STAT4012
Statistical Principles of Deep Learning with Business Application
Comparison of deep learning algorithms on horse racing prediction

Cheuk Nam CHUNG [*] Ting Tin MA [†] Chung Yin LAM [‡]
Kai Wai CHAN [§]

May 7, 2021

Abstract

In this project, we would study the data set of horse racing in Hong Kong and attempt to conduct prediction on finishing position of horses using different characteristics. In particular, random forest and deep neural network are used. We would access their performances and compare them in theoretical aspects. Also, different optimizers would be used to build up the neural network. It is remarkable that in general, there is no criteria for choosing the number of layers and neurons, so we regard them as a hyper-parameter to be tuned by the K-fold cross validation.

[*]1155109131@link.cuhk.edu.hk
[†]1155109956@link.cuhk.edu.hk
[‡]1155108930@link.cuhk.edu.hk
[§]1155109231@link.cuhk.edu.hk

1 Introduction

The project is organized as follows. In Section 2, we review the basic theory concerning random forest and deep neural network. In Section 3, the mathematical notation used frequently in this project is introduced.

2 Literature Review

2.1 Random Forest

f In prior to introducing random forest, we first discuss about the classification tree.
AIM of classification tree: For $S = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{N}$, where $\boldsymbol{x_i} = \left(x_i^{(1)}, x_i^{(2)}, \cdots, x_i^{(K)}\right)$ contain K features and $y_i \in Y = \{\beta_1, \cdots, \beta_M\}$ with M labels, partition S into disjoint $R_1, \cdots, R_n$ s.t. $\forall x_i \in R_j$, $\hat{y}_i = \beta_j$, $\forall j = 1, \cdots, M$.
In order to attain this task, the notion of entropy, which is a **measure of concentration biasedness of distribution**, is introduced.

- For discrete X with pmf $p(\cdot)$, $H(X) \coloneqq -\mathbb{E}[\log_2 p(X)] = -\sum_{x \in X} p(x) \log_2 p(x)$. For $X \sim \text{Bern}(p)$, $H(X) = -p \log_2 p - (1-p)\log_2(1-p)$. In practice, we use empirical dist to estimate the entropy.

- For continuous X with pdf f, $h(X) \coloneqq -\mathbb{E}[\log f(X)] = -\int_X f(x) \log_2 f(x) dx$. [NOT direct generalization of Shannon entropy; does not impose non-negativity]. Degenerating $X \Rightarrow H(X) = -\infty$ instead.

- **Conditional Entropy:** Suppose X, Y are discret RVs.

$$H(Y|X) \coloneqq -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(y|x)$$

  is a measure of chaotic information in Y given X is known.

The increase in entropy implies the decrease of *Useful Distinguishable Information* and increase of *Chaotic Information* (Randomness). This is true as indeed the definition of entropy is the negation of log-likelihood function, therefore, as the log-likelihood increases, i.e., we have more information concerning the model and hence the chaotic information decreases. Suppose we are given a set of decision rule, then we can measure how much uncertainty is eliminated through the classification, which motivate us to consider the notion of information gain. Suppose x^{k_i} chosen as attribute at node i with sample S. The Information Gain is defined as

$$IG(S, x^{(k_i)}) \coloneqq H(S) - \sum_{v \in V(x^{(k_i)})} p_{x^{(k_i)}}(v) H(S_v) = H(S) - \sum_{v \in V(x^{(k_i)})} \frac{|S_v|}{|S|} H(S_v),$$

where $V(x^{(k_i)})$ is set of possible values taken by $x^{(k_i)}$ and S_v is collection of sample points from S with feature $x^{(k_i)}$ taking value $v \in V(x^{(k_i)})$. We then choose attribute with larges IG as the classifier if IG large enough. Noticing that entropy is not the only measure of impurity. There are other impurity measures. Let X be discrete RV with pmf $p(\cdot)$. Define

- **Gini-Index:** $G(X) \coloneqq 1 - \sum_{x \in X} p^2(x)$.
- **Misclassification Error:** $M(X) \coloneqq 1 - \max_{x \in X} p(x)$.

Then Can replace H by G or M to define general Im and replace $IG(S, x^{k_i})$ by $\Delta(x^{(k_i)})$ correspondingly. We are then ready to conduct **splitting for continuous attributes** and it is indeed done by maximizing the entropy difference. For simplicity, let $x = (x_1, \cdots, x_n)$ be the continuous attribute.

