1. Learning to Rank

Learning to rank or machine-learning rank is very important in the construction of information retrieval system. The following picture shows a general learning to rank framework.

In the picture, above, $q_1, q_2, q_3 ... q_n$ represents the search query. Each query has a set of associated documents, represented by feature vectors in this picture, labeled as $x_i^j$. Feature vectors reflect the relevance of the documents to the query. Typical feature vectors include the frequency of query terms etc. And $y_i^{(1)}, y_i^{(2)}, y_i^{(3)} ... y_i^{(n)}$ represents the preference label for each query document pair. The goal is to build a model to predict the ground truth label of test data as accurately as possible, in terms of loss function. Generally speaking, many ranking algorithms can be grouped into three approaches: the pointwise approach, the pairwise approach and the listwise approach. This project is to explore multiple ranking algorithms across these three kinds of approaches in terms of accuracy and efficiency.

2. Datasets

The dataset I will use in this project is “Yahoo! Learning to Rank Challenge”. This dataset consists of three subsets, which are training data, validation data and test data. The data format for each subset is shown as follows:[Chapelle and Chang, 2011]

```
<line> . =. <relevance> qid:<qid> <feature>:<value> ... <feature>:<value>
<relevance> . =. 0 | 1 | 2 | 3 | 4
<qid> . =. <positive integer>
<feature> . =. <positive integer>
<value> . =. <float>
```

Each line has three parts, relevance level, query and a feature vector. There are five levels of relevance from 0 (least relevant) to 4 (most relevant). Each query has a unique id and there are 699 features in this dataset labeled from 1 to 699. The value of each feature is normalized to a float in the [0,1] range.

3. Experiments Setup
3.1. Library and measurement function

In this project, I will use RankLib to do all the experiments. There are many ways to measure the performance of a ranking model, like ERR, DCG, NDCG and MAP. [Chapelle et al., 2009] I choose Normalized Discounted Cumulative Gain (NDCG) as the measure function, which measures the usefulness, or gain, of a document based on its position in the result list. The equations to compute NDCG are:

\[
NDCG_p = \frac{DCG_p}{IDCG_p}
\]

where:

\[
DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i)}
\]

\[
IDCG_p = \sum_{i=2}^{\vert REL\vert} \frac{2^{rel_i} - 1}{\log_2(i + 1)}
\]

3.2. Expanding missing data

The original dataset has 700 features. However, for each entry, it might not contain values for all features. The missing value of a feature is indicated as NaN. This fact will make the dimension reduction a little harder because dimension reduction cannot be applied to dataset containing unknown data. So, I need to expand the missing data by using the properties of existing data. To do this, I first read 200 features. Then I select features in which the proportion of NaN is no more than 50%. Because if the proportion is too high, the predicted data might not be accurate. The algorithm I am going to use is called Alternating Least Squares (ALS). This process can easily be done in MATLAB by using the following two lines of code:

```matlab
[coeff,score,latent,tsquared,explained,mu] = pca(y,'algorithm','als');
t = score*coeff' + repmat(mu,number of entries,1);
```

From these two pictures, we can see that after expansion, all NaN are replaced by predicted data and the existing data will remain the same.

3.3. Dimension reduction

Taking account of all the features in the dataset will be very time consuming, therefore, it will be necessary to reduce the dimension of the dataset. I use PCA to do the dimension reduction on the
expanded data. The first 20 features with the largest variability will be chosen. Before PCA, it is necessary to standardize the data. I use Z-score standardization to do that, which can easily be realized in MATLAB by using the following command:

\[
X_{\text{train}} = \text{zscore}(X_{\text{train}});
\]

### 3.4. k-fold cross-validation

In this project, 10-fold cross-validation will be used to do parameter tuning. For each ranking algorithm, I will first decide its major parameters and the range within which parameters can change. Then, a 10-fold cross-validation will be processed on each set of parameters. The whole set will be split into two sets, training set (90%) and validation set (10%). I first do PCA to the training set, the output of PCA will then be used to project the validation set. This process will be repeated ten times and the mean and standard deviation of NDCG will be returned. Considering the running time, I only use the first 2000 data of the whole dataset.

