefficient is more widely applicable since it is efficient, can work with ordinal data and does not require normality. Whereas Pearson works with actual values, Spearman measures the monotonic relationship between variables. Kendall's tau measures dependence, which is a more general relation than correlation. It is generally preferred over Spearman due to its robustness and less sensitivity to error, but is slower ($O(n^2)$ vs $O(n\log n)$). It is not used here due to long computation time and overflow errors in the sklearn implementation.

Wrapper methods look at the performance of a feature subset rather than their relation to the target. They generally perform better than filter methods, but can be prone to overfitting and large computation time. Recursive feature elimination, a method where models are repeatedly fitted and the worst feature discarded each time, was tested but discarded due to slow convergence, with each iteration taking upwards of 20 minutes.

Embedded methods can combine benefits of both wrapper and filter models. These carry out feature selection simultaneously as part of a learning algorithm. An example can be seen in Section 4 with the feature importances calculated using Random Forest.

3.2 - Principal Component Analysis

As the prior feature selection methods focused on correlations with the target variable, some of the selected features are highly correlated with each other. This potentially results in multicollinearity, affecting the relationship between features and the target variable and increases the variance of our models. Having highly correlated features in our dataset is also inefficient. One method to further reduce the dimension of our training dataset is to generate a set of linearly uncorrelated principal components which are linear combinations of the scaled features. The number of principal components generated should be less than the number of features in our original data while still explaining most of the variation in the data.

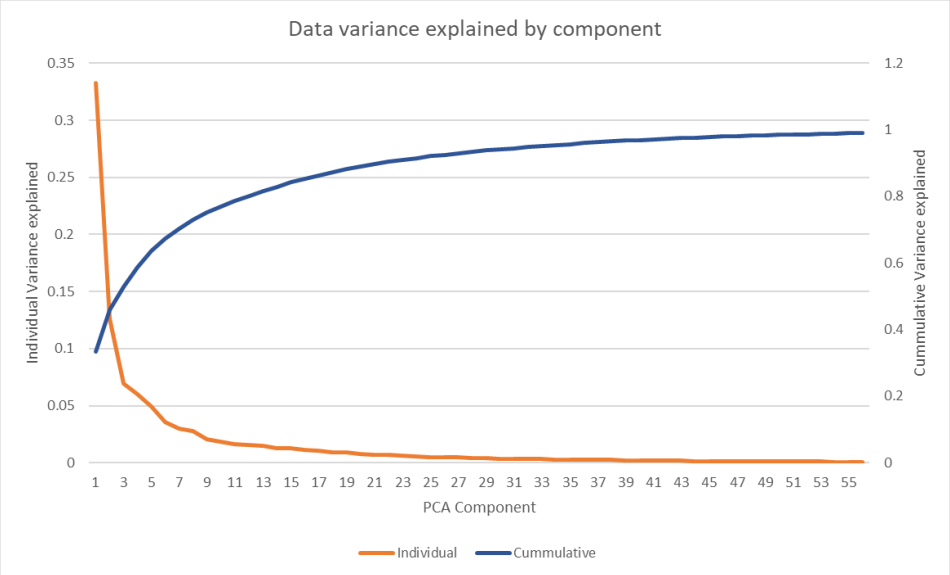
We generate principal components from our data after feature selection such that 99% of the variation in the dataset is preserved by the components. This results in the following reductions:

Table 3.2.1: Feature reduction through PCA

Dataset	Features before PCA	Principal components generated
Using Pearson correlation	79	47
Using Spearman correlation	97	56

Figure 3.2.1 shows the individual and cumulative variance explained respectively by adding the next principal component.

Figure 3.2.1: Variance explained by component



Both the test and training data are transformed using principal components generated from the training dataset. One of the outputs of the PCA is the correlation matrix between principal components (rows) and features (columns) (see Appendix 3.2.1). Each PCA will be correlated with some of the original features which acts similarly to clustering correlated features together. The key features in the top 5 (by variance explained) principal components are:

Table 3.2.2: Top 5 features (by variance explained) in the top 5 PCA components (by variance explained)

PCA	Top 5 features
1	Trainer Last Season Horse Earnings_race_z_score, Trainer 12 Month Horse Earnings_race_z_score, Trainer This Season Horse Earnings_race_z_score, Trainer 12 Months Places_race_z_score, Trainer Last Season Wins_race_z_score
2	Jockey 12 Months Strike Rate_race_z_score, Jockey 12 Months Place Strike Rate_race_z_score, Jockey Last 100 Place Strike Rate_race_z_score, Jockey This Season Place Strike Rate_race_z_score, Jockey Last 100 Places_race_z_score

3	Prize Money_rank, First Time Jockey_rank, First Time Trainer_rank, gender_other_rank, Last Start Margin_rank
4	Trainer This Season Strike Rate, Trainer 12 Months Strike Rate, Trainer Last 100 Strike Rate, Trainer 12 Months Place Strike Rate, Trainer This Season Place Strike Rate
5	Last Start Margin_race_z_score, Last Start Prize Money_race_z_score, Career Place Strike Rate_race_z_score, Best Fixed Odds_race_z_score, Average Prize Money_race_z_score

---

Section 4: Model Training

4.1 - Approach to Modelling

Based on our review of existing literature, we decided on a set of models which were most likely to have success in predicting this binary classification problem - eXtreme Gradient Boosting, Neural Networks, Logistic Regression and Random Forest. Each of these models were trained on the training dataset, with cross-validation applied through a gridsearch during hyperparameter tuning.

Random forests and gradient tree boosting models are both ensembles of weak decision trees learners. Random forests is the mean predictor of a set of decision trees learners while for gradient boosting, trees are iteratively fitted on the residuals and added to the predictor. For gradient boosting, we use eXtreme Gradient Boosting, or XGBoost, which is a scalable gradient tree boosting algorithm developed by Chen 2016 [4]. Its speed and performance meant it has seen wide use in machine learning competitions such as Kaggle. The output of the model will be probabilities the horse will win, where we deem if the probability > 0.5, the model predicts the horse will win, otherwise it will lose. We use xgboost package with GPU enabled to improve the training speed so that we can explore a larger hyperparameter space.

The neural network structure was based on the structure used by Liu (2018) on the usage of machine learning in horse racing predictions [2]. The structure is a deep neural network, which makes use of two densely connected layers. In densely connected layers, every neuron in the layer has a connection with every neuron in the preceding and the succeeding layer [3]. Immediately after each dense layer, the model applies batch normalisation to the output of the dense layer, ensuring that the input into the next layer is standardised, improving the training speed and the model's stability [5]. Immediately after batch normalisation, dropout is applied to the outputs. Note that this only occurs during training as a regularisation method, and the model disables dropout during evaluation of data [6]. The activation function at the output of the dense layers is the ReLu function, which is the most commonly used activation in modern deep learning networks [7]. At the final output node, the sigmoid activation function is applied as the target output is binary, and we use 0.5 as the threshold between the 1 (wins) and 0 (losses) classes. The loss function used during training is binary cross entropy, which is the most commonly used loss function for binary classification [8]. As in the paper by Liu in 2018 [2], the