Algorithm 1: Choosing Threshold value for attribute $x = (x_1, \cdots, x_K)$

```
[1] Input: (i) Data set $x = (x_1, \cdots, x_K)$.   &   (ii) Impurity Measure $Im(\cdot)$.
      (Entropy/Gini/Misclassification Error)
[2] (1) Sort $x$ into $\tilde{x} = (x_{(1)}, \cdots, x_{(K)})$ and define $c_i = (x_{(i+1)} + x_{(i)})/2$ for $i = 1, \cdots, K-1$.
[3] (2) Let $x < c_i$ be the classification rule; Calculate the induced average Impurity measures:
      $Im_k \coloneqq (\#x \le c_i)/n \times Im(\{y_i : x_i \le c_i\}) + (\#x > c_i)/n \times Im(\{y_i : x_i > c_i\})$
[4] (3) Find $d = \arg\min_d Im_d$ and $c_d$ is best criteria within $\{c_1, \cdots, c_{K-1}\}$. Repeat (1)-(3) by
      Bisection Method.   Return: Classification rule $x \le c_d$.
```

Following the above algorithm, one can construct a classification tree. However, one have to notice that classification tree might suffer from over-fitting. In order to prevent it, one suggest a slightly modified version of classification tree, which is known as **bagging**. It suggests to draw B bootstrap samples to build B trees, and use the average of the predicted result as the estimate so that it can reduce the effect of over-fitting. While one suggested its generalization, which is the famous **Random Forest**.

```
Algorithm 2: Random Forest
[1] Input: (i). Training data set $S$ (ii). Number of bootstrap sample to be drawn: $B$.
[2] (iii). Size of each bootstrap sample: $N$. (iv). Number of random selected feature: $m$.
    (v). Testing data set $T$
[3] (1) Draw $S_1, \cdots, S_B \subseteq S$ with each $|S_b| = N$.
[4] (2) For each $b = 1, \cdots, B$, build a classification tree by regarding $S_b$ as the full data-set and
    $m$ randomly chosen feature variables.
[5] (3) Pick some $x \in T$, for $b = 1, \cdots, B$, denote the classified result for $x$ in the $b$-th
    classification tree as $\widehat{f_b}$.
[6] Return: The estimated value for class of $x$ is given by $\sum_{b=1} \widehat{f_b}/B$.
```

By allowing the choice of feature variables to be used as a random variable, the random forest can decorrelate the trees and hence the variability of the estimated result is lower. It is similar to the idea of natural selection, claiming that if a feature variable have a strong predictive/classification power on the problem, it would appear in most of the trees and hence still taking a significant role in the random forest, while the effect of those "weak" one will be diversified. There are some remarks:

1. Suppose there are K feature variables, a rule of thumb is to take $m = \lfloor \sqrt{K} \rfloor$ for classification problem and $m = \lfloor K/3 \rfloor$ for regression problem.
2. The estimate converges in L^2 norm under some mild assumptions.

2.2 Deep Neural Network

Deep neural network (DNN) is an artificial neural network (ANN) with multiple hidden layers, which attempt to mimic the interdependent neurons in human brain that process data and transmit them to useful information. In a DNN, we train the weights in the hidden layers as well as the output layer through iterative forward propagation and back-propagation using training dataset. After training, we fit our targeted data into the trained DNN model to obtain the predicted results.

The forward propagation process is summarized as follows:
For hidden layer $1, \ldots, L$ and iteration $1, \ldots, T$:
In the lth hidden layer and the tth iteration, define $W^{(l,t)} := \left(w_1^{(l,t)}, w_2^{(l,t)}, \ldots, w_D^{(l,t)}\right)$ where D is the dimension of the input to that layer and $w_d^{(l,t)} = \left(w_{1d}^{(l,t)}, w_{2d}^{(l,t)}, \ldots, w^{Kd(l,t)}\right)^\top$,

$b^{(l,t)} = \left(b_1^{(l,t)}, b_2^{(l,t)}, \ldots, b_K^{(l,t)}\right)^\top$, where K is the number of neutrons in the l+1th hidden layer. Further denote

$$\begin{aligned} z^{(l,t)} &= W^{(1,t-l)} a^{(l-1,t)} + b^{(l,t-1)} \\ a^{(l,t)} &= f(z^{(l,t)}) \end{aligned}$$

where Activation function $f(\cdot)$ is used in each hidden layer to transform the weighted outputs z. There are several choices for activation function. For a ReLu function, we have

$$a^{(1,t)} = f(z^{l,t)}) = \max\left(0, z^{(l,t)}\right)$$

Mini-batch normalization is typically used for addressing the vanishing gradient problem by normalizating the input $z^{(1,t)} = \left(z_1^{(1,t)}, z_2^{(1,t)}, \ldots, z_K^{(1,t)}\right)^\top$ into $z^{r(1,t)} = \left(z_1'^{(1,t)}, z_2'^{(1,t)}, \ldots, z_K'^{(1,t)}\right)^\top$ Using a mini-batch of size M we have

$$\begin{aligned} \widehat{\mu}^{(1,t)} &= \frac{1}{M} \sum m = 1^M z_m^{(!,t)} \\ \left(\widehat{\sigma}^{(1,t)}\right)^2 &= \frac{1}{M} \sum_{m=1}^{M} \left(z_m^{(1,t)} - \widehat{\mu}^{(1,t)}\right)^\top \left(z_m^{(1,t)} - \widehat{\mu}^{(1,t)}\right) \\ \widehat{z}_m^{(1,t)} &= \left(\left(\widehat{\sigma}^{(1,t)}\right)^2 + \epsilon I_K\right)^{-\frac{1}{2}} \left(z_m^{(1,t)} - \widehat{\mu}^{(1,t)}\right) \\ z_m'^{(1,t)} &= \gamma^{(1,t)} \widehat{z}_m^{(1,t)} + \beta^{(1,t)} \end{aligned}$$

In a fully-connected network like DNNs, the weights and biases trained by neurons heavily rely on the training dataset, and therefore often presents overfitting problems. Reducing the interdependency of neurons helps alleviating the problem, which can be done through an addition of a dropout layer to randomly drop out neurons throughout forward propagation process. The probability of dropout for a neuron in the l-th hidden layer is an i.i.d follows Bern(p), where $1-p$ is the dropout rate. Dropout layer is applied after batch normalization to ensure maintained past variances obtained during training.

