CSCI 3320: Fundamental of Machine Learning **2017-2018 Term 2**

Project: Horse Racing Prediction

Instructor: Prof. John C.S. Lui Due: 17:00 on Friday, April 27, 2018

1 Introduction

Gambling on the horse racing results is a breathtaking entertainment in Hong Kong. According to Hong Kong Jockey Club, the total turnover is 117,456 million HKD for the horse races in the season of 2016/17, which means on average, one in Hong Kong shares a turnover of around 15,000 HKD[1].

Figure 1: Horse Racing in Shatin

There are various ways to bet[2]. For example, if you bet on which horse will "win", then you can get the reward only if the chosen horse ranks the first to finish. In general, to win the reward from the horse betting, you need to have a good prediction on the racing results. The results of horse races don't come from nowhere. They are related to different factors such as the horses and jockeys, the track and distance, etc. Experienced players may have their "formulas" to determine which horses to bet on. It is reported that some Professor in the Statistics department of CUHK has earned tens of millions of HKD by his "formula"[3]. We wonder whether we can "predict" the results of horse races and build our "formula" to win the rewards. In this project, we will use data from past races and try different machine learning techniques to make predictions.

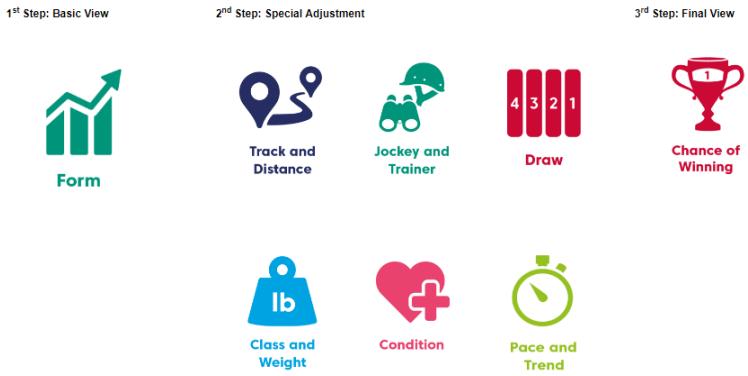

Figure 2: Key Steps to pick the right horses (from tutorials of Hong Kong Jockey Club)

[1] suppose the population in HK is 7.347 million.
[2] http://entertainment.hkjc.com/entertainment/english/learn-racing/racing-academy.aspx
[3] http://orientaldaily.on.cc/cnt/news/20160227/00174_001.html

2 The Dataset and pre-processing (20 pts)

2.1 Dataset Descriptions

The dataset is from kaggle (www.kaggle.com/lantanacamara/hong-kong-horse-racing) which is extracted from the website of the Hong Kong Jockey Club[4][5]. We prepare the dataset for you in `prj.zip`. This dataset contains the results of 2,367 horse races. In the dataset, there are 2,162 horses, 106 jockeys and 95 trainers. Averagely, every horse has participated in around 14 races. For detailed descriptions of the dataset, we refer you to the kaggle page mentioned above. You are also recommended to read related discussions on kaggle. The file `race-result-race.csv` describes the races. Each entry in another file `race-result-horse.csv` corresponds to one horse in a race. Now, we list and explain the attributes in the file `race-result-horse.csv`.

- finishing_position: the rank of the horse. (E.g. the horse with finishing_position 1 is the first to finish)
- horse_number: the number for the horse in the specific race. (Note that the same horse may have different numbers in different races)
- horse_name: English name of the horse.
- horse_id: ID of the horse. (The ID for a horse is unique in all the races)
- jockey: the one who rides the horse in the race. (A jockey can ride different horses in the races)
- trainer: the one who trains the horse. (Multiple horses from a trainer can appear in the same race)
- actual_weight: the extra weight that the horse carries in the race. (The horses with better performances in the previous races should carry heavier extra weights, to make the race more competitive[6])
- declared_horse_weight: the weight of the horse on date of the race.
- draw: the position of the horse at the starting point[7]. The inner positions are usually advantageous and correspond to smaller draw numbers.
- length_behind_winner: the length behind the winner at the finish line. The unit is the "horse length".
- running_position_1: the rank of the horse at the first timing point.
- running_position_2: the rank of the horse at the second timing point.
  ...
- running_position_i: the rank of the horse at the i^{th} timing point. (The running position will be "NA" if the total distance of the race is short and the horses don't go across the particular timing point)
- finish_time: the total time from the starting point to the finish line. The unit is "second".
- win_odds: the ratio between the reward you will get and the money you bet, supposing that you will win. The odds are usually determined automatically by the total money bet on each horses[8].
- race_id: the ID of the race for this entry. The race_id is consistent in the two data files.