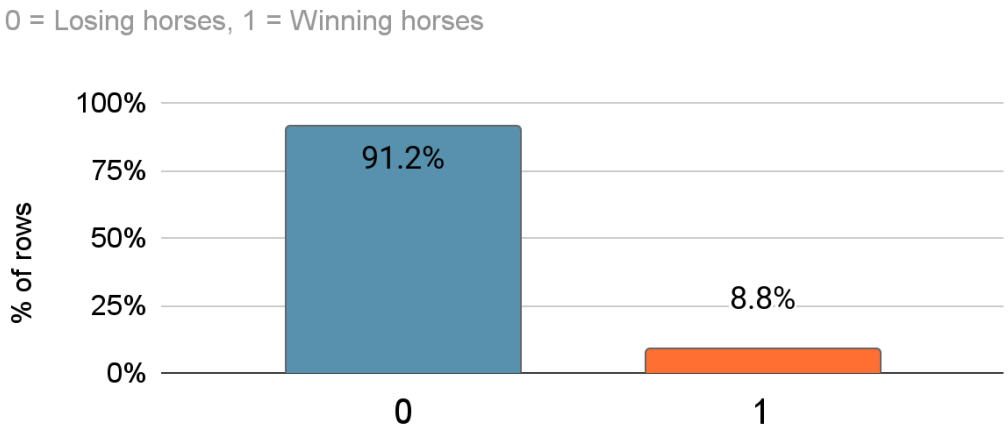
weights of each dense layer is initialised randomly according to a Normal distribution with $\mu = 0$ and $\sigma = 0.02$, and all biases are initialised to 0.For this model, we used the Keras API [9] from Tensorflow to build and evaluate the model. We also use early stopping to cease training if the ROC AUC score is not improving over the past 10 epochs, to cut down on training time and also reduce overfitting.

Logistic Regression is a popular approach to binary classification problems and can be used to model the probability of one event (win or lose) occurring, by having the log-odds for the event to be a linear combination of the selected independent variables that best explain the outcome. The output is then passed through a sigmoid function which bounds the output in [0,1] to give the probability of the outcome. Optimization function is applied to reduce the difference between the expected outcome and the true value. This is done repeatedly until a minimum loss condition is achieved. Based on the cutoff value of the final outcome the model predicts the output class.

4.2 - Handling Unbalanced Data

The dataset is unbalanced by nature, with an approximate 10/90 split between the positive class (winning horses) and the negative class (losing horses). This translates to an average of 10 horses in a race with only a single winning horse per race. This posed several challenges during model training and had to be addressed.

Figure 4.2.1: Class Distribution



Some research suggested over-sampling or under-sampling the dataset when training to address the unbalanced data, however doing so will change the distribution of the data and therefore may not be appropriate. Instead we decided to implement class weightings in each of the models, upweighting the minority class using a gridsearch during hyperparameter tuning.

We also needed to revise the evaluation metrics in light of the unbalanced data, as accuracy alone was not a particularly good indication of model performance in picking winning horses. Generally, in the initial models without the class weightings, accuracy was very high but recall was particularly poor. However when using class weightings, recall improved significantly at the

expense of a slight drop in accuracy and an expected drop in precision. AUC was used as the primary evaluation metric as it handles unbalanced data the best.

Table 4.2.1: Train metrics for models with/without class weightings

	Train Metrics							
	AUC score		Accuracy		Recall		Precision	
Class Weights Applied	No	Yes	No	Yes	No	Yes	No	Yes
Logistic Regression	0.759	0.760	0.904	0.690	0.021	0.695	0.476	0.192
Random Forest	0.796	0.871	0.908	0.765	0.058	0.809	0.784	0.264
XGBoost	0.637	0.897	0.963	0.844	0.275	0.963	0.976	0.384
Neural Network	0.814	0.814	0.905	0.578	0.011	0.900	0.708	0.172

4.3 - Hyperparameter Tuning with Feature Selection Comparison

Determining the best model for each technique required running a gridsearch during hyperparameter tuning. As discussed in Section 3 - Feature Selection, there were also several variations of feature selection techniques that we could apply to the data. Considering this, feature selection was tested on each model to determine which was best suited for each technique.

4.3.1 - eXtreme Gradient Boosting

Hyperparameter Optimisation

For our training, we consider the following hyperparameters with the corresponding grid we are exploring along with the rationale and impact. We then perform a 5-fold cross-validation on the training dataset and rank each combination of hyperparameters by the average AUC score on the CV set. The AUC score provides a balanced view of a model's recall and false positive rate.

From initial testing, it was found that the auc score didn't vary much by changing "colsample_bytree", "subsample", "reg_lambda", whereas changing the other parameters had a more pronounced impact on the cross validation score as shown in Appendix 4.3.1.1. Based on this, we refined the hyperparameter space to perform a more in-depth search for the more relevant hyperparameters. Going forward, we may consider using Bayesian hyperparameter search to perform a more efficient search on a larger hyperparameter space.

Table 4.3.1.1: XGBoost hyperparameter optimisation

Parameter name	Rationale and impact	Search space	Refined Search Space
sample_weight	Weight of sample introduced to balance classes in data	5, 10, 12	5, 10, 12
max_depth	Increases the model complexity to see if it improves the fit	4, 6	4, 6, 8
learning_rate	Needs to be fast enough to reach the optimal solution but not too fast that it overfits to training data	0.01, 0.005	0.01, 0.02, 0.03
reg_alpha	L1 regularisation	100, 200	50, 100, 200
reg_lambda	L2 regularisation	0.1, 1	N/A
colsample_bytree	Randomly samples features used in each tree to reduce overfitting	0.6, 0.8, 1	N/A
subsample	Randomly samples data used to fit each tree to reduce overfitting	0.8, 1	N/A
n_estimators	Balance number of trees between underfitting and overfitting	500, 1000, 1500	1000, 1500, 2000

Feature Selection Method Comparison

5-fold cross-validation was applied with a gridsearch for optimal hyperparameters for each feature selection technique. Based on the results, the best model is built on the training data with Spearman correlation feature selection without PCA. There is negligible difference between results using Spearman correlation compared to Pearson correlation for feature selection. The final submitted model will be built on data with Pearson feature selection without PCA using the full training set with the optimal hyperparameters selected. The model is then scored on the test set for evaluation.

Table 4.3.1.2: Hyperparameter optimisation by feature selection method for XGBoost

	Metric	Optimal Hyperparameters				
Feature Selection	**AUC**	**sample_weight**	**max_depth**	**learning_rate**	**reg_alpha**	**n_estimators**
Spearman (no PCA)	0.824	10	4	0.02	50	1000
Spearman (with PCA)	0.808	5	6	0.01	50	2000
Pearson (no PCA)	0.823	10	4	0.01	50	2000
Pearson (with PCA)	0.808	5	6	0.01	50	1500

4.3.2 - Neural Network

Hyperparameter Optimisation

For our training, we consider the following hyperparameters with the corresponding grid we are exploring along with the rationale and impact. We then perform a 2-fold cross-validation (CV) on the training dataset and rank each combination of hyperparameters by the ROC AUC score on the CV set. The CV is reduced to 2-fold CV to reduce computation time, due to the high time complexity of training a deep neural network. Additionally, due to the high complexity of hyperparameter combinations, the layer 1 and 2 nodes will be tested in a separate grid search after determining the other optimal hyperparameters. Initially, default values of 32 and 256 nodes will be used in layer 1 and 2 respectively, as was used by Liu in their 2018 thesis [2].

