SOME TOPICS IN MODELING RANKING DATA

by

FANG QI

A thesis submitted in partial fulfillment of the requirements for

the Degree of Doctor of Philosophy

at The University of Hong Kong.

April 2014

Abstract of thesis entitled

SOME TOPICS IN MODELING RANKING DATA

Submitted by

FANG QI

for the degree of Doctor of Philosophy

at The University of Hong Kong

in April 2014

Many applications of analysis of ranking data arise from different fields of study, such as psychology, economics, and politics. Over the past decade, many ranking data models have been proposed. AdaBoost is proved to be a very successful technique to generate a stronger classifier from weak ones; it can be viewed as a forward stagewise additive modeling using the exponential loss function. Motivated by this, a new AdaBoost algorithm is developed for ranking data. Taking into consideration the ordinal structure of the ranking data, I propose measures based on the Spearman/Kendall distance to evaluate classifier instead of the usual misclassification rate. Some ranking datasets are tested by the new algorithm, and the results show that the new algorithm outperforms traditional algorithms.

The distance-based model assumes that the probability of observing a ranking depends on the distance between the ranking and its central ranking. Prediction of ranking data can be made by combining distance-based model with the famous k-nearest-neighbor(kNN) method. This model can be improved by assigning weights to the neighbors according to their distances to the central ranking and assigning weights to the features according to their relative importance. For the feature weighting part, a revised version of the traditional ReliefF algorithm is proposed. From the experimental results we can see that the new algorithm is more suitable for ranking data problem.

Error-correcting output codes (ECOC) is widely used in solving multi-class learning problems by decomposing the multi-class problem into several binary classification problems. Several ECOCs for ranking data are proposed and tested. By combining these ECOCs and some traditional binary classifiers, a predictive model for ranking data with high accuracy can be made.

While the mixture of factor analyzers (MFA) is useful tool for analyzing heterogeneous data, it cannot be directly used for ranking data due to the special discrete ordinal structures of rankings. I fill in this gap by extending MFA to accommodate for complete and incomplete/partial ranking data. Both simulated and real examples are studied to illustrate the effectiveness of the proposed MFA methods.

Declaration

I declare that this thesis represents my own work except that Chapter 5 is a joint work with Dr. Jianhua Zhao from Yunan University of Finance and Economics. All materials in this thesis have not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualification.

Signed Qi Fang

Fang Qi

Acknowledgements

Foremost, I would like to express my most sincere gratitude to my supervisor, Dr. Philip Leung-Ho Yu, for his full support and excellent supervision during my graduate study in the University of Hong Kong. Without his incredible encouragement, precious suggestions and valuable guidance, this thesis could not been completed.

Also, many thanks go to Dr. Jianhua Zhao from Yunan University of Finance and Economics. Dr. Zhao gave me a lot of valuable suggestions and shared his experience in implementing mixture of factor analyzers. Lastly, I would like to thank my parents and my wife for their support during my Ph.D. study.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Statistical models for ranking data

Ranking data occur when judges (individuals) are asked to rank a set of items by their own preference. For example, in a political election, citizens rank a few candidates in order of their preference. Before a clinical computerization system is deployed, doctors might be asked to rank a list of barriers/incentives according to their self-perceived importance. In marketing surveys, consumers might be asked to rank a set of items, such as sushi, video games, movies or jobs.

An important task in the analysis of ranking data is to develop statistical models for ranking data. We might develop models to describe similarity or distribution for ranking data (unsupervised learning). And more often, we develop models to predict rankings for new judges (supervised learning).

The applications of analyzing ranking data cover many field of studies, such as politics and elections (Croon, 1989; Stern, 1993; Skrondal and Rabe-Hesketh, 2003; Murphy and Martin, 2003; Vermunt, 2004; Moors and Vermunt, 2007; Gormley and Murphy, 2008), psychology (Riketta and Vonjahr, 1999; Decarlo and Luthar, 2000; Bockenholt, 2001; Maydeu-Olivares and Bockenholt, 2005;

Regenwetter et al., 2007), market research (Beggs et al., 1981; Chapman and Staelin, 1982; Dittrich et al., 2000), food preference (Nombekela et al., 1993; Kamishima and Akaho, 2006), and health-care (Salomon, 2003; McCabe et al., 2006; Ratcliffe et al., 2006; Krabbe et al., 2007; Craig et al., 2009).