Before back-propagation is started, we need to choose a loss function to measure the deviation of the weightings in predicting the outcome, of which we aim to minimize. For a mean square error (MSE) function of a mini-batch size M, we have

$$E_m^{(t)} = \left(y_m - f(x_m^{(t)}, W_m^{(1,t)}\right)^2 \quad \text{and} \quad E^{(t)} = \frac{1}{M}\sum_{m=1}^{M} E_m^{(t)}$$

In DNN, the optimizer for minimizing the loss function is gradient descent method. Here we define $g^{(l,t)} = \frac{\partial}{\partial z^{(l,t)}} E^{(t)}$. The general methodology of backpropagation is iteratively deriving gradients where the weights are adjusted accordingly, in achieving a minimized loss function. Using Adam as optimizer, we have

$$W^{(l,t+1)} = W^{(l,t)} - \eta\left(\widehat{V} - \epsilon I_K\right)^{-\frac{1}{2}} \widehat{m}^{(l,t)}$$

where

$$\begin{aligned}
m^{(l,t)} &= \beta_1 m^{(l,t-1)} - (1-\beta_1) g^{(l,t)} \\
\widehat{m}^{(l,t)} &= \frac{m^{(l,t)}}{1-\beta_1^{(t)}} \\
V^{(l,t)} &= \beta_2 V^{(l,t-1)} + (1-\beta_2) g^{(l,t)} \ddot{g}^{(l,t)} \\
\widehat{V}^{(l,t)} &= \frac{V^{(l,t)}}{1-\beta_2^{(t)}}
\end{aligned}$$

The Adam optimizer captures both the adaptive gradient property and drifting momentum, and hence speeds up the converging process and enhances effectiveness in minimum convergence.

The number of parameters in the $(l+1)$-th hidden layer is $(n_l+1)(n_l+1)$. For batch normalization in a hidden layer l, we would require $\mu, \sigma^2, \beta, \gamma$ for each neuron to be estimated, concluding a total number of $4n_l$ parameters.

The general algorithm is given by **Algorithm 3**.

Algorithm 3: DNN model

```
[1] (1) Input: (i) Dataset $x = (x_1, \cdots, x_K)$. (ii) Impurity Measure $Im(\cdot)$.
      (Entropy/Gini/Misclassification Error)
[2] (2) For each iteration $t$:
[3] (2a) Forward propagate through hidden layers towards the output layer, with possible
      inclusion of batch-normalization and drop-out layer, to obtain the activated output
      $\widehat{y} = f\left(x^t, W^{(l,t)}\right)$ for each datum.
[4] (2b) With a batch size $M$, obtain the loss function $\epsilon_m^t = \left(y_m - f(x_m^t, W_m^{(l,t)})\right)^2$ and
      $E^t = \frac{1}{M}\sum_{m=1}^M \epsilon_m^t$. Optimize the loss function through Adam Scheme and backward
      propagate to update $w_d^{(l,t)} = \left(w_{1d}^{(l,t)}, w_{2d}^{(l,t)}, \ldots, w_{Kd}^{(l,t)}\right)^\top$ and
      $b_{(l,t-1)} = (b_1^{(l,t)}, b_2^{(l,t)}, \ldots, b_K^{(l,t)})^\top$
[5] (3) Return: estimated output $\widehat{Y} = (\widehat{y}_1, \widehat{y}_2, \ldots, \widehat{y}_n)$
```

3 Mathematical Notations

In this project, we adopt the dataset from here. In general, our goal is to predict the ranking of each horses in a race given some associated explanatory variable. We would apply two different models: (1) Random Forest and (2) Deep Neural Network to achieve the goal. While there is some technical reasons to be explained later, we will use different collections of feature variables for those two model. WLOG, suppose for a algorithm, we have the dataset denoted by $S = \{\widetilde{X}_i, \widetilde{y}_i\}_{i=1}^n$, where

- $\widetilde{y}_i \in \mathbb{R}^{h_i}$, where h_i is the number of horse in the i-th race.
- $\widetilde{X}_i \in \mathbb{R}^{h_i,k}$, where k is the number of explanatory variables associated to each horse.

Therefore, the most ideal case is that we can find a function f that maps $\widetilde{X}$ to $\widetilde{y}$. However, there is two remark to be noticed.

1. f is not a well-defined function as the dimension of input matrix is not fixed.
2. We cannot regress the feature variable of each horse to its rank separately, i.e. regress $[\widetilde{y}_i]_j$ against $[\widetilde{X}_i]_{j,1}, \cdots, [\widetilde{X}_i]_{j,k}$ for $j = 1, \cdots, h_i, i = 1, \cdots, n$. It is because that the rank of a horse in a race is intrinsically dependent on its competitors, and hence we have to consider the matrix itself as a feature matrix.