### 3.5. Parameters tuning

Based on the introduction of each algorithm, I will choose its hyper parameters. There are several ways to find the best parameters like grid search or random search. But a direct parameter tuning will be very time consuming because I have no idea in which regions the best parameters should be. So, I will first do single parameter tuning based on cross-validation. Then I will choose some possible good values for each parameter and do grid search, that is to test all pairs of these possible values. The best parameters will be determined by using bias-variance tradeoff. To clearly show the bias and variance, I will draw a normal distribution curve for each set of parameters by using the corresponding mean and variance of NDCG.

### 3.6. Comparison among ranking algorithms

After getting the best parameters, I will compare the performance among algorithms under those parameters by testing them on the same unseen test data. I will also gradually increase the size of training set to see how the ranking accuracy will change. I will also use box plot to analyze the sensitivity of different algorithms in the dataset dimension.

## 4. Parameter tuning and k-fold cross-validation

### 4.1. RankNet

#### 4.1.1. Introduction

RankNet is a pairwise ranking algorithm, which means its loss function is defined on a pair of documents or urls. For a given query, each pair of documents or urls \( \text{U}_i, \text{U}_j \) with different relevance level will be chosen. And let \( y_i, y_j \) be the computed label from a ranking model. Let \( \text{U}_i \triangleright \text{U}_j \) denotes the event that \( y_i > y_j \). Then a posterior is defined as:

\[
P_{ij} = P(\text{U}_i \triangleright \text{U}_j) = \frac{1}{1 + e^{-\left(y_i-y_j\right)}}
\]

Then, define the cost function of this pair to be:

\[
C_{ij} = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log (1 - P_{ij})
\]

\( \bar{P}_{ij} \) is the known probability that the ranking of \( \text{U}_i \) is greater than \( \text{U}_j \). Let \( S_{ij} \) to be defined to be 1 if \( \text{U}_i \) is labeled more relevant, -1 if \( \text{U}_j \) is labeled more relevant and 0 when \( \text{U}_i \) and \( \text{U}_j \) are labelled the same relevance. Then \( \bar{P}_{ij} \) can be defined as:

\[
\bar{P}_{ij} = \frac{1}{2} (1 + S_{ij})
\]

Therefore, the cost function will become:
$$C_{ij} = \frac{1}{2}(1 - S_{ij})(y_i - y_j) + \log (1 + e^{-(y_i - y_j)})$$

It is easy to see that cost function is larger when $U_i \triangleright U_j$ while $U_j$ is actually known to be more relevant. And the idea of learning via gradient descent is to update the weights (model parameters) with the gradient of the cost function.

$$w_k \rightarrow w_k - \eta \left( \frac{\partial C_{ij}}{\partial y_i} \frac{\partial y_i}{\partial w_k} + \frac{\partial C_{ij}}{\partial y_j} \frac{\partial y_j}{\partial w_k} \right)$$

where $\eta$ is a positive learning rate. Then a neural network will be used to optimize the ranking model to minimize the cost function with the back-prop equations.[Orr and Müller, 2003] Typically, the accuracy of the trained model will be affected by learning rate, the number of hidden layers and the number of hidden nodes per layer.