Suppose a group of judges (individuals) are asked to rank J items, labeled $1, \cdots, J$. A ranking of these J items, denoted by $\boldsymbol{\pi}$, is defined as a permutation of 1 to J. We denote $\pi(i)$ the rank of item i, and $\pi^{-1}(i)$ the item whose rank is i. For example, if $J = 4$ and a judge prefers item 2 the most, item 4 the second, item 1 the third, and item 3 the least. Then the ranking $\boldsymbol{\pi} = (3, 1, 4, 2)$, where $\pi(1) = 3, \pi(2) = 1, \pi(3) = 4, \pi(4) = 2$; and the inverse ranking $\boldsymbol{\pi}^{-1} = (2, 4, 1, 3)$, where $\pi^{-1}(1) = 2, \pi^{-1}(2) = 4, \pi^{-1}(3) = 1, \pi^{-1}(4) = 3$. Sometimes people call $\boldsymbol{\pi}^{-1}$ the ordering of items, which represents the preference ordering of J items for a judge.

Sometimes judges do not rank all J items, they only provide ranking on their most favorite q items ($q < J$). We call this partial ranking (top-q ranking). Partial ranking is more common than full ranking in survey studies. For a q-of-J partial ranking, judges only give their top-q favorite items, so $\pi(i)$ sometimes does not take any value, $\pi^{-1}(i)$ takes a value when $i \leq q$ and does not take any value when $i > q$. Some researches about partial ranking can be found in Critchlow (1985), Alvo and Cabilio (1991).

1.2 Introduction to traditional statistical models for ranking data

During the last decade, a lot of ranking data models have been proposed. They cover most of the fields in statistics and data mining, we categorize them into several classes and discuss them in the following subsections.

1.2.1 Order statistics models

The most traditional statistical ranking data model is the order statistics model proposed by Thurstone (1927). If N judges are asked to rank J items, then J random utilities $V_{nj}, 1 \leq n \leq N, 1 \leq j \leq J$ will be assigned by each judge. Each random utility V_{nj} is defined as:

$$V_{nj} = U_{nj} + \varepsilon_{nj}, \tag{1.2.1.1}$$

where U_{nj} is the expected utility to item j determined by judge n, and ε_{nj} are i.i.d. random errors.

Under this definition, the probability of observing ranking $\boldsymbol{\pi}$ is:

$$\Pr(\boldsymbol{\pi}) = \Pr(V_{n\pi^{-1}(1)} > V_{n\pi^{-1}(2)} > \cdots > V_{n\pi^{-1}(J)}). \tag{1.2.1.2}$$

The original order statistics model assumed that the error terms follow the normal distribution. Luce (1959) suggested that the error terms follow an extreme value distribution. Under Luce's assumption, the probability of observing a ranking $\boldsymbol{\pi}$ becomes:

$$\prod_{j=1}^{J} \frac{\exp(U_{\pi^{-1}(j)})}{\sum_{i=j}^{J} \exp(U_{\pi^{-1}(i)})}. \tag{1.2.1.3}$$

Such model is often called Plackett-Luce model. The Plackett-Luce model can be extended by allowing covariates to explain the utilities. If K covariates are included from judge n, the random utilities will become:

$$U_{nj} = \beta_{j0} + \sum_{k=1}^{K} \beta_{jk} x_{nk}, \tag{1.2.1.4}$$

where $\beta_{jk}, 0 \leq k \leq K$ are parameters of K covariates for item j, and $x_{nk}, 0 \leq k \leq K$ are the K covariates for judge n. This extended Plackett-Luce model is known as the rank-ordered-logit (ROL) model (Beggs et al., 1981).

The log-likelihood functions for both Plackett-Luce and rank-ordered-logit models are globally concave, so the estimation of the model parameters can be solved by traditional optimization method such as Newton-Raphson. After all parameters $\beta_{jk}, 0 \leq k \leq K$, have been estimated, the rank-ordered-logit model can be used to predict the ranking for a new judge by sorting the J predicted utilities from (1.2.1.4). Applications of rank-ordered-logit models can be found from Hausman and Ruud (1987); Allison and Christakis (1994); Koop and Poirier (1994); van Dijk et al. (2007).

Classical order statistics models assume that all utilities V_{nj} are independent to each other. However, some of these utilities are usually correlated. To tackle this problem, Yu (2000) and Yao and Bockenholt (1999) extended the order statistics models by assuming that the utility vector $\mathbf{V} = (V_1, \cdots, V_J)$ follows a multivariate normal distribution.

1.2.2 Distance-based models

Distance-based model assumes that the probability of observing a ranking is inversely proportional to its distance to a central ranking. Before giving details of the distance-based model, the distance between rankings need to be defined first.