Initial testing revealed several hyperparameters were not effective in increasing the ROC AUC metric. Optimiser choice was not a large factor, but SGD slightly outperformed Adam and so it was selected for further hyperparameter testing (see Appendix 4.3.2.1). Batch size also did not prove to be a significant factor (see Appendix 4.3.2.2). A batch size of 32 presented a more consistent performance, while a batch size of 64 had a higher maximum performance but significantly more variance. Therefore, since the difference is quite small, the more consistent batch size of 32 was chosen for future hyperparameter testing . It also appears the learning rate does not present much performance improvement, and as with batch size a lower learning rate provides more precise and consistent performance (see Appendix 4.3.2.3). Learning rates less than 1e-04 appear to be learning much too slowly and cannot reach a good performance in a reasonable timeframe. Therefore, we will continue with a learning rate of 0.001, which performed the most consistently high. Finally, momentum applied to the optimiser does not appear to make a large difference either (see Appendix 4.3.2.4). Momentum being applied appears to decrease the variance of results, although the highest ROC AUC score was with no momentum (momentum = 0). Therefore, momentum will not be used for the model.

Based on this, we refined the hyperparameter space to perform a more in-depth search for the more relevant hyperparameters. Dropout achieves a significant improvement in performance, with the best performance at dropout of 0.5 (see Appendix 4.3.2.5). Hidden nodes in both layers are optimal at around 32 nodes, and the best combination is also (32, 32) for the layers (see Appendix 4.3.2.6 and Appendix 4.3.2.7). This parameter was surprisingly not a high impact factor in performance, with worst performance around 0.797 ROC AUC and the best performance around 0.806 ROC AUC. This is likely because we are using both batch normalisation and dropout as regularisation methods, which is useful for preventing the model from overfitting. Surprisingly, the class weighting did not have a large influence on results (see Appendix 4.3.2.8), but the best performance was with a weight of 15 on the winning class.

Table 4.3.2.1: Neural Network hyperparameter optimisation

Parameter Name	Rationale and impact	Search Space	Refined Search Space	Optimised Value
Layer 1 Nodes	The number of nodes in the layers correspond to the number of parameters/complexity in the network, and is therefore one of the most important parameters to analyse.	8, 16, 32, 64	8, 16, 32, 64	32
Layer 2 Nodes		32, 64, 128, 256	32, 64, 128, 256	32
Dropout Rate	Differing dropout will improve performance depending on how much noise/uncorrelated features are present.	0.1, 0.4	0, 0.1, 0.25, 0.5, 0.8	0.5
Learning Rate	Certain learning rates are more likely to reach an optimum, whereas some lower values may get stuck at local optimums or cause instability.	1e-5, 1e-4, 1e-3, 0.01	N/A	0.001
Momentum	Momentum may reduce computation time, and may also assist in preventing optimisers getting stuck at local optimums.	0, 0.9	N/A	0
Batch Size	Batch size will present a tradeoff between accuracy and speed of computation.	32, 64	N/A	32
Optimiser	Two of the most commonly used optimisers for deep learning will be compared.	SGD, Adam	N/A	SGD
Class Weights	We will fix the 0 (losses) class' weight at 1.0, and perform a search on the optimal class weight for the 1 (wins) class.	1, 5, 10, 15	1, 5, 10, 15	15

Feature Selection Method Comparison

The previous iterations of hyperparameter tuning were completed using Pearson correlation used to select features. The optimised hyperparameters from this section will be used for all feature selections, due to the high time of computation and the low likelihood that the performance will be noticeably increased by rerunning grid search with all feature selection methods. Therefore, the model with the previously chosen hyperparameters are trained on the data resulting from the feature selection method, and then the resulting metrics are calculated on the model's performance on the test data.

The performance does not vary much, although the Spearman method is slightly better, though the improvement is insignificant. PCA does not make much of a difference, and we can see a very minor reduction in performance when using PCA vs the no PCA counterparts. Therefore, the final submitted model will be on the Spearman method with no PCA and the previously chosen optimal hyperparameters. Note that upon rerunning the model, performance may slightly vary due to the large amount of randomness involved in neural networks, including during weights/bias initialisation and in dropout during training.

Table 4.3.2.2: Neural Network comparison of feature selection methods

	Metrics									
Feature Selection	AUC	Log loss	Accuracy	Precision	Recall	F1 Score	FP	FN	TP	TN
Pearson (no PCA)	0.8108	0.7067	0.5822	0.1721	0.8862	0.2882	6895	184	1433	8430
Spearman (no PCA)	0.8138	0.2546	0.5776	0.1722	0.8998	0.2891	6995	162	1455	8330
Pearson (PCA)	0.8083	0.7111	0.5895	0.1736	0.8775	0.2898	6757	198	1419	8568
Spearman (PCA)	0.8120	0.6892	0.5683	0.1689	0.8986	0.2843	7150	164	1453	8175

4.3.3 - Logistic Regression

Hyperparameter Optimisation

GridsearchCV was performed using 5-fold cross validation on the training data based on a scoring criteria that provides for selection of the best parameters. Considering the high level of imbalance of 91.2% to 8.8% against the “win” class, AUC score was ranked first followed by recall score and then, Accuracy. Table 4.3.1.1 gives an overview of the parameters that optimised the algorithm.

Table 4.3.1.1: Logistic Regression hyperparameter optimisation

Parameter name	**Rationale and impact**	**Search space**	**Optimal value**
Sample_weight	Weight of sample introduced to balance classes in data	balanced	balanced
Learning_rate	Needs to be fast enough to reach the optimal solution but not too fast that it overfits to training data	0.05 - 1.0	0.2
Penalty	Regularisation applied to penalise less relevant predictor variables	'L1', 'L2', 'None'	‘L1’
Solver	Optimization algorithm for loss minimization	Liblinear, newton-cg, saga, lbfgs	liblinear
max_iter	Maximum iteration	100 - 5000	100

4.3.4 - Random Forest

Hyperparameter Optimisation

For the Random Forest model, grid searching and stratified K-fold cross-validation were applied and a subset of the training set was used for validation. Over 30 different iterations were done based on different hyperparameter values and feature selection methods. In each iteration, 1-4 values were specified per hyperparameter. These started far apart from each other, and would be altered to converge on some value in subsequent iterations. For example, the class_weight

in an initial iteration would be [{0:1, 1:5}, {0:1, 1:10}, {0:1, 1:15}] (using the fact that there is a ratio of roughly 10-1 losers to winners in the data). If the best model used {0:1, 1:10}, then the class_weight would be set to [{0:1, 1:7.5}, {0:1, 1:10}, {0:1, 1:12.5}] in the next iteration.

From this experimentation, it became clear that the optimal value of some hyperparameters was near-constant. These included max_features = sqrt(n_features), max_leaf_nodes = None (unlimited), and min_samples_split = 2. n_estimators also did not significantly affect the results and were set to a relatively low value of 50 for decreased computational time.