In order to simplify the setting, we consider applying flattening to covariate matrices as introduced as a step in the Convolutional Neural Network. However, it still faces the problem in remark(1). Noticing that $\max_j(h_j) = 14$ (As there are at most 14 horses per race), we append the flattened covariate vectors to define

$$X_i \coloneqq \Big([\widetilde{X}_i]_{1,1:k}, [\widetilde{X}_i]_{2,1:k}, \cdots, [\widetilde{X}_i]_{h_i,1:k}, \underbrace{0,0,0,0,\cdots,0,0,0}_{\text{Number of zeros}=(14-h_i)k}\Big)^T \in \mathbb{R}^p ,$$

where $p \coloneqq 14k$, for $i = 1, \cdots, n$. As somehow expecting the predicted label $\widehat{y}$ equating $\widetilde{y}$ entry-wisely is an too ideal/harsh goal, and in reality, most of the horse racing strategy does not require us to well specify the ranking of all participants in a race. Furthermore, it is in hard to define a generally accepted metrics to measure the accuracy of specifying a high dimension object. Therefore, it motivates us to consider the following two goals.

1. Define $y_i = \arg\max_j [\widetilde{y}_i]_j$, i.e., the y_i-th horse won the i-th race for $i = 1, \cdots, n$. The goal is to regress $y \in \{1, \cdots, 14\}$ against $X_i \in \mathbb{R}^p$, which reduces to be the standard regression setting.

2. Define $y_i^* \coloneqq \{d_{i1}, d_{i2}, d_{i3}\}$, where d_{ij} is characterized by the following: the d_{ij}-th horse is of rank j in the i-th race, i.e., y_i^* is a collection of those who are the top three in the i-th race. Notice that y^* is a set without ordering structure. We aim to regress $y^* \subset \{1, \cdots, 14\}$ against $X_i \in \mathbb{R}^p$.

4 Model I: Random Forest

4.1 Variable Setting

4.1.1 Variable Selection

The datasets 'races.csv' and 'runs.csv' are available on the website of Kaggle, an online community of data scientists and machine learning practitioners. The former file contains data of each horse racing match from 02/06/1997 to 05/08/2005 in Hong Kong, while the latter consists of data of each horse in each match.

Since we aim at building the model to predict the ranks of horses in each match given the data that we can observe before the start of the match, the data gathered after the

	Data Name	Data Type	Remarks
1	Date	continuous	presented as string
2	Venue	categorical	Happy Valley or Shatin racecourse
3	Race Number	discrete	race number of each match day
4	Configuration	categorical	length of home straight and width of the starts
5	Surface	categorical	binary with 1 stands for wet surface
6	Distance	discrete	5 different distances
7	Going	categorical	ranks of width of track
8	Horse Rating	categorical	ranges of horse rating
9	Prize	discrete	in HKD
10	Race Class	discrete	1-5 with the highest class
11	Sec Time	continuous	time elapsed of the first ranked horse from the first ranked horse from each checkpoint to the next one
12	Time	continuous	time elapsed of the first ranked horse from the first ranked horse from the start to the each check point
13	Place Combination	categorical	horse number of the final thirds
14	Place Dividends	continuous	dividend of a $10 bet of 'PLACE' of each horse in the final thirds
15	Win Dividend	continuous	dividend of a $10 bet of 'WIN' of the winning horse

Figure 1: 'races.csv' data

start of the race are discarded. However, we still keep the win odds and place odds as it is believed that they are the significant factor in determining the rank of the horses, and should not have a huge difference between the last observable odds and the final odds given in the data.

We have discarded some other variables. IDs of horses, jockeys and trainers are removed as we believe individual effect is not significant in the race. Race numbers and date are removed as we assume the time of race along a day and a year will not matter a lot within a race. We also choose race class (1-5) and drop prize and horse rating range to avoid multicollinearity as they are highly correlated. We also use the 'result' in 'runs.csv' only and remove the data related to final rank and odds in 'races.csv' as they are redundant. 'Surface' is also removed as configuration and further explains the condition of the track. We use horse numbers instead of ratings as each horse as the numbering is in accordance to the rating and a portion of the data does not have the exact rating of the horses (marked as 60).

	Data Name	Data Type	Remarks
1	Race ID	discrete	numbering of matches in the dataset
2	Horse number	categorical	
3	Horse ID	categorical	
4	Result	discrete	Rank of each horse in each match
5	Horse age	discrete	
6	Lengths Behind	continuous	length between the position of each horse and the winning horse (in meters)
7	Horse Country	categorical	
8	Horse Type	categorical	
9	Horse Rating	discrete	
10	Declared Weight	continuous	weight of the horse and jockey combined (in lbs)
11	Actual Weight	continuous	weight of the jockey only (in lbs)
12	Draw	discrete	Draw of the gate in the start (1 stands for the most interior gate of the track)
13	Position sec	discrete	Position of each horse in each checkpoint
14	Behind Sec	continuous	distance between the winning horse and the position of each horse (in horse length)
15	Finish Time	continuous	
16	Win Odds	continuous	
17	Place Odds	continuous	
18	Trainer ID	discrete	
19	jockey ID	discrete	

Figure 2: 'runs.csv' data

For the race ID, we still keep them at this stage as a mark of extracting a single match data. Also note that 'result' and 'won' are the response variable we want to predict. Therefore, we have the dependent variables in use in Table 3.