A distance $d(\cdot, \cdot)$ between two ranking $\boldsymbol{\pi_1}$ and $\boldsymbol{\pi_2}$ should satisfy the following three properties:

- $d(\boldsymbol{\pi_1}, \boldsymbol{\pi_1}) = 0$,
- $d(\boldsymbol{\pi_1}, \boldsymbol{\pi_2}) > 0$ if $\boldsymbol{\pi_1} \neq \boldsymbol{\pi_2}$,
- $d(\boldsymbol{\pi_1}, \boldsymbol{\pi_2}) = d(\boldsymbol{\pi_2}, \boldsymbol{\pi_1})$.

For ranking data, we always require that the distance should be right invariant, i.e. $d(\boldsymbol{\pi_1}, \boldsymbol{\pi_2}) = d(\boldsymbol{\pi_1} \circ \boldsymbol{\tau}, \boldsymbol{\pi_2} \circ \boldsymbol{\tau})$, where $\boldsymbol{\tau}$ is any ranking. We list some popular and widely used distances for ranking data below.

Square root Spearman distance:

$$d_{SS}(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \left(\sum_{i=1}^{J} [\pi_1(i) - \pi_2(i)]^2 \right)^{\frac{1}{2}}. \qquad (1.2.2.1)$$

Spearman distance:

$$d_S(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \sum_{i=1}^{J} [\pi_1(i) - \pi_2(i)]^2. \tag{1.2.2.2}$$

Spearman's footrule distance:

$$d_F(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \sum_{i=1}^{J} |\pi_1(i) - \pi_2(i)|. \tag{1.2.2.3}$$

Kendall distance:

$$d_K(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \sum_{i<j} \boldsymbol{I}[\pi_1(i) - \pi_1(j)][\pi_2(i) - \pi_2(j)] < 0, \tag{1.2.2.4}$$

where $\boldsymbol{I}(\cdot)$ is an indicator function. More distances can be obtained from Critchlow et al. (1991).

Diaconis (1988) first proposed the class of distance-based models:

$$P(\boldsymbol{\pi}|\lambda, \boldsymbol{\pi}_0) = \frac{\exp(-\lambda d(\boldsymbol{\pi}, \boldsymbol{\pi}_0))}{C(\lambda)}. \tag{1.2.2.5}$$

In this model, $\boldsymbol{\pi}_0$ is the central ranking which represents the most probable ranking. $\lambda(\geq 0)$ is the dispersion parameter, who governs the speed of decay in ranking probability $P(\boldsymbol{\pi})$ when $\boldsymbol{\pi}$ moves away from $\boldsymbol{\pi}_0$. $C(\lambda)$ is the proportionality constant such that the sum of all $J!$ ranking probabilities is one.

When Kendall distance is used as the distance function in (1.2.2.5), the distance-based model is called the Mallows ϕ-model (Mallows, 1957). Critchlow and Verducci (1992) and Feigin (1993) showed that the Mallows ϕ-model is a special case of paired comparison models.

In the distance-based model, the proportionality constant $C(\lambda)$ can be computed by going through all possible rankings. However, there will be $J!$ possible rankings for J items. When J increases, $J!$ will increases much faster (for example, $10! = 3,628,800$ and $15! = 1,307,674,368,000$). Then it will be very time consuming to compute $C(\lambda)$. Fortunately, if we use Kendall distance as the ranking distance, a closed form solution for $C(\lambda)$ exists:

$$C(\lambda) = \prod_{j=1}^{J} \frac{1 - \exp(-j\lambda)}{1 - \exp(-\lambda)}. \tag{1.2.2.6}$$

More details can be found in Fligner and Verducci (1988). Another important parameter in distance-based model is the central ranking $\boldsymbol{\pi}_0$. When we know a set of rankings $\{\boldsymbol{\pi}_1, \cdots, \boldsymbol{\pi}_n\}$, it is easy to verify that the MLE of the central ranking should be:

$$\widehat{\boldsymbol{\pi}}_0 = \arg\min_{\pi} \sum_{n=1}^{N} d(\boldsymbol{\pi}_n, \boldsymbol{\pi}). \tag{1.2.2.7}$$

Similar to the process of finding constant parameter, a thoroughly search for all possible rankings is needed to find the MLE $\widehat{\boldsymbol{\pi}}_0$. It is not practical when J is large. Busse et al. (2007) proposed a local search algorithm. The algorithm uses iteration method; starts from an arbitrary initial ranking. In each step, the algorithm searches all the rankings who are close to the initial ranking. This can be done by finding rankings whose Kendall distance to initial ranking equals one. And all the nearest rankings will be tested in (1.2.2.7), the best ranking will replace the initial ranking. The process will continue until the initial ranking does not change.