Table 4.3.4.1: Random Forest hyperparameter optimisation

Parameter name	Rationale and impact	Search space	Optimal value
max_depth	Maximum height of trees	3, 15	12
n_estimators	Number of trees	10, 200	50
min_samples_leaf	Minimum number of samples after splitting	1, 60	35
class_weight	Compensates for data unbalance	{0:1, 1:1}, {0:1, 1:15}	{0:1, 1:9.75}
max_features	Maximum features for one tree	"sqrt", "log2", "None"	“sqrt”
max_leaf_nodes	Limits splitting of nodes	2, None	None
min_samples_split	Number of observations to split	2, 20	2

4.4 - Feature Importance

Along with the model prediction, we would want to know what features are important in predicting whether a horse will win. An output we generate for each model is the permutation feature importance. Permutation feature importance [10] evaluates features’ importance by permuting features from a validation set and evaluating the change in performance.

The top 5 important features are compared in Table 4.4.1. There is a large overlap in the top 5 features identified in each model, aside from logistic regression which is largely expected. Information relating to fixed odds and trainer record in the current season appear to be the most significant in determining the outcome of the horse in the race for most models. Interestingly, the logistic regression model seems to deviate slightly on important features, deeming trainer-related features to be more significant.

Interestingly, the PCA analysis did not rank features relating to Best Fixed Odds highly, whereas they were deemed extremely important in many of our models. This is likely due to the fact that there is a large amount of correlation between the Best Fixed Odds and other features, meaning the column does not provide additional information that is not present in other features. However, our models do not recognise this and prefer to use this feature heavily in its predictions.

Table 4.4.1 - Top 5 important features by model

	Neural Network	XGBoost	Random Forest	Logistic Regression
Best Fixed Odds_rank	1	2	1	
Best Fixed Odds_race_z_score	2	4	3	
Trainer This Season Strike Rate_race_z_score	3		4	
Trainer This Season Strike Rate	4		5	
Trainer 12 Months Strike Rate_race_z_score	5			
Best Fixed Odds		1	2	
Trainer This Season ROI		3		
Trainer This Season ROI_race_z_score		5		
Trainer 12 Months Wins				1
Trainer 12 Months Places				2
Trainer This Season Wins				3
Trainer Last Season Starts				4
Trainer This Season Places				5

The feature importance for each model can be found in the Appendix.

Section 5: Model Evaluation Results

5.1 - Model Evaluation

The decision for converting a predicted probability or scoring into a class label is governed by a parameter referred to as "*discrimination threshold*," with default value, 0.5 for normalized predicted probabilities such that:

- Prediction < 0.5 = Class 0
- Prediction >= 0.5 = Class 1

The problem here is that the default threshold may not represent an optimal interpretation of the predicted probabilities when dealing with an unbalanced dataset. To overcome this, we try to address the imbalance in our dataset in different models used and cross-validated it in the

training set. To avoid evaluation mistakes we focus on alternative metrics suitable for a model trained with imbalanced data and ranked them as follows:

1. ROC curve was used to analyse the predicted probabilities of a model and ROC AUC scores to compare and select a model.
2. A probability-based metric, log loss (cross-entropy)
3. Accuracy was considered last. Since the discrimination threshold used in the training is default at 0.5. accuracy would be not a best metrics to consider first

Although an optimized discrimination threshold is required for class labels, choosing this threshold that results in the best balance between the true positive rate and the false positive rate is considered an extension to this project.

Table 5.1: Evaluation metrics for the original data (unbalanced)

	Metrics									
Classifiers	**AUC**	Log loss	Accuracy	**Precision**	**Recall**	**F1 Score**	FP	FN	TP	TN
NN	0.8135	0.2546	0.9051	0.7083	0.0105	0.0207	7	1600	17	15318
RF	0.7957	0.2622	0.9065	0.6222	0.0519	0.0959	51	1533	84	15274
Logit	0.8055	0.2592	0.9049	0.5357	0.0278	0.0529	39	1572	45	15286
Xboost	0.8141	0.254	0.9074	0.6395	0.0680	0.1230	62	1507	110	15263

Table 5.2: Evaluation Metrics for tuned models with class weights (balanced)

	Metrics									
Classifiers	**AUC**	Log loss	Accuracy	**Precision**	**Recall**	**F1 Score**	FP	FN	TP	TN
NN	0.8138	0.6890	0.5776	0.1722	0.8998	0.2891	6995	162	1455	8330
RF	0.7936	0.4814	0.7414	0.2199	0.6710	0.3312	3849	532	1085	11476
Logit	0.8022	0.5420	0.6931	0.2062	0.7773	0.3260	4839	360	1257	10004
Xboost	0.8217	0.506	0.7138	0.2182	0.7737	0.3404	4482	366	1251	10843

5.2 - Discussion of Best Model

Consistent with the goal of this project, if a particular model was used to place bets on winners, we have the following matrix of outcomes:

	Action	**Horse actually loses**	**Horse actually wins**
Model predicts horse loses	Don't bet	No loss/gain	Lost opportunity
Model predicts horse wins	Bet	Money lost	Money won

We observe the following:

- **Models with low recall would underestimate the chances of a horse winning leading to lost opportunities.** This is apparent on the model which does not implement class balancing, where the model has a high accuracy, but it would not be very useful in identifying key features of winning horses.
- **Models with high false positive rate (1-precision) could overestimate the chances of a horse winning which results in losses incurred from incorrect bets**. As we improve the recall through class balancing, the precision drops. Notably, the neural network model has the highest recall (around 90%) but also has a high false positive rate (82.8%).
- Based on metrics which balance recall and precision (like AUC and F1-score), xgboost performs slightly better than the other models.
- There was negligible difference in results between using Pearson and Spearman correlations as the feature selection method. There was also limited impact from applying PCA to the data before training models.
- Changing the descrimination threshold for bet decisions will likely have a significant impact on the profitability of any betting system which uses these models. Particularly a system which aims to predict winners and place bets will be exposed to higher risk as models will generally predict more winners within a race. Therefore increasing the threshold from 0.5 will likely limit the risk and exposure, ensuring that only predictions which are extremely certain will have bets placed on them.

Section 6: Real-world Applications and Extensions

The major motivating factor for horse racing predictions is for the purpose of making smart decisions on horse betting. The models produced in this project can be used to bet strategically, and could be used to produce an automated betting system, though the profitability of this can

be explored in future work. A common theme amongst the models was that the models perform extremely well on predicting if a horse will lose, driven by the unbalanced nature of the dataset where losing horses were heavily overrepresented. Therefore, the first iteration of models produced in this project can be particularly useful for “lay betting”, which involves betting on whether a horse will lose a race. Another interesting angle to approach automated betting is to seek opportunities where the market’s betting odds are in conflict with high confidence model predictions, which may reveal opportunity for profit off of the market’s lack of information.

Additionally, the features included in the dataset could be endlessly expanded, to include data such as expert tips, price forecasts for the betting market, context of the race (i.e. weather, other horses in the race), change in betting prices leading up to the race, previous speed map/race times for the horses.

In their current state, our models do little to consider the “quality” of races, that is; the best horses race against each other and the amateur horses tend to race against each other. Another potential extension to this project could be the use of unsupervised clustering to rank races in quality, and use this ranking as a feature in the prediction models.

Binary classification is a very naive approach to apply to horse racing due to the large amount of immeasurable factors that influence race outcomes. Therefore, future work could look at predicting the finishing placement of horses, which will allow for much more information to be gathered from the models for deciding bets.

References

[2] Y. Liu, "Predicting Horse Racing Result with Machine Learning", Undergraduate, The Chinese University of Hong Kong, 2018.