	Data Name	Data Type
1	Venue	categorical
2	Configuration	categorical
3	Distance	discrete
4	Race Class	discrete
5	Going	categorical
6	Horse Number	discrete
7	Horse Age	discrete
8	Horse Country	categorical
9	Horse Type	categorical
10	Declared Weight	continuous
11	Actual Weight	continuous
12	Draw	discrete
13	Win Odds	continuous
14	Place Odds	continuous

Figure 3: Variable in used

4.1.2 Variable Transformation

In order to facilitate calculation, we transform some of the categorical variables into number.

1. **Venue:** We use 0 and 1 stands for Shatin racecourse and Happy Valley racecourses respectively

2. **Configuration:** As there are only six classes in the dataset, we use integer to classify them as they mainly differ in the width of the start.

Original	Transformed
A	6
A+3	5
B	4
B+2	3
C	2
C+3	1

3. **Going:** We use the following table to transform the class to integers to show the gradual change between each class. Note that 'Wet', 'Wet Fast', 'Wet Slow', 'Slow' are actually the class of all weather track while the others are of the turf track. We try to combine them into the same scale.

Original	Transformed
Wet	1
Wet Fast	1
Firm	1
Good to firm	2
Good	3
Good to yielding	4
Yielding	5
Yielding to soft	6
Soft	7
Wet Slow	7
Slow	7

4. **Horse Country:** Since there are horses from 15 countries, the dimension of dependent variable vector would be boosted a lot if we transform to into zero-one vectors. Therefore, we try to calculate the average position of the horses in all races for each country, and replace the country with the rank of the country. (1 stands for the best country and 15 stands for the worst).

5. **Horse Type:** Same method is used as 'Horse Country' as there are 9 different horse types.

4.1.3 Formation of the dependent variable matrix and the response variable matrix

1. the dependent variable matrix $\boldsymbol{X}$: Note that $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ where $N = 6349$ races and $d = 5$ match variables 14 horses $\times$ 8 horse variables.

   (a) Extract the match variables $x_i^{(1)} \in \mathbb{R}^5$ of each match and the horse variables $x_i^{(2)} \in \mathbb{R}^{N \times 13}$ where they have the same race ID. then we discard the horse number and race ID sort the horse variables using the draw.

   (b) Add zeros for NaN entries (missing data).

   (c) Extract the sorted values of column "Result" and "Won" as $y_i^{(1)}$ and $y_i^{(2)}$ and discard them together with "Horse Number".

   (d) Note that it is possible that $n < 14$ since there are matches with less than 14 horses. Add rows of zeros until the rows of x_2 reaches 14.

(e) Concatenate $x_i^{(1)}$ and $x_i^{(2)}$ gives x_i, a vector contains all data of race i.

(f) Concatenate all x_i gives the entire $\boldsymbol{X}$

Remark: The horse number is revealed in the index of the $\boldsymbol{x}$. For example, in the sixth match, the horse no.3 data are $x_{6,(5+3\times1):(5\times3+8)}$, corresponding to the 7-14 variables in Table 3.

2. the response variable vector $\boldsymbol{Y}$: We have constructed four different response vectors, namely

   - Win: a matrix $\boldsymbol{y} \in \mathbb{R}^{n\times14}$ formed from rows $y_i^{(1)}$ indicating the winning horse as 1 and others as 0.
   - Result: a matrix $\boldsymbol{y} \in \mathbb{R}^{n\times14}$ formed from rows $y_i^{(1)}$ indicating the position of all horses.
   - Place Binary: Similar to 'Win' but indicating the first three horses as 1 and others as 0.
   - First Four: Similar to 'Result' but indicating the position of only first four horses with others being 0.

4.2 Model Implementation and Results

In this section, we attempt to conduct task (1). We proceed by two different approach.

1. Apply Random Forest to the training data set $\{X_i, y_i\}$ and test its performance through the testing data set.

2. Apply Principal Component Analysis to the training data set to get the Principal Components (PCs). Then apply Random Forest on the transformed data set.

The motivation to consider two different approach is as follow. Notice that there is plenty of feature variable in our setting, while the sample size is not large enough relative to the number of feature variables. It follows that the number of sample in each node would be small, (approximately 10 per node) and it is not sufficient to train up the Random forest to get satisfactory result. Therefore, one should consider PCA instead. Recall that the PCs of a data set are linear combinations of the feature variables which contains most of the variability of the dataset. Hence using only few number of PCs may already be sufficient for analysis, meanwhile keeping relatively large number of samples under each node under the trees. Notice that by default, there is 100 bootstrapped sample Again, we split the training and testing data set in ratio of 7: 3 and obtained the following result

1. For the standard random forest, the accuracy for successfully guessing the first one and the first three in a race and is 0.084 and 0.2415 respectively.

2. For the PC transformed random forest, the accuracy for successfully guessing the first one and the first three in a race and is 0.082 and 0.2236 respectively.