The population of judges is more likely to be heterogeneous than homogeneous. In this case, it is difficult to find a good central ranking. Mixture model can solve this problem by assuming that the population is composed of several homogeneous sub-groups. The distribution within each sub-group can be modeled by a distance-based model. Murphy and Martin (2003) applied mixture model to distance-based model and explained the heterogeneity among judges. Lee and Yu (2012) extended distance-based model to weighted distance-based model which can give a better fit for some ranking data.

1.2.3 Pairwise comparison models

Hüllermeier et al. (2008) proposed solving ranking data problems by pairwise comparison models. If J items will be ranked, $J(J-1)/2$ binary classifiers will be built.

For each item pair (a_i, a_j), a binary classifier will be built independently. For any new individual, the binary classifier will predict the pairwise relation between a_i and a_j: whether $a_i \succ a_j$ or $a_j \succ a_i$? The binary classifier can also predict the probability that $a_i \succ a_j$, denoted by $R(a_i, a_j)$. After the relation of $J(J-1)/2$ pairs have been predicted, a voting strategy can be used to predict the full ranking. The voting strategy will compute the evaluation value of each item:

$$S_i = \sum_{a_j \neq a_i} R(a_i, a_j). \tag{1.2.3.1}$$

Then the predict ranking can be determined by the ordering of $(S_1, S_2, \cdots, S_J)$. In another words, the following relationship should be satisfied:

$$(a_i \succeq a_j) \Leftrightarrow S_i \geq S_j. \tag{1.2.3.2}$$

Hüllermeier et al. (2008) proved that under the assumption that all pairwise comparison models provide the correct predictions, the model yields a risk minimizing prediction (in the sense of Spearman distance).

1.2.4 Decision tree models

Decision tree is a nonparametric statistical model for classification and prediction. Decision tree is a flow-chart like structure in which internal node represents test on a covariate, each branch represents outcome of test and each leaf node represents class label. A path from root to leaf represents classification rules. Decision tree models are widely used in data mining and machine learning because they have the following advantages:

- Easy interpretation (not black-box like neutral network).
- Handle both numerical and categorical covariates.
- Similar to the thinking process of human beings.

- Automatic feature selection.

Many variants of decision tree model have been proposed, such as C4.5 (Quinlan, 1992), CART (Breiman et al., 1984), FACT (Loh and Vanichsetakul, 1988) and QUEST (Loh and Shih, 1997).

1.2.4.1 CART

Among different kinds of decision tree models, the classification and regression tree (CART) (Breiman et al., 1984) is the most popular one. The process of building a tree using CART consists of two stages: tree growing and tree pruning.

The growing stage starts from the root node, which contains all the training data. Then the root node is spitted into two parts, called the child nodes. In order to find the best split of a model, an impurity function for a node need to be defined first. The impurity function should satisfy the following three properties:

- The function reaches to its minimum when the node is pure, meaning that all the data in this node belongs to one class.
- The function reaches to its maximum when the node is most impure, meaning that the data from all classes are uniformly distributed in this node.
- Changing the class labels does not change the value of the impurity function.

Misclassification Error, Gini and Entropy are impurity functions which people use mostly. We assume $p_j, 1 \leq j \leq J$ are the proportion of J classes in one node, then these three impurity functions are:

- Misclassification Error: $1 - \max_j p_j$.
- Gini: $1 - \sum_{j=1}^{J} p_j^2$.
- Entropy: $-\sum_{j=1}^{J} p_j \log_2 p_j$.

It is trivial to prove that the above three functions satisfy the impurity properties.

The model will search all possible covariates and all possible splitting values to find the best splitting, and then two child nodes will be obtained. The splitting process will be applied to these two child nodes recursively and a divide and conquer, top-down structured decision tree model is built.

The growing process will continue until a stopping criterion is met. We can build a tree in which all the leaf nodes are pure. In this case, some leaf node might only contain one observation, and the size of the tree will be very large. This will lead to the problem of over-fitting. Over-fitting means that the model fits training data very well, however it has a very poor performance on testing data. To solve this problem, pruning is necessary.

In the pruning stage, the tree is trimmed into a suitable size for better interpretation and avoiding the over-fitting problem. There are many pruning methods, for example: the cost-complexity pruning for CART (Breiman et al., 1984), reduced error pruning and pessimistic error pruning (Quinlan, 1987), minimum error pruning (Niblett and Bratko, 1986), critical value pruning (Mingers, 1987), error-based pruning (Quinlan, 1992).