[3] Y. Verma, "A Complete Understanding of Dense Layers in Neural Networks", *Analytics India Magazine*, 2021. [Online]. Available: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks. [Accessed: 28- Jul- 2022].

[4] Chen, T, and Guestrin, C. "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785-794. 2016.

[5] J. Brownlee, "A Gentle Introduction to Batch Normalization for Deep Neural Networks", *Machine Learning Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/. [Accessed: 28- Jul- 2022].

[6] Keras, "Dropout layer", *Keras.io*. [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/. [Accessed: 28- Jul- 2022].

[7] J. Brownlee, "A Gentle Introduction to the Rectified Linear Unit (ReLU)", *Machine Learning Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. [Accessed: 28- Jul- 2022].

[8] J. Brownlee, "How to Choose Loss Functions When Training Deep Learning Neural Networks", *Machine Learning Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/. [Accessed: 28- Jul- 2022].

[9] Keras, "Keras: the Python deep learning API", *Keras.io*. [Online]. Available: https://keras.io/. [Accessed: 28- Jul- 2022].

[10] Scikit-learn, "sklearn.inspection.permutation_importance", *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html. [Accessed: 31- Jul- 2022].

[11] Srivastava, N, Hinton, G, Krizhevsky, A, Sutskever, I, Salakhutdinov, R, 2014, 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', *Journal of Machine Learning Research*, vol 14, issue 15, pp.1929–1958, accessed 10 April 2022, <https://jmlr.org/papers/v15/srivastava14a.html>

[12] Williams, J, Li, Y, 2008, 'A case study using neural networks algorithms: horse racing predictions in Jamaica', ICAI 2008: International Conference on Artificial Intelligence, pp16-22, accessed 5 March 2022, <https://eprints.usq.edu.au/4300/1/Williams_Li_Publ_version.pdf>

[13] Cheng, T, Lau, M, 2022, 'Predicting Horse Racing Result Using TensorFlow - Term 1 Report', CUHK CSE, pp.1-55, accessed 18 March 2022, <https://www.cse.cuhk.edu.hk/lyu/_media/students/lyu1603_term_1_report.pdf?id=students%3Afyp&cache=cache>

[14] Borowski, P, Chlebus, M, 2021, 'Machine learning in the prediction of flat horse racing results in Poland', Working Papers 2021-13, Faculty of Economic Sciences, University of Warsaw, pp.1-37, accessed 16 March 2022, <https://www.researchgate.net/publication/352778299_MACHINE_LEARNING_IN_THE_PREDICTION_OF_FLAT_HORSE_RACING_RESULTS_IN_POLAND>