It is notable that both of the accuracy are greater than 0.071 = 1/14, i.e. the probability of random guess. However, the performance of both method is not satisfactory enough. In order to show that the Random Forest is not being able to handle our problem, we consider the following:

1. Varying the number of bootstrapped tree B against the accuracy.

B	1	11	21	31	41	51	61	71	81	91
Accuracy	0.116	0.092	0.090	0.078	0.080	0.088	0.092	0.083	0.083	0.085

2. Varying the number of Principal Components (PCs) used against the accuracy.

# of PCs	1	2	3	4	5	6	7	8	9	10
Accuracy	0.074	0.067	0.081	0.094	0.086	0.092	0.075	0.080	0.080	0.082

We can observe that both of table above do not show monotonically increasing trend, instead they are slighly fluctuating along a horizontal line, which should not be the case if the algorithm works well.

4.3 Interpretation of the Result and Algoritm

It seems to be unexpected that the result is only slightly better than the random guess. However, it is still explainable. Recall from the literature review that Random forest is indeed a generalization of classification tree, which sequentially maximize the decrease in entropy in each step. However, this algorithm is only sequentially optimal, which means it consider each feature variable one by one, without taking their joint structure into account, and hence the result could be a poor one. Further more, the classification is a straightforward one, as in each step, the tree is branched by the following criteria: (i) $x < c$ (ii) $x \geq c$, which may only works for simple problems. However, such criteria is hard to interpret as each of the variable may not be too meaningful or having much predictive power. It is not hard to believe that the horse racing result is a highly non-linear problem, and also there may be joint dependence between feature variable. Hence hard to use the random forest to make a good prediction.

5 Model II: Deep Neural Network

5.1 Model Implementation

In order to handle the horse racing problem, which its prediction result is not satisfactory in the Random Forest Algorithm, we consider applying Deep Neural Network in this section. It is natural to believe that Deep Neural Network would perform better than Random Forest because of the following:

1. In each hidden layer, each neuron is a linear combination of outputs in the previous layer. Hence it somehow incorporate and generalize the structure of PCs, which are also linear combination of the feature variables.

2. After obtaining the value of neuron, there is some **non-linear** activation function to transform the neurons, which enable the deep neural network to capture the non-linear and complicated structure between the feature variable and the ranking.

However, the are many **hyperparameter** to be set within the Deep Neural Network.

1. **Epoch Size**: Number of time of data set being used for Gradient Descent.

2. **Batch Size**: Number of datum within a batch, which is a trade off between rate of convergence and computational efficiency.

3. **Number of Hidden Layer & Number of Neuron in each Hidden Layer**

4. **Objective Function**: The function to be minimized through the neural network.

5. **Numerical Method**: Method to be combined with the Gradient Descent, eg: Adam, Adagrad, RMSprop and SGD.

6. **Learning Rate**: Learning rate being too high or low would lead to exploding or vanishing gradient problem respectively.

7. **Activation function**: Choosing softmax or ReLU type function.

8. **Dropout rate in dropout layer**

In most of the time, people choose the above hyperparameters without clear reasoning. In this section, we attempt to choose a good combination of them through the K-Fold Cross Validation. However, there are too many hyperparameter to be chosen and hence it is computationally intensive to consider all combination of them for the K-Fold Cross Validation. Instead, we propose the following **Sequential K-Fold Cross Validation Algorithm**.

Algorithm 4: Sequential K-Fold Cross Validation

```
[1] Input: Initialization of hyperparameters $\Theta^{(0)} = (\theta_1^{(0)}, \cdots, \theta_M^{(0)})$, where $M$ = is the number
      of hyperparamaters
[2] For $k$ from 1 to $M$, implement the standard K Fold cross validation to the following
      parameter space   $\Theta^{(k)} := \left(\theta_1^{(k-1)}, \cdots, \theta_{k-1}^{(k-1)}, \theta_k, \theta_{k+1}^{(k-1)}, \cdots, \theta_M^{(k-1)}\right)$
      to set $k$-th parameter $\theta_k^*$. Then set $\theta_k^{(k)} = \theta_k^*$ and for $i \neq k$, set $\theta_i^{(k)} = \theta_i^{(k-1)}$.
[3] Return: The choice of hyperparameters $\Theta^{(M)}$.
```

The algorithm above is sequentially optimal and strike balance between the optimality and computational cost. We then implement it to choose the hyperparameter in the DNN model mentioned above step by step. The initiali choice is as follow: epochs size = 30, batch size = 32, two hidden layers , 64 neurons per layer, softmax as output activation, mse as the loss funciton, adam with lr 0.001 as the optimizer, without Batch Normalization and dropout layer (dropout rate=0.2). In the following section, there is two type of accuracy measure: (i) proportion of successful prediction on the first one in race **(AC1)** and (ii) proportion of successful prediction on the first three in race **(AC2)**. If not otherwise specified, we are referring to (i).