### 4.1.2. Single parameter tuning

<table>
<thead>
<tr>
<th>RankNet (learning rate)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>log(learning rate)</strong></td>
<td>training</td>
</tr>
<tr>
<td>-14</td>
<td>0.25</td>
</tr>
<tr>
<td>-13</td>
<td>0.30</td>
</tr>
<tr>
<td>-12</td>
<td>0.35</td>
</tr>
<tr>
<td>-11</td>
<td>0.40</td>
</tr>
<tr>
<td>-10</td>
<td>0.45</td>
</tr>
<tr>
<td>-9</td>
<td>0.50</td>
</tr>
<tr>
<td>-8</td>
<td>0.55</td>
</tr>
<tr>
<td>-7</td>
<td>0.60</td>
</tr>
<tr>
<td>-6</td>
<td>0.65</td>
</tr>
<tr>
<td><strong>mean of (1-NDCG@10)</strong></td>
<td>validation</td>
</tr>
<tr>
<td>0.23</td>
<td>0.24</td>
</tr>
<tr>
<td>0.24</td>
<td>0.25</td>
</tr>
<tr>
<td>0.25</td>
<td>0.26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RankNet (number of hidden layers)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>number of layers</strong></td>
<td>training</td>
</tr>
<tr>
<td>1</td>
<td>0.235</td>
</tr>
<tr>
<td>2</td>
<td>0.24</td>
</tr>
<tr>
<td>3</td>
<td>0.245</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
</tr>
<tr>
<td><strong>mean of (1-NDCG@10)</strong></td>
<td>validation</td>
</tr>
<tr>
<td>0.235</td>
<td>0.24</td>
</tr>
<tr>
<td>0.24</td>
<td>0.245</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RankNet (number of nodes per layer)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>number of nodes per layer</strong></td>
<td>training</td>
</tr>
<tr>
<td>5</td>
<td>0.23</td>
</tr>
<tr>
<td>10</td>
<td>0.235</td>
</tr>
<tr>
<td>15</td>
<td>0.24</td>
</tr>
<tr>
<td>20</td>
<td>0.245</td>
</tr>
<tr>
<td>25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

### 4.1.3. Grid search

The possible good value I choose based on the results above are listed below.
This picture shows the bias and variance of all pairs of parameters. We can see that the variance and bias cannot be the smallest simultaneously. I finally choose (3,25,0.0016) as the best parameters set because it can give us the smallest bias, and the variance is not that large.

4.2. MART

4.2.1. Introduction

Multiple Additive Regression Trees (MART) is an algorithm of using gradient boosted decision trees for prediction tasks.[Friedman, 2001] Its output function $F_N(x)$ is a linear combination of a set of regression trees.

$$F_N(x) = \sum_{i=1}^{N} \alpha_i f_i(x)$$

where $f_i(x)$ is the output function of a single regression tree and $\alpha_i$ is its weight. Then viewing the cost $C$ as a function of the output function.

$$C = C_0 + \frac{\partial C}{\partial F} \delta F$$

To reduce the cost $C$, $\delta F \propto -\eta \frac{\partial C}{\partial F}$ for a suitable step size, which is then used as the weight. The step size for the $j$th leaf node of the $m$th tree is:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} \log (1 + e^{-2y_i(F_{m-1}(x_i) + \gamma)})$$

The accuracy of MART is affected by the number of regression trees, the number of nodes and learning rate.[Wu et al., 2010]

4.2.2. Single parameter tuning
To avoid over-fitting, the learning rate is suggested no more than 0.1. [Li et al., 2007]

4.2.3. Grid search

The possible good value I choose based on the results above are listed below.

<table>
<thead>
<tr>
<th></th>
<th>500</th>
<th>700</th>
<th>1800</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trees</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leaves for each tree</td>
<td>13</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.06</td>
<td>0.09</td>
<td></td>
</tr>
</tbody>
</table>
This picture shows the bias and variance of all pairs of parameters. We can see that the variance and bias cannot be the smallest simultaneously. Therefore, I choose $(700, 18, 0.06)$ as the best parameters because it has the smallest bias and a quite small variance.