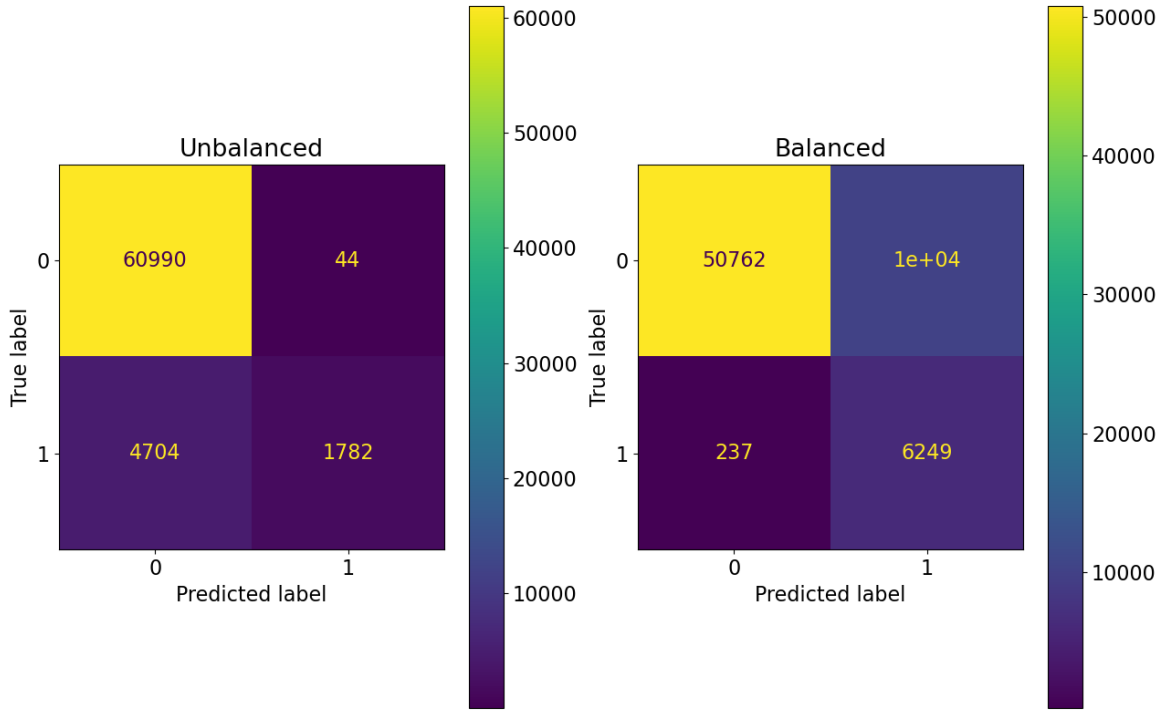
Appendices

Appendix 3.2.1: Correlation between principal components and features

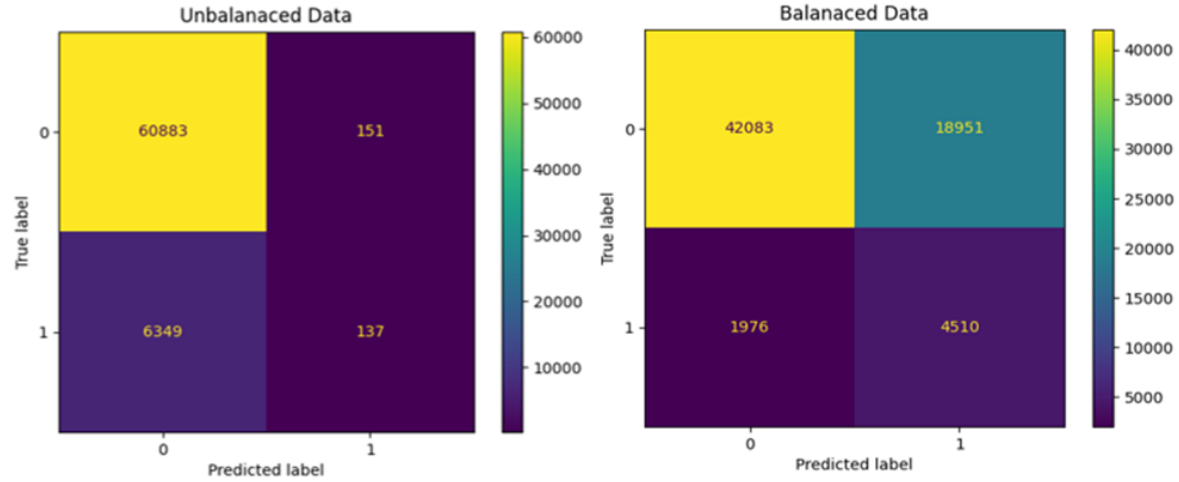
	Trainer 12 Month Horse Earnings_race_chi_sq	gender_colt_race_chi_sq	Jockey This Season Place Strike Rate	Wet Track Runs_race_z_score	gender_other_race_chi_sq	This Distance Places_race_z_score	Trainer This Season Strike Rate_race	Trainer Last Season Horse Earnings	Trainer 12 Months Starts_race_chi_sq	Wet Track ROI	Jockey Last Season Horse Earnings_r	Trainer This Season Place Strike Rate	Trainer This Season Horse Earnings_r	This Track Distance Places_race_z_score	gender_gelding_race_chi_sq	Best Fixed Odds_rank	Best Fixed Odds_race_z_score
PC-1	0.22	0.02	0.09	-0.05	0.00	-0.02	0.00	0.23	0.22	0.02	0.09	0.11	0.22	-0.01	-0.01	-0.07	-0.07
PC-2	0.03	-0.02	0.01	0.21	0.00	0.26	-0.01	0.02	0.03	-0.02	0.02	-0.03	0.03	0.24	-0.01	0.05	-0.01
PC-3	-0.14	0.02	0.23	-0.10	0.00	0.02	0.01	-0.07	-0.15	0.04	0.12	0.21	-0.14	0.04	-0.01	-0.20	-0.22
PC-4	-0.02	0.01	0.06	0.04	0.01	-0.06	-0.03	-0.01	-0.03	-0.02	0.27	-0.16	-0.02	-0.11	0.00	0.15	0.15
PC-5	-0.04	0.01	-0.03	0.12	-0.01	0.07	-0.07	0.02	-0.04	0.00	0.12	0.03	-0.04	0.12	-0.03	-0.18	-0.07
PC-6	0.00	0.03	-0.10	-0.13	0.04	-0.05	-0.03	0.01	-0.01	0.22	0.04	-0.05	0.00	0.06	-0.02	0.11	0.09
PC-7	-0.08	0.05	-0.15	-0.05	0.00	0.06	-0.02	0.03	-0.09	-0.07	0.05	0.16	-0.08	0.00	0.01	0.02	-0.03
PC-8	0.05	-0.06	0.02	-0.03	-0.07	0.00	0.17	-0.05	0.06	-0.01	0.02	-0.01	0.05	0.04	0.01	-0.02	0.01
PC-9	0.00	-0.06	0.02	0.02	0.57	-0.01	0.01	0.00	0.00	-0.02	-0.01	0.01	0.00	-0.01	0.03	0.00	0.00
PC-10	-0.04	0.08	0.06	0.11	-0.03	0.00	-0.26	0.07	-0.06	0.07	-0.07	-0.03	-0.04	-0.13	-0.02	0.03	-0.01
PC-11	0.01	0.31	0.02	-0.20	0.04	-0.06	-0.10	0.01	0.00	-0.09	-0.02	-0.08	0.01	0.22	0.03	0.03	0.02
PC-12	0.01	0.39	-0.03	0.06	0.05	0.12	0.09	-0.02	0.01	0.03	0.00	-0.02	0.01	-0.06	0.03	-0.02	-0.02
PC-13	-0.01	0.22	0.07	0.16	0.01	-0.08	0.11	0.00	-0.01	0.02	-0.02	0.06	-0.01	-0.08	-0.01	-0.02	0.02
PC-14	0.09	-0.01	-0.03	-0.08	0.01	0.14	-0.10	-0.06	0.10	0.00	0.02	-0.17	0.09	0.05	0.04	-0.23	-0.23
PC-15	-0.13	-0.10	0.06	-0.03	-0.01	0.06	-0.04	0.17	-0.13	0.04	-0.02	-0.09	-0.13	0.11	0.00	-0.01	0.02
PC-16	0.01	0.01	-0.35	0.01	0.01	-0.07	-0.01	0.00	0.00	0.03	-0.05	0.03	0.01	-0.06	-0.01	-0.02	-0.05
PC-17	0.01	0.00	0.03	-0.06	0.00	-0.12	-0.01	-0.02	0.02	-0.13	0.01	-0.07	0.01	-0.20	0.01	0.00	-0.04
PC-18	0.01	-0.01	-0.06	0.02	0.00	0.02	0.11	-0.04	0.01	-0.23	-0.02	0.03	0.01	0.18	0.07	0.05	0.14
PC-19	0.03	0.02	0.01	0.02	0.01	-0.01	0.06	-0.06	0.04	0.54	-0.02	0.03	0.03	0.08	-0.01	0.02	0.03
PC-20	0.08	-0.01	0.08	0.00	-0.01	0.02	0.01	-0.10	0.07	-0.02	0.14	0.02	0.08	-0.01	0.01	-0.04	-0.03
PC-21	0.04	-0.02	0.08	-0.09	0.00	0.15	-0.11	-0.03	0.04	-0.09	-0.26	0.11	0.04	0.03	0.05	-0.01	0.03
PC-22	-0.05	-0.02	0.09	-0.09	0.00	0.18	0.27	0.02	-0.04	0.10	-0.12	-0.03	-0.05	-0.07	0.04	0.03	0.17

The above is a snapshot of the correlation matrix between principal components generated and the original features in the data. The cells in green indicate where the absolute value of correlation between the feature and the principal component is over 0.2, which indicates the feature is somewhat correlated with the principal component. This acts similarly to clustering correlated features together. We can see that the first principal component (PC-1) is mainly a linear combination of trainer related features such as “Trainer 12 Month Horse Earnings” and “Trainer This Season Horse Earnings”.

Appendix 4.2.1 - XGBoost confusion matrix (unbalanced vs balanced data)

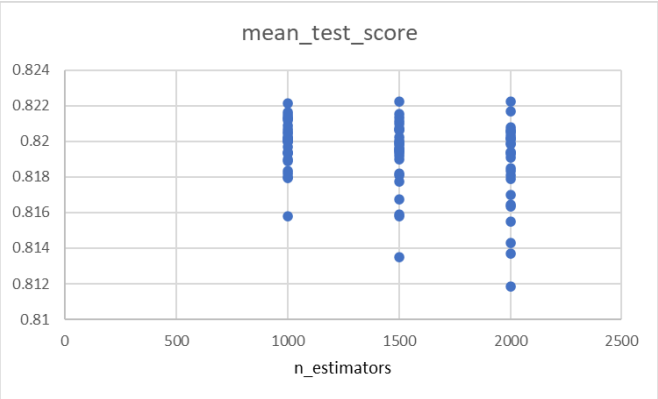


Appendix 4.2.2 - Logistic Regression Confusion Matrix (unbalanced vs balanced data)

