**CSCI 3320: Fundamentals of Machine Learning**

**Project: Horse Racing Prediction**

Wang Haoliang (1155112154) & Atif Khurshid (1155085457)

**2. The Dataset and pre-processing**

**2.2 Data pre-processing**

2.2.3 Indices and features for horses, jockeys and trainers

Number of horses: 2155

Number of jockeys: 105

Number of trainers: 93

Note that the number of horses is calculated after removing rows where finishing position is not a number, so some horse are removed from the data. Same for jockeys and trainers. Before removal, the number of horses, jockeys and trainers are 2162, 106 and 95 respectively.

1. **Classification**

Note that for all four classifiers, we tried every combination of the eight features, namely: 'jockey\_ave\_rank', 'trainer\_ave\_rank', 'actual\_weight', 'declared\_horse\_weight', 'draw', 'win\_odds', 'recent\_ave\_rank' and 'race\_distance' to find out which features works best for each of these classifiers respectively, the code for this can be found in ‘find\_features.py’ in folder ‘supplementary\_code’.

What’s more, to find out the optimal depth for the random forest classifier, we tried depth ranging from1 to 20 and find that 8 works best, the code for this can be found in ‘find\_depth.py’ in folder ‘supplementary\_code’.

**3.1 Training Classifiers in Scikit-Learn**

3.1.1 Logistic Regression

The prediction results of Top1, Top3 and Top50percent by the logistic regression classifier can be found at ‘lr\_predictions.csv’ in folder ‘predictions’.

The F1-score for Top1, Top3 and Top50percent predictions by the logistic regrssion classifier are 0.25651, 0.51704 and 0.71984 respectively.

The running time for fitting the logistic classifier to traing data for Top1, Top3 and Top50percent are 0.026498s, 0.025691s and 0.019104s respectively, and the average training time for the logistic classifier is 0.023764s.

3.1.2 Naïve Bayes

Among these three naive bayes classifiers, choosing the BernoulliNB is not reasonable since the features(actual weight, recent average rank etc.) are not bipolar distributed, obviously. Comparing the F1-score of the GaussianNB and MultinomialNB implemented by scikit-learn, we find that the GaussianNB gives a higher score, so the GaussianNB is the best choice.

The prediction results of Top1, Top3 and Top50percent by my own GaussianNB classifier can be found at ‘nb\_predictions.csv’ in folder ‘predictions’.

The F1-score for Top1, Top3 and Top50percent predictions by my own GaussianNB classifier are 0.26520, 0.49279 and 0.71880 respectively.

The F1-score for Top1, Top3 and Top50percent predictions by the scikit-learn GaussianNB classifier are 0.27645, 0.49637 and 0.71880 respectively.

The running time for fitting my own GaussianNB classifier to traing data for Top1, Top3 and Top50percent are 0.003688s, 0.003817s and 0.006730s respectively, and the average training time for my own GaussianNB classifier is 0.004745s.

The running time for fitting the scikit-learn GaussianNB classifier to traing data for Top1, Top3 and Top50percent are 0.003996s, 0.004007s and 0.003709s respectively, and the average training time for the scikit-learn GaussianNB classifier is 0.003904s.

3.1.3 SVM

We have selected the RBF kernel because it can handle both linear and non-linear relationships. Moreover, the number of features is small and RBF performs well for small number of features.

The prediction results of Top1, Top3 and Top50percent by the SVM classifier can be found at ‘svm\_predictions.csv’ in folder ‘predictions’.

The F1-score for Top1, Top3 and Top50percent predictions by the SVM classifier are 0.23194, 0.49634 and 0.71807 respectively.

The running time for fitting the SVM classifier to traing data for Top1, Top3 and Top50percent are 18.032782s, 17.948604s and 17.429695s respectively, and the average training time for the SVM classifier is 17.803694s.

3.1.4 Random Forest

The prediction results of Top1, Top3 and Top50percent by the random forrest classifier can be found at ‘rf\_predictions.csv’ in folder ‘predictions’.

The F1-score for Top1, Top3 and Top50percent predictions by the random forrest classifier are 0.32556, 0.54470 and 0.69737 respectively.

The running time for fitting the random forrest classifier to traing data for Top1, Top3 and Top50percent are 0.120456s, 0.066900s and 068317s respectively, and the average training time for the random forrest classifier is 0.085224s.