Hyperparameter	Training			Validation		Testing	
Epoch Size	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
10	0.0539	0.3907	7.6049	0.1363	0.3848	0.1386	0.3709
20	0.0389	0.6150	11.9877	0.1384	0.3828	0.1315	0.3756
30	0.0299	0.7141	16.0192	0.1396	0.3893	0.1441	0.3858
40	0.0256	0.7585	20.7245	0.1329	0.3897	0.1480	0.3913
50	0.0200	0.8141	26.0475	0.1347	0.3830	0.1528	0.3811

Table 1: Step 1: Sequentially choosing the Epoch size by fixing other hyperparameters.

Comment: Epoch size is chosen to be 30. Noticing that for epoch size being 40 or 50, it suffers from over-fitting and becomes more significant as epoch size increases.

Hyperparameter	Training			Validation		Testing	
Batch size	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
8	0.0234	0.7839	51.5711	0.1349	0.3832	0.1252	0.3528
16	0.0273	0.7404	28.0031	0.1353	0.3783	0.1441	0.3874
32	0.0304	0.7116	15.3164	0.1459	0.3885	0.1512	0.4024
64	0.0361	0.6519	9.6173	0.1461	0.3909	0.1063	0.3701

Table 2: Step 2: Sequentially choosing the Batch size by fixing other hyperparameters.

Comment: Batch size is chosen to be 32. For batch size being 8/16, it takes too many time for training up the deep neural network, meanwhile it is of lower accuracy. For batch size being 64, the validation accuracy is much greater than testing accuracy, which is also undesirable. It is noticible that according to theory, the computation cost for Minibatch SGD is $O(N/\eta)$, where N is the batch size and η is the learning rate, and hence smaller batch size should leads to shorter iteration time, which contradicts the result above. One possible interpretation is as follow: the structure of the horse racing problem is too complicated, and using too few data to update the parameter in each step may likely to point its movement to a wrong direction and hence overall the algorithm is slower.

Hyperparameter	Training			Validation		Testing	
# of Hidden Layer	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
1	0.043	0.5597	15.2117	0.1457	0.4007	0.1378	0.3969
2	0.0299	0.7105	15.6574	0.1373	0.3838	0.1480	0.4055
3	0.0245	0.7701	17.3700	0.1304	0.3657	0.1433	0.3717
4	0.0218	0.7934	18.5156	0.1276	0.3641	0.1315	0.3583

Table 3: Step 3: Sequentially choosing the number of hidden layer.

Comment: Number of hidden layer is choosen to be 1

Hyperparameter	Training			Validation		Testing	
# of Neuron	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
16	0.0586	0.3169	13.5873	0.1428	0.3881	0.1268	0.3764
32	0.0515	0.4468	14.4558	0.1406	0.3907	0.1496	0.4087
64	0.0420	0.5793	14.8299	0.1443	0.4049	0.1543	0.4031
128	0.0288	0.7341	14.4243	0.1514	0.4171	0.1504	0.4150
256	0.0175	0.8476	16.1349	0.1554	0.4204	0.1543	0.4236
512	0.0101	0.9055	17.6551	0.1638	0.4232	0.1614	0.4016
1024	0.0086	0.9173	20.2531	0.1593	0.4210	0.1748	0.4173

Table 4: Step 4: Sequentially choosing the number of neuron in each hidden layer.

Comment: Number of neuron in each hidden layer is choosen to be 512. Notice that it appears to be overfitting for neuron number being 1021.

Hyperparameter	Training			Validation		Testing	
Objective Function	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
MSE	0.0113	0.8928	17.7074	0.1591	0.4307	0.1654	0.4165
Binary Cross Entropy	0.0467	0.9744	18.6723	0.1495	0.4061	0.1575	0.4055
Categorial Cross Entropy	0.1931	0.9835	17.6445	0.1493	0.4023	0.1535	0.3992

Table 5: Step 5: Sequentially choosing the objective function to be minimized by DNN.

Comment: MSE is chosen to be the objective function to be minimized by the DNN.

Hyperparameter	Training			Validation		Testing	
Numerical Method	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
Adam	0.0101	0.9041	17.4208	0.1642	0.4313	0.1811	0.4315
Adagrad	0.0676	0.0826	16.9065	0.0900	0.2985	0.0795	0.2606
RMSprop	0.0110	0.8968	18.9654	0.1642	0.4234	0.1803	0.4449
SGD	0.0686	0.0732	16.6312	0.0723	0.3165	0.0756	0.3433

Table 6: Step 6: Sequentially choosing the numerical optimizer.

Comment: Adam is chosen to be the numerical optimizer used. Noticing that the performance of Adagrad and SGD are both awful. The interpretation is as follow: For SGD, we basically run the iteration without updating the learning rate, and hence the performance might be bad, probably because of the exploding/vanishing gradient problem. While for Adagrad, even though the learning rate is adaptive, the performance is poor as the learning rate vanishing as the gradient terms accumulate. For Adam, it incorporates the ideas of RMSprop and Momentum with bias correction, leading to a better performance.

Hyperparameter	Training			Validation		Testing	
Learning Rate	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
0.00005	0.0579	0.3658	18.8938	0.1597	0.4208	0.1378	0.4000
0.00010	0.0524	0.4641	19.2278	0.1508	0.4167	0.1598	0.4236
0.00050	0.0211	0.8244	20.1207	0.1644	0.4394	0.1787	0.4260
0.00100	0.0104	0.9068	19.6227	0.1764	0.4350	0.1709	0.4299

Table 7: Step 7: Sequentially choosing the learning rate η.