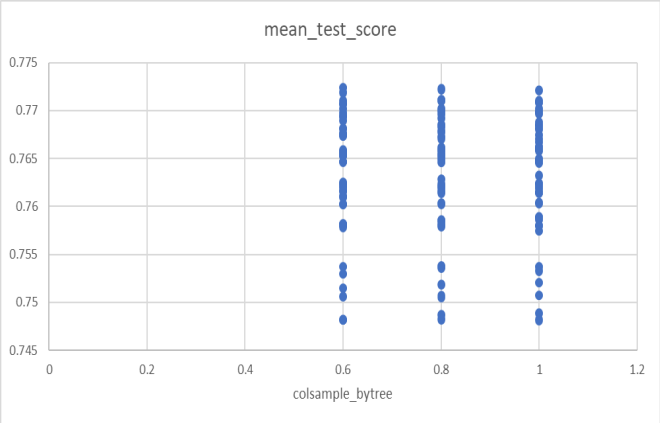


Appendix 4.3.1.1: Mean ROC AUC score for XGBoost Learning Rate

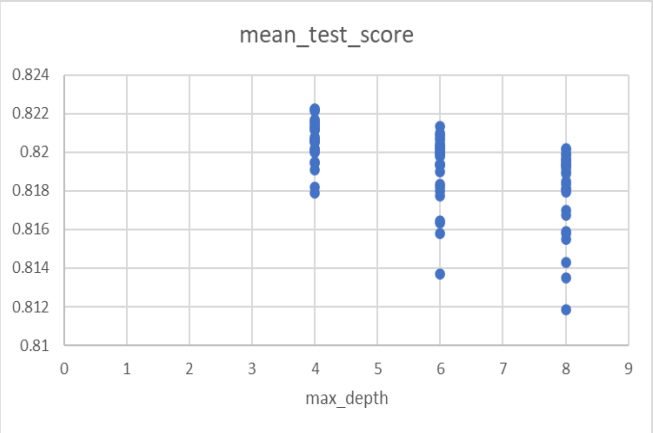
Appendix 4.3.1.2: Mean ROC AUC score for XGBoost N Estimators



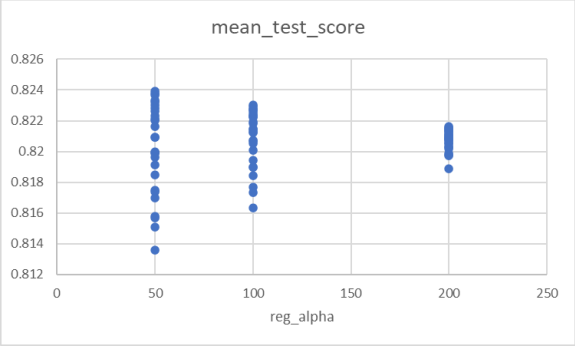
Appendix 4.3.1.3: Mean ROC AUC score for XGBoost Colsample Bytree



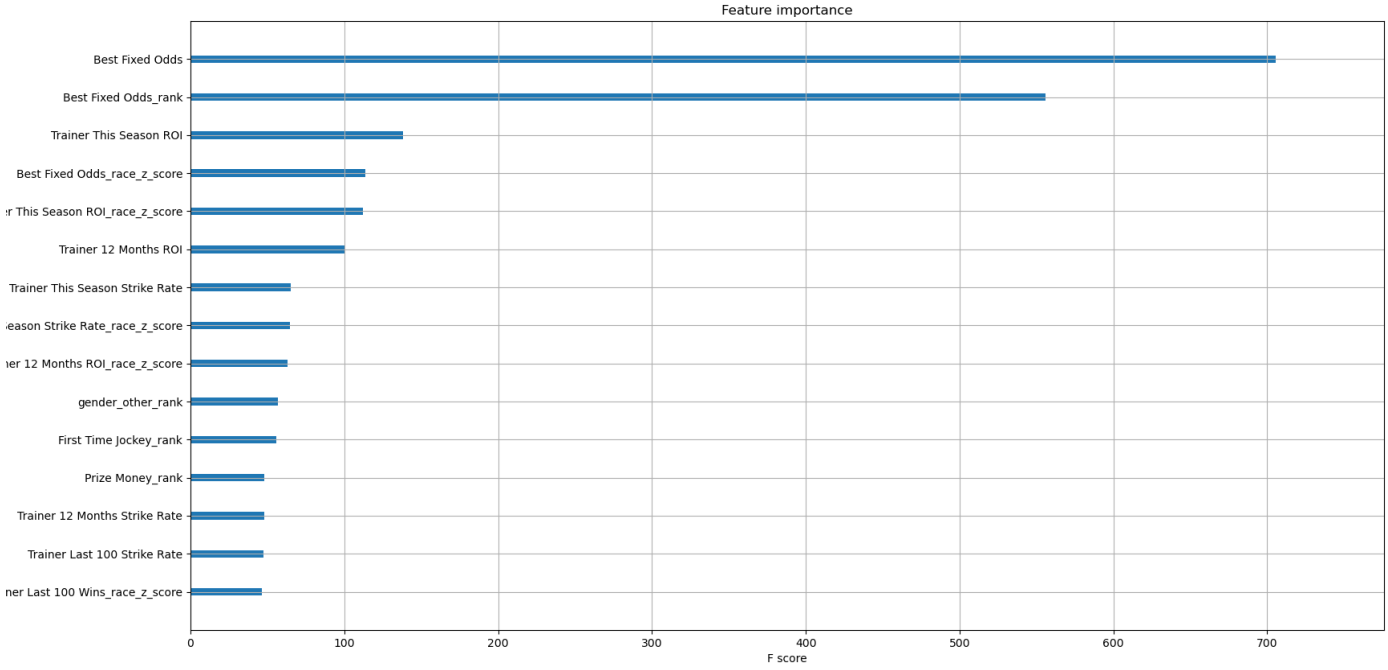
Appendix 4.3.1.1: Mean ROC AUC score for XGBoost Max Depth



Appendix 4.3.1.1: Mean ROC AUC score for XGBoost Reg Alpha

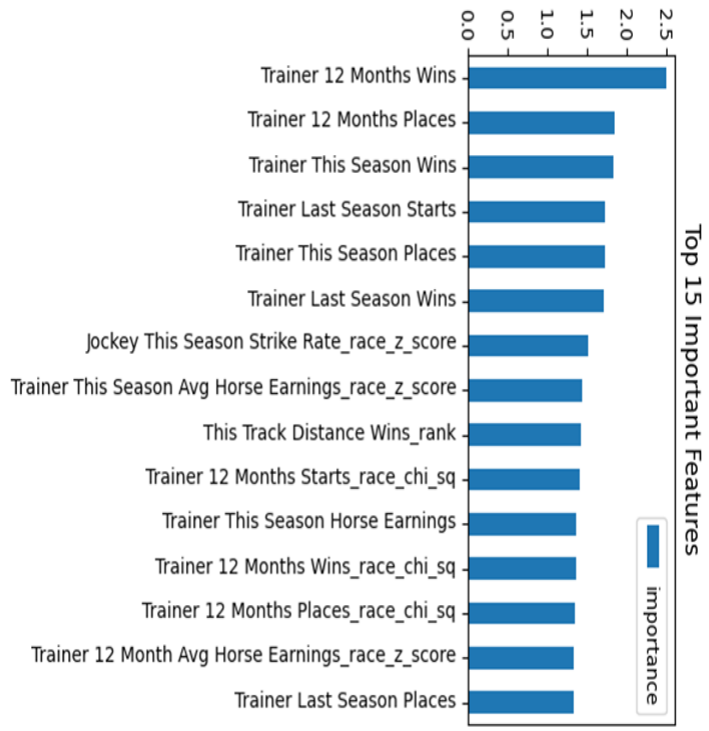


Appendix 4.4.1: XGBoost Feature importance (top 15)