**3.3 Training Classifiers in Scikit-Learn**

All of the prediction results can be found in folder ‘predictions’.

**3.4 Writing A Report**

Q1: What are the characteristics of each of the four classifiers?

A1: The Naive Bayes method is very low time complexity, easy to build and particularly useful for very large data sets. SVM is considerably memory efficient due to its own advantage of kernel mapping to high-dimensional feature spaces, but takes much longer time compared to other classifiers as shown above. Both logistic regression and random forest are very fast, there isn't much to distinguish them in terms of run-time. Logistic regression is simple and has low variance and so is less prone to over-fitting. Random forest can be scaled up to be very complex, are are more liable to over-fit.

Q2: Different classification models can be used in different scenarios. How do you choose classification models for different classification problems? Please provide some examples.

A2: The Naive Bayes method assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature and usually works quite well in some real-world situations such as spam filtering and document classification. SVM is very effective in high dimensional spaces but if the number of feature is much greater than the number of samples, SVM is likely to give poor performance. Logistic regression will work better if there's a **single** decision boundary, not necessarily parallel to the axis. Random forest can be applied to situations where there's not just one underlying decision boundary, but many, and will work best if the class labels roughly lie in hyper-rectangular regions.

Q3: How do the cross-validation techniques help in avoiding overfitting?

A3: Actually cross-validation cannot avoid overfitting, it just reduces it. Splitting the data into a training and testing set can also reduce overfitting but a static split doesn’t use the data efficiently. Cross-validation has the advantage of static splitting mentioned above and also uses the data as efficiently as possible at the same time. In this project, we choose k-fold cross-validation where k=10 and selects the split giving the lowest error, which has a better chance representing the real data distribution, to train our classifier.

Q4: In addition to the Precision-Recall metric, there are many other metrics can be derived according to the confusion matrix, e.g., the true negative rate TNR=TN/TN+FP, the negative predictive value NPV= TN/TN+FN and so forth. How do you choose evaluation metrics for imbalanced datasets according to the class distribution? Please give your understanding and provide some examples.

A4: The choice of evaluation metrics depends on the goal of our task. For example, in the scenario of an access control system, we hope the precision tobe high, that is, we hope that the true positives take up a large proportion among the positive predictions but the recall is not that impoartant. As another example, precision is not that important when we are doing advertising, on the other hand, recall is what really matters since we want to cover as many potential customers as possible.

**4. Regression**

**4.1 Training Regression Model in Scikit-Learn**

4.1.1 Support Vector Regression Model(SVR)

Kernel: We have selected the RBF kernel because it can handle both linear and non-linear relationships. Moreover, the number of features is small and RBF performs well for small number of features.

Hyperparameters: Epsilon is the width of SVM's soft boundary whereas C is the regularization parameter that controls the trade-off between epsilon and classification accuracy. Hence, a larger C and smaller epsilon result in a complex model which may overfit whereas smaller parameters mean that the model is more general with a large margin of separation between classes. We have selected epsilon=0.1 and C = 5000 because these parameters result in the lowest RMSE which means that they provide a good fit of the training data while ensuring the generality of the model.

4.1.2 Gradient Boosting Regression Tree Model(GBRT)

Loss: We have selected ls (squared distance) because it is similar to lad (absolute difference) and huber (ls + lad) and has no extra parameters unlike quantile. Furthermore, ls results in a model with the best RMSE result.

Hyperparameters: learning\_rate is a parameter of gradient descent and it determines the effect of new trees on the result of regression. n\_estimaters is the number of decision trees which are combined to form a final regression model and it is directly proportional to the complexity of the model. max\_depth is the maximum depth of each decision tree and a small max\_depth prevents overfitting. After a grid search, we assigned the values of learning\_rate=0.05, n\_estimaters=120, max\_depth=5 because these parameters result in the model with the least RMSE.

**4.2 Predicting on Test Data**

SVR: kernel='rbf', C = 5000, epsilon=0.1, gamma= 0.000001

(svr\_model, 1.567, 0.200, 0.473, 4.763)

GBRT: loss='ls', learning\_rate=0.05, n\_estimators=120, max\_depth=5, random\_state=42

(gbrt\_model, 1.550, 0.229, 0.517, 4.390)

SVR: Before normalisation: 1.567

After normalisation: 1.564

It appears that RMSE has decreased slightly but more importantly, the scaled\_svr\_model is far less complex than the unscaled model since it has a smaller C value and a larger gamma value. Hence, normalization had a positive impact on SVR because it is sensitive to differences in the ranges of values of different features.