[4] http://racing.hkjc.com/racing/Info/meeting/racecard/
[5] https://market.mashape.com/hkdatahose/hong-kong-horse-race-data-hose
[6] http://www.hkjc.com/english/racinginfo/handicap_policy.asp
[7] http://racing.hkjc.com/racing/info/meeting/Draw/English/Local
[8] https://special.hkjc.com/racing/info/en/betting/guide_qualifications_pari.asp

2.2 Data pre-processing

From the raw data that might be messy, we need to extract useful features in preparation for the prediction tasks later. Important features include the recent performances of the horse, the jockey and trainer the of horse, the draw of the horse, the extra weight added to the horse, the distance of the race, to mention a few. We are going to prepare these features. Write your scripts in `preprocess.py` to complete the following tasks. Our final aim is to get the training data which contains all the races with `race_id` no more than "2016-327" (first 80%), and the testing data which contains the rest of races (last 20%) with necessary features.

2.2.1 Loading the data with pandas and cleaning the data (5 pts)

For data pre-processing, `pandas` is a great Python package. To represent tabular data, pandas uses a custom data structure called "dataframe". A dataframe is a highly efficient, 2-dimensional data structure that provides a suite of methods and attributes to quickly explore, analyze, and visualize data. The dataframe is similar to the NumPy 2D array but adds support for many features that help you work with tabular data. To learn how to explore the raw data with `pandas`, please read the document of `read_csv()` in pandas official website. Pandas provides many statistic functions, such as `mean()`, `std`. We can also use `df.info()` to show us some basic knowledge of raw data, and `df.describe()` to give us an overview of the raw data.

- Read the data from `race-result-horse.csv` into a dataframe by `read_csv()`, and drop the rows where the "finish_position" is not a number (e.g. WV-A, WV). Note that if there are accidents of a race, special codes are provided in this column[9].

2.2.2 Recent performances of the horses (5 pts)

One of the most important information needed to predict the racing results is the recent performance of a horse. Here we hold the belief that whether a horse is in a good form should be consistent in a short period. Hong Kong Jockey Club announces the ranks for each horse in its recent 6 runs[10] for the references of the gamblers. Unfortunately, our raw data doesn't directly contain such information. Therefore, we need to extract the recent ranks of the horses for each race. Note that the entries in `race-result-horse.csv` are already sorted by the time (or race_id). For each data entry, we have a horse in focus, and you are required to find the horse's ranks in its latest 6 runs before the race.

- Add a column named `recent_6_runs` to the dataframe, which records the recent ranks of the horse in each entry. The ranks are separated by "/", and a record is like `1/2/6/3/4/7`.

Note: if the total past runs of the horse is less than 6 in our data, just record all the ranks of these past runs. For example, if a horse have participated in 3 runs before a race and ranked 3, 5, 13 respectively, then the record of `recent_6_runs` should be `3/5/13`.

- Add a column named `recent_ave_rank` for each entry to the dataframe, which records the average rank of the recent 6 runs of a horse. If there are less than 6 past runs, take the average of all the past runs. For example, the horse with past ranks 3,5,13 has a average rank (3+5+13)/3=7. If there are no previous runs, set the `recent_ave_rank` to be a prior value 7.

2.2.3 Indices and features for horses, jockeys and trainers (5 pts)

In the raw data, the horses/jockeys/trainers are identified by their name or ID. Giving indices to these entities will allow us to represent them in the mathematical world. Also, we need to extract features for jockeys and trainers. In this part, you are required to

- Give a unique index to each horses, where "which horse has which index" is not restricted. Similarly, a unique index should be assigned to each jockey, and a unique index should be assigned to each trainer.

✍ Write down the number of horses, the number of jockeys, and the number of trainers in `prjreport.pdf`.

[9] http://www.hkjc.com/Chinese/include/special_race_index.htm
[10] http://racing.hkjc.com/racing/Info/meeting/RaceCard/

- Add a column named `jockey_ave_rank` to the dataframe that records the average rank of the jockey in the training data. Similarly, add a column named `trainer_ave_rank` that records the average rank of the trainer in the training data. Note that if a jockey or a trainer doesn't appear in the training data, set the average rank of the jockey to be 7.