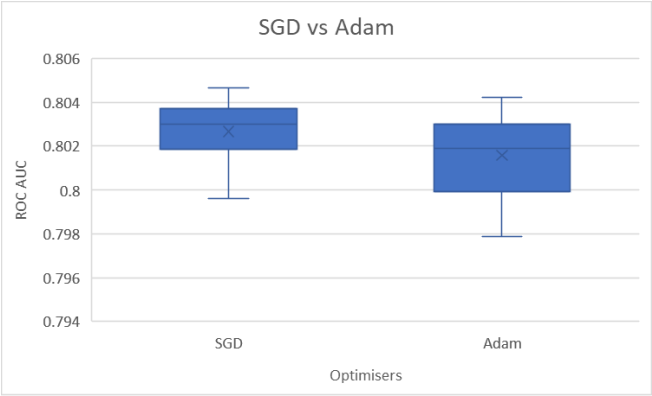


Appendix 4.4.2: Random Forest Feature importance (top 15)

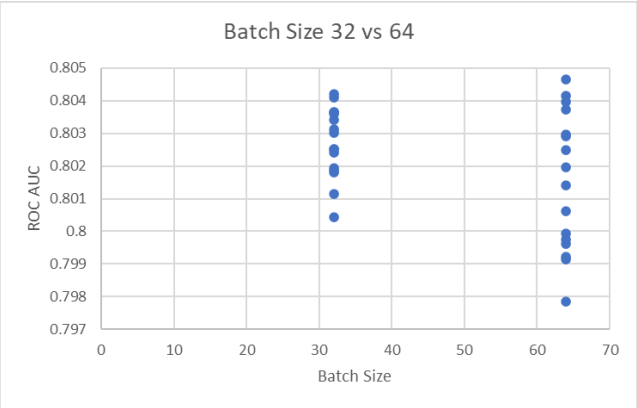
Appendix 4.4.3: Logistic Regression Feature importance (top 15)



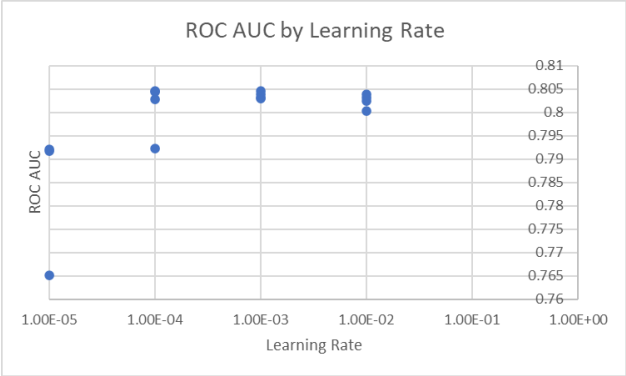
Appendix 4.3.2.1: ROC AUC Score by Neural Network Optimiser



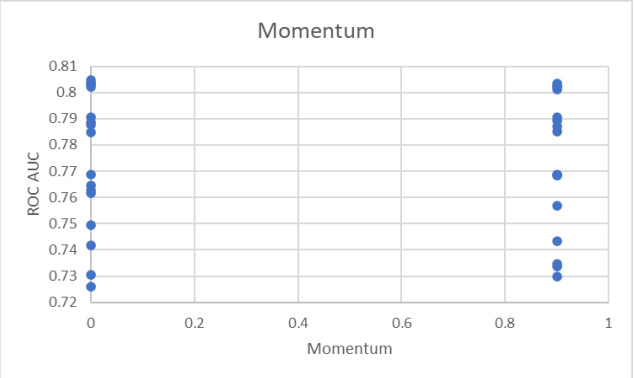
Appendix 4.3.2.2: ROC AUC Score by Neural Network Batch Size



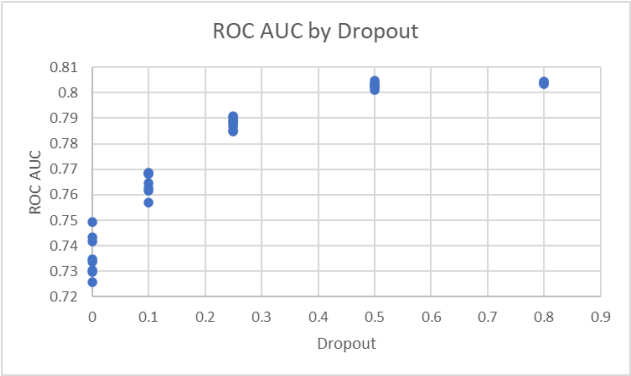
Appendix 4.3.2.3: ROC AUC Score by Neural Network Learning Rate



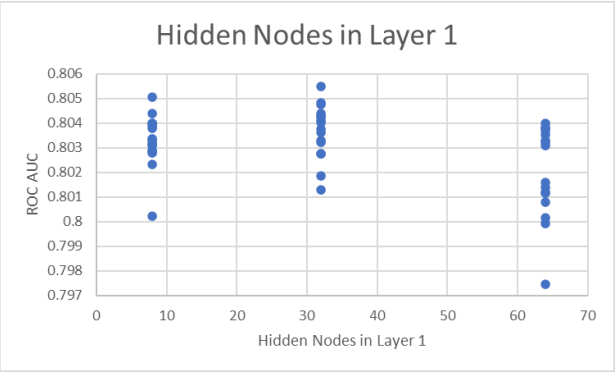
Appendix 4.3.2.4: ROC AUC Score by Neural Network Optimiser Momentum



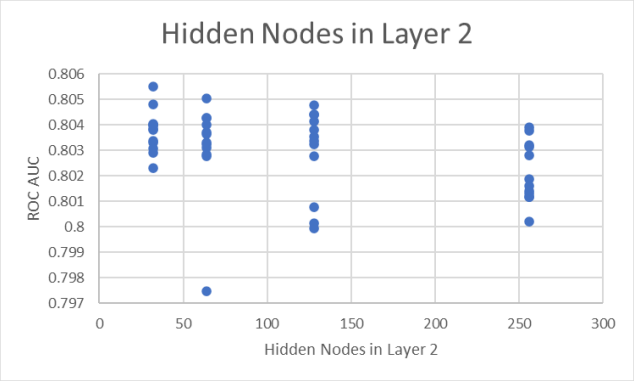
Appendix 4.3.2.5: ROC AUC Score by Neural Network Dropout



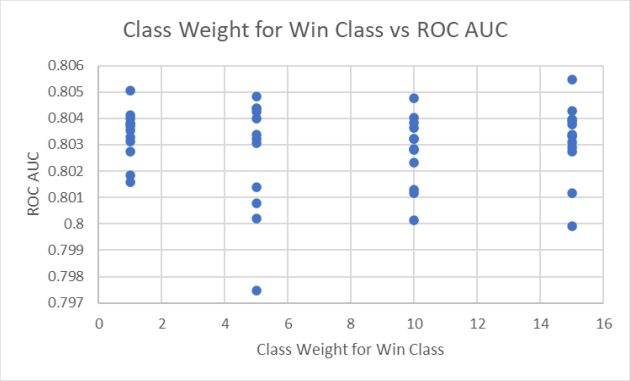
Appendix 4.3.2.6: ROC AUC Score by Hidden Nodes in Neural Network Layer 1



Appendix 4.3.2.7: ROC AUC Score by Hidden Nodes in Neural Network Layer 2



Appendix 4.3.2.8: ROC AUC Score by Neural Network Class Weight for Win Class



Appendix 5.2.1 - ROC for XGboost