GBRT: Before normalisation: 1.550

After normalisation: 1.549

RMSE is almost unchanged. Hence, normalization had no effect on the result of decision trees because they are insensitive to linear transformations.

1. **Betting Strategy**
   1. **Basic Part**

Classification:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Money won | Won | Lost | Win loss ratio |
| lr\_model | 494 | 139 | 340 | 0.41 |
| Normalized lr\_model | 494 | 139 | 340 | 0.41 |
| nb\_model | 317 | 98 | 271 | 0.36 |
| Normalized nb\_model | 290 | 92 | 262 | 0.35 |
| svm\_model | 468 | 135 | 342 | 0.39 |
| Normalized svm\_model | 481 | 137 | 341 | 0.40 |
| rf\_model | 481 | 137 | 341 | 0.40 |
| Normalized rf\_model | 481 | 137 | 341 | 0.40 |

Regression:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Money won | Won | Lost | Win loss ratio |
| svr\_model | 172 | 93 | 386 | 0.24 |
| Normalized svr\_model | 179 | 94 | 385 | 0.24 |
| gbrt\_model | 277 | 108 | 371 | 0.29 |
| Normalized gbrt\_model | 277 | 108 | 371 | 0.29 |

Winning Horse:

The way we decide on the winning horse is as follows:

In each race, the trained model makes its predictions on ‘HorseWin’, ‘HorseRankTop3’ and ‘Horse RankTop50Percent’(if the model is a classifier, then the prediction result is simply 0 or 1. See \*\_predictions.csv in folder ‘predictions’; for regression models, we can also turn the regression result on test data, which is a real number, into 0 or 1 labels. See function ‘Time\_to\_label’ in ‘betting.py’). If we get all ones in these three columns for a horse, then we say the horse is a ‘winning candidate’ in this race. The problem is that even by doing so, there still might be multiple winning candidates in one race, then how do we decide which particular horse we should bet on?

The method we come up with is that for each winning candidate in a race, we find out the reciprocal of its corresponding winning odds and find out the horse with the maximum value and this is our winning horse. The logic behind this is that the greater the winning odds, the less likely the horse would win. So after taking the reciprocal of the winning odds, we can decide the horse which is most likely to win by finding out the maximum value.

**5.2 Bonus Part**

Betting Strategy:

As the results above show, none of the algorithms is accurate enough in its prediction of the winning horse. Therefore, we consider a combination of the results of the 4 classification models and the GBRT regression model to predict the winning horse. We have already discussed how to decide on a unique winning horse in each race, as mentioned above.

More formally, we create a vector, , the final prediction for each model, where an element of is 1 only if the predicted top1, top3 and top50percent labels of the model are 1:

Then, we calculate a weighted sum of the predictions of each model for each horse and the horse with the highest sum is the predicted winner.



The weights are defined according to the relative top1 precision score of each model. More specifically,



The reason we don’t use the top1 F1-score as the weights for each model is that the F1-score is a harmonic mean of precision and recall but obviously the precision score is more important in this case, the code for calculating these weights can be found in ‘find\_weight.py’ in folder ‘supplementary\_code’.

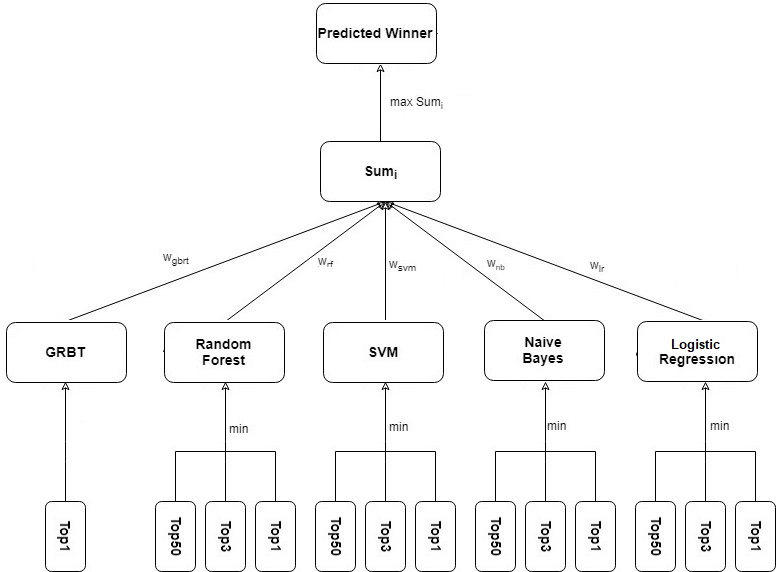
There is one more trick we use: we set a threshold for the score

.