2.2.4 Distances of the races (5 pts)

Another important feature is the distance of the race. The distance could be 1000, 1200, 1400, 1600, 1800, 2000, 2400, etc. Some horses are good at short-distance races, while some are good at long-distance races. The race distance information is contained in `race-result-race.csv`.

- Read the distance information in `race-result-race.csv` and add a column to the dataframe `race_distance` for each entry in `race-result-horse.csv`.

2.2.5 Preparing the training and testing dataset

The dataset is divided into the training and the testing set. Recall that the training set contains all the races with `race_id` no more than "2016-327" (first 80%), and the testing set contains the rest of races (last 20%).

- Split the dataframe after all the above pre-processing into two parts, and save them to files. The training data should be saved as `training.csv` and the testing data be saved as `testing.csv`.

3 Classification (30 pts)

Now with the training data, you can train classifiers you have learned in this course to make predictions on the racing results, e.g., will a horse win the race or rank top 3 in the race? Here you are required to build four classifiers: logistic regression classifier, Naïve Bayes classifier, SVM classifier and random forest classifier. Note that you cannot use features which indicate the result of the current race, e.g., finishing_position, finish_time, length_behind_winner, running_position_1,...,running_position_i, to make predictions. But you can utilize these features of historical races in the training set.

3.1 Training Classifiers in Scikit-Learn

3.1.1 Logistic Regression (5 pts[11])

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. You can find the detailed introduction to logistic regression in the User Guide of scikit-learn [1].

- Add scripts to the file `classification.py` to train a logistic regression model `lr_model` with the training data. Please name your logistic regression model with the exact name `lr_model`.

- ✍ Report your prediction result, prediction evaluation and running time in `prjreport.pdf`.

3.1.2 Naïve Bayes (7 pts)

Naïve Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. You can find the details of Naïve Bayes method in the User Guide of scikit-learn [2]. To train a Naïve Bayes classifier, you need to the following.

- ✍ Choose a Naïve Bayes classifier from `GaussianNB`, `MultinomialNB`, and `BernoulliNB` for the problem, and state your reason to choose the specific classifier in the project report `prjreport.pdf`.

[11] Your score for each classifier highly depends on your prediction performance evaluation result, which includes the precision and recall measures. This scoring rule also holds for the other three classifiers.

- Add scripts to the file `classification.py` to train a selected Naive Bayes classifier `nb_model` with the training data. Please name your Naïve Bayes classifier with the exact name `nb_model`.
- Implement the selected Naive Bayes classifier and complete the class `NaiveBayes` in the file `naive_bayes.py`. After your implementation, the following function calls should be valid (in the same folder).

```
import NaiveBayes from naive_bayes
......
clf = NaiveBayes()
clf = clf.fit(X, y)
y_predict = clf.predict(X)
```

- Compare your implementation with the Naïve Bayes classifier implemented in scikit-learn in terms of the running time and prediction performances. Report your prediction results, prediction evaluations and running time in `prjreport.pdf`.

3.1.3 SVM (5 pts)

Support vector machines (SVMs) are typical classification methods. You can find the details of SVM in the User Guide of scikit-learn [3]. To train an SVM classification model, you are recommended to use the `SVC` classifier in the scikit-learn module `sklearn.svm`.

- `SVC` accepts different kernel functions. Kernel functions could be one of `'linear'`, `'poly'`, `'rbf'`, `'sigmoid'`, `'precomputed'` or a callable function defined by yourself. Please choose a kernel function and state your reasons in the `prjreport.pdf`.
- Add scripts to the file `classification.py` to train an SVM classifier `svm_model` with the training data. Please name your SVM classifier with the exact name `svm_model`. (1 pt)
- Report your prediction result, prediction evaluation and running time in `prjreport.pdf`.

3.1.4 Random Forest (5 pts)

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. You can find the details of random forest in the User Guide of scikit-learn [4]. To train a random forest classifier, you are recommended to use the `RandomForestClassifier` classifier in the scikit-learn module `sklearn.ensemble`.