4.3. LamdaMART

4.3.1. Introduction

A new algorithm called LamdaRank can be achieved by directly computing the gradients instead of deriving them from a cost function. After the gradients, denoted as $\lambda_{ij}$, are calculated, there will exist an imagined cost function $C_{ij}$ such that:

$$
\lambda_{ij} = \frac{\partial C_{ij}(y_i - y_j)}{\partial y_i}
$$

And the weights update will become:

$$
w_k \rightarrow w_k + \eta \frac{\partial C_{ij}}{\partial w_k}
$$

LamdaMART is an algorithm that combines LamdaRank and MART. For each pair of $U_i, U_j$ with $U_i \triangleright U_j$, the gradients are defined as:

$$
\lambda_{ij} = \frac{-|\Delta Z_{ij}|}{1 + e^{y_i - y_j}}
$$

where $Z_{ij}$ means the difference of cost function $C_{ij}$ by swapping the scores of $U_i, U_j$. And the cost function will become:

$$
C = \sum_{(i,j) \neq l} |\Delta Z_{ij}| \log (1 + e^{-e^{y_i - y_j}})
$$

And the step size can be computed by using Newton approximation. Similarly, the accuracy of LamdaMART also depends on the number of regression trees, the number of nodes and learning rate.

4.3.2. Single parameter tuning
4.3.3. Grid search

The possible good value I choose based on the results above are listed below.

<table>
<thead>
<tr>
<th></th>
<th>Trees</th>
<th>Leaves</th>
<th>Learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1900</td>
<td>5</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.09</td>
</tr>
</tbody>
</table>
This picture shows the bias and variance of all pairs of parameters. We can see that the variance and bias cannot be the smallest simultaneously. Since the variance overall is not so large, I choose (1900, 14, 0.03) as the best parameters because it has the smallest bias.

4.4. RankBoost

RankBoost is a pairwise ranking algorithm. Its idea is to transfer the ranking problem as a problem of binary classification on instance pairs, and then to adopt boosting approach. It will train one weak ranker at each iteration and then combine these rankers as the final ranking function. The hyper-parameter I choose for RankBoost is the number of threshold candidates to search. Since the total number of features after reduction is 20, the threshold should also be no more than 20. The results below show that 13 will be a good value for the threshold.

4.5. ListNet

ListNet is a list wise ranking algorithm by optimizing the list wise loss function based on top one probability, with Network as model and Gradient Descent as optimization algorithm. The learning rate is chosen as the hyper parameter. The results below show that a good learning rate is 0.00001.
4.6. Random Forests

4.6.1. Introduction
Random forests is an ensemble learning method for classification, regression and other tasks. It constructs a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees. The hyper parameters I choose for random forests include the number of bags, the number of trees in each bag, the number of leaves for each tree and the learning rate.

4.6.2. Single parameter tuning
4.6.3. Grid search

The possible good value I choose based on the results above are listed below.

<table>
<thead>
<tr>
<th>Bags</th>
<th>220</th>
<th>420</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trees in each bag</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Leaves</td>
<td>60</td>
<td>240</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.025</td>
<td>1.6</td>
</tr>
</tbody>
</table>

From this figure, we can find that the best parameter set is (220, 7, 60, 1.6).

5. Comparison of ranking algorithms
5.1. Accuracy comparison

By using the results from parameter tuning, I can now test the performance of these algorithms on some unseen testing data. I will gradually increase the training size to see how their performance will change.

From these two figures, we can find that RankNet and ListNet are the two worst ranking algorithms in both ranking accuracy and the running time. If we only consider the accuracy, Random Forests is the best algorithm. If we also consider the running time, MART and Random Forests are the two best algorithms.

5.2. Sensitivity analysis

The process is very like cross-validation. I first divide the whole dataset into 20 subsets. Then I leave one subset out and train the model on the remaining 19 subsets. Then I use the trained model to calculate the NDCG@10 and MAP on the same test data. Finally, I will use box plots to analyze the results.

From these two pictures, we can find that LambdaMART and ListNet have smaller sensitivity because they have smaller IQR. But based on the position of box, ListNet has pretty bad performance. On the other hand, the data of Random Forests and MART shows not only good performance but also quite centralized. Therefore, I think they are the best two ranking algorithms.

6. Reference

2) Chapelle, O., D. Metzler, Y. Zhang, and P. Grimspan (2009), Expected reciprocal rank for
graded relevance, paper presented at Proceedings of the 18th ACM conference on Information and knowledge management, ACM.