If the score is below the threshold, then we do not bet on any horse. Obviously, if we set the threshold to be high, we tend to be more conservative or more radical is the threshold is low.

The following is a graphical representation of the selection process:



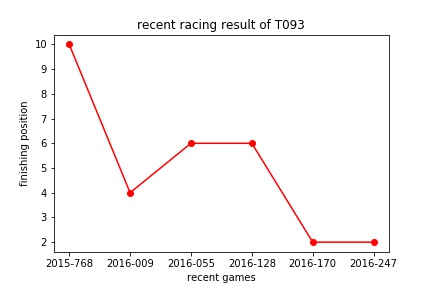
For our own betting strategy, the results are shown below:

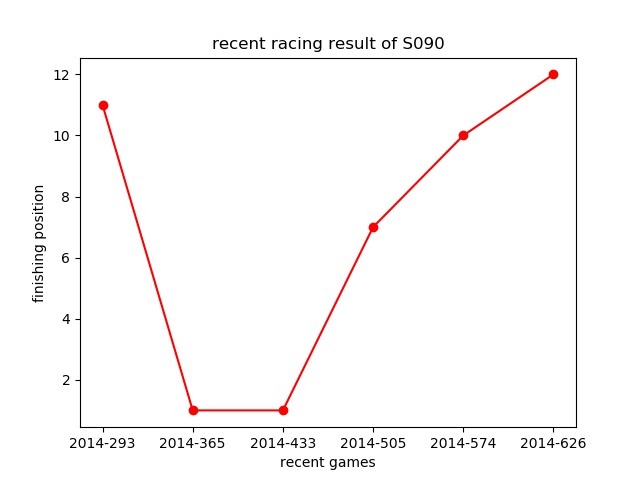
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Money won | Won | Lost | Win loss ratio |
| lr\_model | 509 | 139 | 325 | 0.43 |

**6. Visualization**

**6.1 Line Chart of Recent Racing Result**

The racing result of T093 and S090 is shown below. From the plot we can observe that the racing result for different horses varies a lot and do not have a clear tendency.

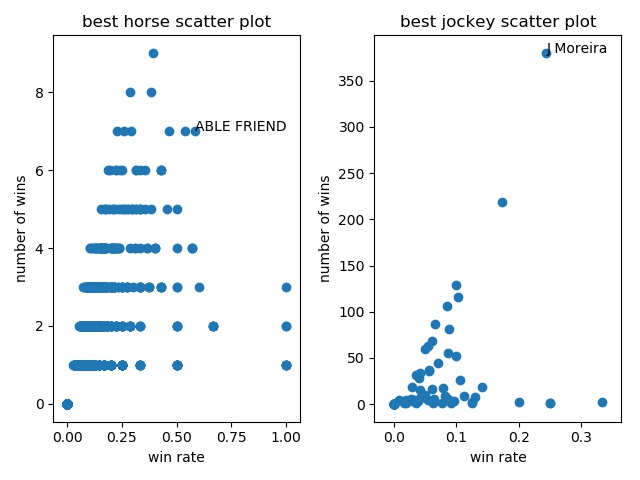




**6.2 Scatter Plot of Win Rate and Number of Wins**

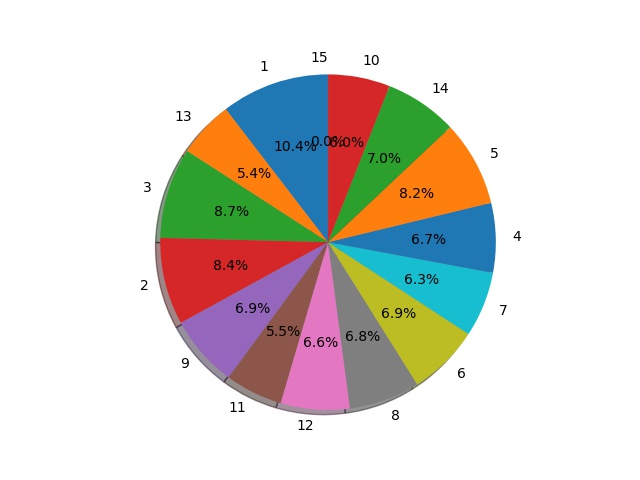
The ‘best’ horse and ‘best’ jockey are ABLE FRIEND and J Moreira respectively. The reason for this is that they are on the upper-right corner of the ‘Win\_Rate’ vs ‘Number\_of\_Wins’ scatter plot which means that they have both high win rate and a large number of wins.