- Add scripts to the file `classification.py` to train a random forest classification model `rf_model` with the training data. Please name your random forest classifier with the exact name `rf_model`.
- Report your prediction result, prediction evaluation and running time in `prjreport.pdf`.

3.1.5 Tips in Training Classification Models

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is so-called overfitting. To avoid overfitting, you are recommended to use the cross validation techniques in scikit-learn. Such techniques can also help you tune the parameters of your classifiers. For information about cross validation, you can read the documentation of scikit-learn[12]. Generally, you can get more accurate models if you use the cross validation techniques to tune your classifiers. You will also get additional marks if you use cross validation techniques in training your classifiers.

[12] http://scikit-learn.org/stable/modules/cross_validation.html

3.2 Getting Predictions

Next, you are required to obtain the predictions on the test data, which you have obtained in the data preprocessing step, using your four trained classifiers.

- Add scripts to the file `classification.py` to get and save prediction results of different classification models (You need read the test data by yourself. Hint: Use similar method as used in obtaining the training data.). Please save the predictions of `lr_model` in the file `lr_predictions.csv`, the predictions of `nb_model` in the file `nb_predictions.csv`, the predictions of `svm_model` in the file `svm_predictions.csv` and the predictions of `rf_model` in the file `rf_predictions.csv`. Note all these prediction files should be in the folder `predictions`. The information that you are required to predict is listed as follows:
  - HorseWin: If the horse was in the first finish speeds, '1' else '0'.
  - HorseRankTop3: If the horse was in the top 3 finish speeds, '1' else '0'.
  - HorseRankTop50Percent: If the horse was in the top 50 percent finish speeds, '1' else '0'.

  The format of prediction results in these `*_predictions.csv` files are described as follows.

```
RaceID, HorseID, HorseWin, HorseRankTop3, HorseRankTop50Percent
2014-001, K019, 1, 1, 1
2014-001, S070, 0, 1, 1
2014-001, P072, 0, 1, 1
...
```

In particular, the first line is `RaceID,HorseID,HorseWin,HorseRankTop3,HorseRankTop50Percent` and the following each line is printed with the format string `%s,%s,%d,%d,%d\n`. (Only a single comma, without any space is in the middle of two `%s` or `%d`.) Note that the comma `,` splits the line into two columns. Each string in the 1st column is the race id and each string in the 2nd column is the horse id. Each number in the 3rd column is the indicator of whether the horse win the race (1) or not (0). Each number in the 4th column is the indicator of whether the horse rank top 3 in the race (1) or not (0). Each number in the 5th column is the indicator of whether the horse rank top 50% in the race (1) or not (0).

Your prediction results should be **reproducible**. That is, your prediction results should be the same each time when you run your scripts. Your marks would be deducted if we find your prediction results and the running results of your scripts are not consistent. Hint: set the parameter `random_state` to be a fixed value in all the functions that provide this parameter.

3.3 Evaluation of Predictions

The evaluation criteria is a key factor in assessing the classification performance and guiding the classifier modelling. For classification problems with imbalanced datasets, i.e., the examples of one class (e.g., HorseWin =0) significantly outnumber the examples of the other one (e.g., HorseWin=1), traditional metric of model performance evaluation - accuracy often provides false indication of model correctness. Many work[13] focuses on the evaluation in imbalanced domains. In a two-class problem, the confusion matrix (shown in Table 1) records the results of correctly and incorrectly recognized examples of each class.

Table 1: Confusion matrix for a two-class problem

	Positive prediction	Negative prediction
Positive class	True Positive (TP)	False Negative (FN)
Negative class	False Positive (FP)	True Negative (TN)

In imbalanced domains, the evaluation of the classifiers' performance must be carried out using specific metrics in order to take the class distribution into account. We can obtain many metrics[14] from Table

[13] http://sci2s.ugr.es/imbalanced
[14] https://en.wikipedia.org/wiki/Precision_and_recall

1 to measure the classification performance of both, positive and negative, classes independently. Here we consider Precision-Recall, which is a useful measure of success of prediction when the classes are very imbalanced, as our evaluation metric.

- **Recall**: $R=\frac{TP}{TP+FN}$ is the percentage of positive instances correctly classified.
- **Precision**: $P=\frac{TP}{TP+FP}$ is the probability that one classified positive instance is classified correctly.

We will compare your results with our ground truth data and evaluate your classification performance according to the Precision-Recall metric. The better performance your classifiers have, the higher score you can obtain.