Comment: $\eta = 0.0005$ is chosen. For $\eta = 0.001$, it suffers from overfitting.

Hyperparameter	Training			Validation		Testing	
Arctivation Function	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
ReLU	0.0219	0.8106	19.4774	0.1719	0.4388	0.1772	0.4402
Softmax	0.0633	0.2590	20.5452	0.1406	0.3809	0.1323	0.3780
Sigmoid	0.0566	0.3617	19.9385	0.1638	0.4364	0.1661	0.4480
Tanh	0.0404	0.6111	20.1756	0.1898	0.4772	0.1811	0.4661
Exponential	0.0249	0.7629	20.0849	0.1564	0.4191	0.1591	0.4102

Table 8: Step 8: Sequentially choosing the activation function.

Comment: tanh function is chosen for the activation function.

Hyperparameters		Training			Validation		Testing	
Batch Norm	Dropout Layer	Loss	Accuracy	Iter Time	AC1	AC2	AC1	AC2
No	**No**	0.0371	0.6593	19.5932	0.1861	0.4612	0.1795	0.4764
Yes	No	0.0407	0.5855	23.7974	0.1560	0.4189	0.1543	0.4268
No	Yes	0.0486	0.4800	21.2372	0.1743	0.4567	0.1811	0.4575
Yes	Yes	0.0514	0.4329	26.0050	0.1595	0.4267	0.1669	0.4276

Table 9: Step 9: Determining whether dropout layer and batch normalization should be applied.

Comment: Neither batch normalization nor drouput layer is applied, it indicates that there is no vanishing gradient problem within this dataset (Otherwise the performance of DNN should be better upon usage of batch normalization).

We implement the DNN with set of hyperparameters chosen above and the result is as follow:

1. Running the model again using final state's parameters:

2. The training time is 20.138571977615356s.

3. In validation, the average accuracy of predicting the first place is 0.18156491214460263.

4. In validation, the average accuracy of the prediction in the 1st-3rd place is 0.46179609014390444

5. In testing, the accuracy of predicting the first place is 0.18110236220472442.

6. In testing, The accuracy of the prediction in the 1st-3rd place is 0.4700787401574803.

Noticing that indeed for DNN with logistic output, the output is the estimated probability of each horse being rank 1, so we can predict the ranking or each horse according to those value, instead of just completing the relatively trivial task, i.e. only guessing the champion in a race. It is also a large advantage of DNN compare to other algorithm (eg: Random Forest as mentioned above). The following is the confusion matrix.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	R
1	42	13	16	8	5	4	7	4	5	2	4	3	0	4	0.36
2	11	26	17	9	12	5	8	4	6	5	6	5	1	5	0.22
3	11	5	41	7	7	6	7	13	6	2	4	13	3	4	0.32
4	15	15	16	18	10	6	4	7	6	2	7	5	0	2	0.16
5	12	10	7	10	15	8	13	4	7	4	3	5	0	5	0.15
6	17	8	10	5	6	10	8	4	7	2	5	3	4	3	0.11
7	12	12	14	10	7	3	17	3	1	2	0	3	2	7	0.18
8	12	8	6	7	7	5	8	11	5	6	3	4	3	6	0.12
9	16	12	10	4	4	2	6	6	9	3	8	7	2	0	0.10
10	12	11	8	6	10	6	7	4	2	12	3	4	2	3	0.13
11	10	13	7	9	6	1	6	3	3	5	8	4	0	6	0.10
12	15	4	10	5	9	2	2	1	5	4	3	12	1	4	0.16
13	5	6	2	5	4	1	3	2	1	0	3	3	7	3	0.16
14	5	2	1	1	5	4	1	0	3	1	0	2	3	2	0.07
P	0.22	0.18	0.25	0.17	0.14	0.16	0.18	0.17	0.14	0.24	0.14	0.16	0.25	0.04	

Table 10: Confusion Matrix

where P and R refer to precision and Recall respectively. Those blue colored are the correct prediction, and those red colored indicating there are more than 10 mis-specification. It is noticible that the wrong prediction are mainly made for the first three rank. It is probably because that the difference in quality of those participants being first three in the race is quite small, and hence the variability of those are especially large.

5.2 Further Extension

The result of DNN is much better than that of Random forest, giving approximately 50% accuracy on predicting a horse within top three rank, which is consistent to our intuition. Indeed, DNN is already a very powerful algorithm for prediction problem under some complicated structure. There are several way to improve the performance (Changing some details in DNN/ using other more algorithms)

1. Implement Recurrent Neural Network, which one of its main task is to label an ordered list of objects.

2. For the K-Fold Cross Validation, we can consider some random combinations of the hyperparameters **jointly** instead of sequentially so that we can obtain the optimal result.

3. It might be too computational intensive to implement the method mentioned above. Instead, one can consider applying the Genetic Algorithm, which introduce the concept of "Mutation" and "Cross-over" in selection of hyperparameter, which helps lower the computation burden to consider all combinations of hyperparameters.