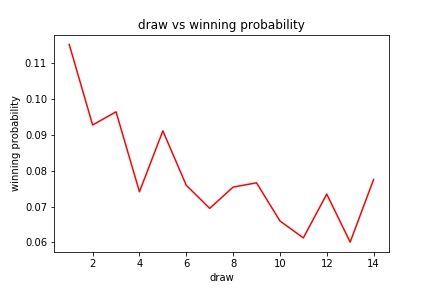
You can change this threshold by modifying ‘alpha\_number\_of\_wins\_horse’, ‘alpha\_win\_rate\_horse’, ‘alpha\_number\_of\_wins\_jockey’ and ‘alpha\_win\_rate\_jockey’ in ‘scatter\_ploy.py’ and see how the annotations change correspondingly.



**6.3 Pie Chart of the Draw Bias Effect**

As we can see from the pie chart, the first lane has a significantly better chance to win compared to other lanes, but whether the inside lane would hold an edge over the field compared to the other lanes is not so obvious from the pie chart. So we plot a line chart where the x-axis is the draw number and the y-axis is the probability of winning, and from this chart we can observe that the draw does have some influence on the chance of winning.





**6.4 Bar Chart of the Feature Importance**

As we can observe from the bar chart, the ‘win\_odds’is the most important feature. The ranking of the features and their relative importance is shown below.

Feature ranking:

1. win\_odds (0.664118)

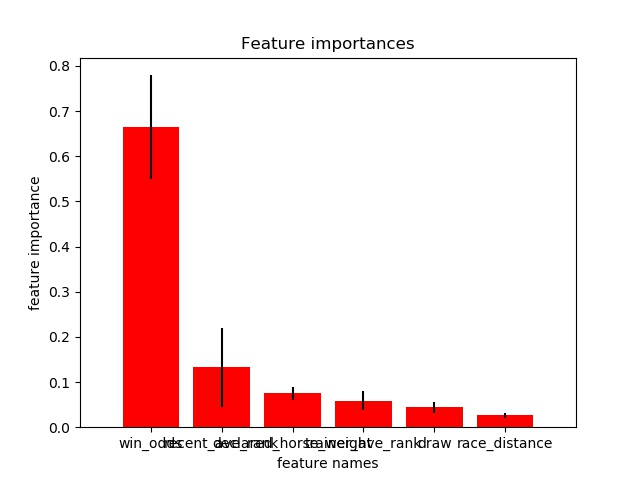
2. recent\_ave\_rank (0.132251)

3. declared\_horse\_weight (0.074660)

4. trainer\_ave\_rank (0.059096)

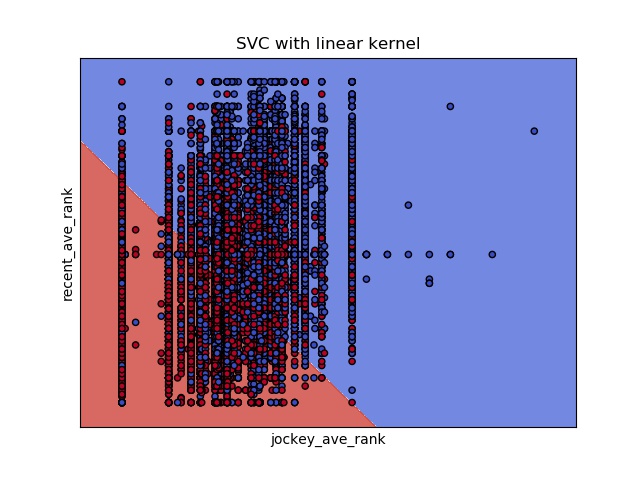
5. draw (0.043819)

6. race\_distance (0.026056)



**6.5 Visualize SVM**

As we can observe from the plot, the boundary between the red points and the blue points is not very clear on testing data.



At first we are suspicious whether we get it wrong so we plot the same figure on the training data(see below) which seems perfect.