✍ As we mentioned in section 3.1, you also need to report your prediction result, prediction evaluation result and running time in `prjreport.pdf`.

3.4 Writing A Report (8 pts)

After you obtain all the results, you are required to write a brief report in the file `prjreport.pdf` to answer the following questions. You can write freely as long as you answer the questions clearly since there is no strict format requirement for the report.

✍ **Q**: What are the characteristics of each of the four classifiers? (2 pts)

✍ **Q**: Different classification models can be used in different scenarios. How do you choose classification models for different classification problems? Please provide some examples. (2 pts)

✍ **Q**: How do the cross validation techniques help in avoiding overfitting? (2 pts)

✍ **Q**: 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 $TNR=\frac{TN}{TN+FP}$, the negative predictive value $NPV=\frac{TN}{TN+FN}$ and so fourth. How do you choose evaluation metrics for imbalanced datasets according to the class distribution? Please give your understanding and provide some examples. (2 pts)

4 Regression (30 pts)

After finishing classification tasks, another prediction we are interested in is a more precise prediction on the rank of each horse in a specific race. Direct prediction on the rank is difficult because rank itself is an ordered categorical variable. Instead, one simple solution is to regress on `finish_time` for each horse. Therefore, in this section, you are required to build a regresion model on the `finish_time` of each horse.

Note that in this section, you are required to use 8 features: 'actual_weight', 'declared_horse_weight', 'draw', 'win_odds', 'jockey_ave_rank', 'trainer_ave_rank', 'recent_ave_rank', 'race_distance'.

4.1 Training Regression Model in Scikit-Learn

In this part, we are building regression model to train and predict `finish_time` based on Support Vector Regression Model and Gradient Boosting Regression Tree Model.

4.1.1 Support Vector Regression Model(SVR) (10 pts)

Support Vector Regression Model is a machine learning technique which inherits the main idea(maximal margin) of SVM to do regression tasks. You can go through Scikit-Learn Documentation [5] and Wiki [6] for reference. In this part, you are recommended to write your model with `SVR` regressor in the scikit-learn module `scikit-learn.svm`.

✍ Parameter tunning/selection is required. First, `SVR` accepts different kernel functions. They could be one of linear, poly, rbf, sigmoid, precomputed, select one of them and state your reason in `prjreport.pdf`. Second, `epsilon` and `C` are two critical parameters.Please state what role do they play in the model, what value do you assign and why do you select these values in `prjreport.pdf`

- Add scripts to the file `regression.py` to train an SVR model `svr_model` using scikit-learn function with the training data. Please name your SVR model with the exact name `svr_model`.

4.1.2 Gradient Boosting Regression Tree Model(GBRT) (10 pts)

Gradient Boosting Regression Tree Model(perhaps one of the most popular model in industry) is a generalization of boosting technique to arbitrary differentiable loss functions. It is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems. Gradient Tree Boosting models are used in a variety of areas including Web search ranking and ecology. We are using it in this project because its natural handling of data of mixed type, great predictive power and robustness to outliers in output space (via robust loss functions). Here, we provide you a very good tutorial [7] and Scikit-Learn documentation [8] for reference.

For this part, you are required to write your model with the `GradientBoostingRegressor` in the module of `sklean.ensemble`.

- ✍ Parameter tunning/selection is required. First, `GradientBoostingRegressor` accepts different loss functions. They could be one of ls, lad, huber, quantile, select one of them and state your reason in `prjreport.pdf`. Second, `learning_rate`, `n_estimators` and `max_depth` are three critical parameters.Please state what role do they play in the model, what value do you assign and why do you select these values in `prjreport.pdf`
- Add scripts to the file `regression.py` to train a GBRT model `gbrt_model` using scikit-learn function with the training data. Please name your GBRT model with the exact name `gbrt_model`.

4.2 Predicting on Test Data (10pts)

In this part, you are required to test your result with 4 evaluation methods:Root Mean Squared Error, Top_1, Top_3 and Average_Rank.

Root Mean Squared Error is a commonly used evaluation method for regression model. If your prediction is $\hat{y_i}$, the groud truth is y_i for horse i($1 \leq i \leq n$). Then the Root Mean Squared Error for these n horses in the test data are $\sqrt{\frac{\sum_{i=1}^{n}(y_i-\hat{y_i})^2}{n}}$. Here, $\hat{y_i}$ is the predicted `finish_time` and y_i is the ground truth in test data.

With regression results, we can easily apply rank analysis:

Top_1 is defined as the percentage/probality when your prediction of top_1 horse(horse with shortest finish_time) for each race is actually the true top_1 horse. Top_3 is defined as the percentage/probability when your prediction of top_1 horse for each race is actually within true top_3 horses for each race. Average_rank is defined as the average true rank of top_1 horse based on your prediction over all races.

For example, when you predict for 3 races and your predicted top_1 horse is actually ranking 1, 3, 5 in these races. Top_1 is 1/3, Top_3 is 2/3 and Average_Rank is 3.0.

- ✍ Record your best result in the form `(model_name, RMSE, Top_1, Top_3, Average_Rank)` for both SVR and GBRT model. Here, you are required to save your best result together with chosen parameters `prjreport.pdf` and we will score this part by your prediction results.
- ✍ Sometimes your result will be dramatically better if you have done some data normolization over both traing_data and training_label. Observe that finish time of different horses is sometimes really close to each other and other features like 'declared_horse_weight' are much larger than others. Please try to normalize them and retrain your model to show whether normalizaiton improves the result. Here, scikit-learn has provided some useful tools [9] like `StandardScalar` in the module of `sklearn.preprocessing`.

  Note that you have to use the mean and variance of the `training data` to normalize/resume the prediction result to get RMSE. Please state your prediction RMSE with and without normalization and analyse the result in `prjreport.pdf` for both SVR and GBRT model.
- Add scripts to the file `regression.py` to do regression using SVR and GBRT model with the test data. Also save your normalizaiton code in `regression.py`.

5 Betting Strategy (Bonus up to 10 pts)

Our ultimate goal of this project is to see whether we can win some money using our predicton model. Suppose we can $1 on for every race. Our betting strategy is to bet all $1 for the predicted winning horse for each race. Concretely, if our prediction is correct for the winning horse, we will receive $1 $\times$ *odds* money. Otherwise, we will lose $1. The final result is positive if we win some money and negative if we lose.

✍ Here, you are required to save your results for each model(4 classification models and regression model with/without normalizaiton) in `prjreport.pdf`. One difficulty is how to decide a unique winning horse based on your classification and regression results, please state how you solve it in `prjreport.pdf`.

✍ Please analyse the result and come up with your own betting strategy to see if you can improve the result(lose less money or win more) in `prjreport.pdf`. Hint: Sometimes we can choose to bet or not on one race based on our confidence of our result. For example, we are more confident if the score of the winning horse if much higher than others. Also you can take the odds into consideration. Any attempts are encouraged and will get bonus scores accordingly. **The bonus you will get is based on both your idea and your results.**

⌨ Add scripts to the file `betting.py` to calculate the total winning money with the test data. Also save your own betting strategy code in `betting.py`.

6 Visualization (20 pts)

Visualization is a great way to understand data and expose some underlying relationships and patterns. In this section, we will take advantage of a powerful Python package called `matplotlib` to visualize the data. We only use the training data in this part. A plotted figure should convey clear information, so we have some basic requirements in this section:

- All the plots should have titles indicating the key message of the figure. You could give your own title.
- All the plots should have label texts for all the axes.

6.1 Line Chart of Recent Racing Result (4 pts)

A line chart or line graph is a type of chart which displays information as a series of data points connected by straight line segments. A line chart is often used to visualize a trend in data over intervals of time. Here, we want to visualize the history racing result of some specific horse.

⌨ Write script in `plot/line_chart.py`. Your program should be interactive so that it takes a horse ID as input, and outputs a line chart that shows the finishing positions of 6 recent races that the horse attended. The x-axis is the `game_id` of the 6 recent games, and the y-axis is the finishing positions. The figure title should specify the horse ID. *Hint:* You may use `plot()` in `matplotlib`.

✍ Select two horses that you are interested in, put down the plots of these two horses in your report, and briefly describe what you observe from the plots.

6.2 Scatter Plot of Win Rate and Number of Wins (4 pts)

Scatter plot is a common way to visualize the relationship between two quantitative variables. The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis. Now we want to find the "best" horse and the "best" jockey. Intuitively, the "best" one should have a high win rate and have won a large number of games.

⌨ Write script in `plot/scatter_plot.py` to draw two scatter plots. One is for horses and the other is for jockeys. These two plots should be in the same figure. Each point represents a horse (or jockey).

The x-axis is the win rate, and the y-axis is the number of wins. Choose a threshold and label the name of the horses (or jockeys) who reach the threshold. E.g., if a horse's win rate is larger than 0.5, and wins more than 4 games, then you should annotate the point of this horse with its name. *Hint:* You may use `subplot()` and `scatter()` in `matplotlib`, and you could set the `alpha` blending value to make the dots more transparent.

✍ Put down the plot in your report, and write down the "best" horse and the "best" jockey in your opinion, and briefly explain why.

6.3 Pie Chart of the Draw Bias Effect (4 pts)

Pie chart is a way to visualize the distribution of categorical data. In this part, we explore the effect of draw bias in horse racing. The draw refers to the stall a horse will start the race from. The draw is normally chosen at random on the day the horses are declared to run. Obviously, the inside lane would hold an edge over the field as they have a shorter distance to the bend, in comparison to the other lanes. Does this really make a difference?

⌨ Write script in `plot/pie_chart.py` to plot a pie chart and show the win rates of different draws. The categories in the pie chart should be the different draws, and the fraction of each category represents the probability that the horse who wins the race starts from the corresponding draw. You should specify the draw numbers using label texts and display the percent value for each category in the figure. *Hint:* You may use `pie()` in `matplotlib`.

✍ Put down the plot in your report, briefly describe what you observe from the plots, and answer whether low draws really have a considerable advantage?

6.4 Bar Chart of the Feature Importances (4 pts)

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. We have used several kinds of machine learning algorithms to make predictions. Since there are so many features, maybe some features are informative, while others are relatively irrelevant to the racing result. Now we want to find out the importance of each feature.

⌨ Write script in `plot/bar_chart.py` to plot a bar chart and show the importance of each feature. The x-axis is for different features and you should label them with the feature names. The y-axis is the feature importance. You should use random forest classifier to evaluate the importance of the features. The bars should be sorted from left to right in descending order. *Hint:* You may use `bar()` in `matplotlib`. You may learn from this example.

✍ Put down the plot in your report, and briefly describe what you observe from the plots.

6.5 Visualize SVM (4 pts)

A good way to understand SVM is to visualize it.

⌨ Write script in `plot/SVM_visual.py` to visualize the SVM classifier. Since it is hard to visualize high-dimensional data, for the input data X, we only consider these two features: `recent_rank` and `jockey_ave_rank`. Also, for the target y, we only care about whether the finishing position is in top 50%. You need to plot the classification boundary of your SVM classifier. You should use `SVC(kernel='linear')` model, and add a legend to your plot. *Hint:* You may learn from this example.

✍ Put down the plot in your report, and briefly describe what you observe from the plots.

7 Submission

Instructions for the submission are as follows. **Please follow them carefully.**

1. Test your code before submission. Any python scripts with syntax errors wont be accepted.
2. Zip all the required files into a single zipped file named `<student-id>_prj.zip`, where `<student-id>` should be replaced with your own student ID. e.g., `1155012345_prj.zip`. The submitted zip file should have the same structure as the provided `prj.zip`. Please do not change the names of provided files.
3. Submit the zipped file `<student-id>_prj.zip` to `cuhkcsci3320@gmail.com` before the deadline 17:00 on Friday, April 27, 2018.
4. The grade of your project will be sent back to you in two weeks after the deadline. If you have any questions about your grade, please contact the TAs.
5. The realistic horse racing might be far from our exercising dataset. We strongly do NOT recommend you to spend money on horse betting.

References

[1] "Logistic regression scikit-learn documentation," http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[2] "Naïve bayes scikit-learn documentation," http://scikit-learn.org/stable/modules/naive_bayes.html.

[3] "SVM scikit-learn documentation," http://scikit-learn.org/stable/modules/svm.html.

[4] "Random forest scikit-learn documentation," http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[5] "Support vector regression scikit-learn documentation," http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.

[6] "Support vector regression wiki," https://en.wikipedia.org/wiki/Support_vector_machine#Regression.

[7] "Gradient boosting tutorial," http://xgboost.readthedocs.io/en/latest/model.html.

[8] "Gradient boosting regressor scikit-learn documentation," http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.

[9] "Standard scaler(normalizer) scikit-learn documentation," http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.