1.2.4.2 Decision tree models for ranking data

Yu et al. (2009) proposed modeling ranking data by decision tree models. Since ranking data have their special structure, the impurity function for ranking data need to be defined first. Yu et al. (2009) introduced two measures for ranking data, namely n-wised measures and top-k measures.

In n-wised measure($n \leq J$), $p_w(i_1, i_2, \cdots, i_n)$ is the proportion of data which satisfy $i_1 \succ i_2 \succ \cdots \succ i_n$. Items not in $i_1, i_2, \cdots, i_n$ are not considered. For example, $p_w(2, 3)$ is the proportion of data who rank item 2 higher then item 3.

In top-k measure, $p_t(i_1, i_2, \ldots, i_k)$ is the proportion of data in which item i_1 ranks 1st, item i_2 ranks 2nd, and item i_k ranks k-th. For example, $p_t(2, 4, 3)$ is the proportion of data where item 2 ranks 1st, item 4 ranks 2nd and item 3 ranks 3rd.

The Gini and Entropy impurity functions for ranking data can be derived based on these two measures. For n-wised measure of ranking k items, $R_{k,n}$ denotes the set of all k-choose-n permutations, $R^*_{k,n}$ denotes the set of all k-choose-n combinations, $R_{k,n}|i_1, i_2, \ldots, i_n$ denotes the set of all $n!$ combinations from $i_1, i_2, \ldots, i_n$. For example, when $k = 4$ and $n = 2$,

- $R_{4,2} = [(1,2),(2,1),(1,3),(3,1),(1,4),(4,1),(2,3),(3,2),(2,4),(4,2),(3,4),(4,3)]$
- $R^*_{4,2} = [(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)]$
- $R_{4,2}|1,2 = [(1,2),(2,1)]$

Based on the above definitions, the Gini for n-wised measure is:

$$\text{Gini}_w = \frac{1}{C^k_n} \sum_{B \in R^*(n,k)} \left[1 - \sum_{r \in R(n,k)|B} p_r \right], \tag{1.2.4.1}$$

the Gini for top-k measure is:

$$\text{Gini}_t = 1 - \sum_{r \in R(n,k)} p_r, \tag{1.2.4.2}$$

the Entropy for n-wised measure is:

$$\text{Entropy}_w = \frac{1}{C^k_n} \sum_{B \in R^*(n,k)} \left[1 - \sum_{r \in R(n,k)|B} p_r \log_2 p_r \right], \tag{1.2.4.3}$$

the Entropy for top-k measure is:

$$\text{Entropy}_t = 1 - \sum_{r \in R(n,k)} p_r \log_2 p_r. \tag{1.2.4.4}$$

After the impurity functions for ranking data are defined, a decision tree model can be built. And the ranking can be predicted by aggregating the rankings of all training examples available in that leaf node. Yu and Lee (2010) proposed modeling ranking data by combining distance-based model and decision tree model. This model uses a recursive partition algorithm to build a tree, and a distance-based model is built at each leaf node.

1.2.5 Factor analysis models

Factor analysis is a powerful tool to identify common characteristics among some variables. Factor analysis is widely used in marketing research and social science. The classical d-factor model (Thurstone, 1947) is defined as:

$$x_{i,j} = \mathbf{z}_i^T \mathbf{a}_j + \varepsilon_{i,j}, \tag{1.2.5.1}$$

where $\boldsymbol{x}_i = (x_{i1}, \cdots, x_{ik})^T$ is the response vector for the i-th instance, $\boldsymbol{z}_i = (z_{i1}, \cdots, z_{id})^T$ is the unobserved common factors for the i-th instance, $\boldsymbol{a}_j = (a_{1j}, \cdots, a_{dj})^T$ is the factor loading for the j-th item, and $\varepsilon_{i,j}$ is the error term.

Yu et al. (2005) proposed analyzing ranking data by factor models. They assumed that the ranking of J items by judge i is determined by the ordering of J latent utilities $x_{i,1}, \cdots, x_{i,J}$, and these utilities satisfy a general version of d-factor model:

$$x_{i,j} = \mathbf{z}_i^T \mathbf{a}_j + b_j + \varepsilon_{i,j}, \tag{1.2.5.2}$$

where $\boldsymbol{b} = (b_1, \cdots, b_J)^T$ is the mean vector for utilities, which reflect the importance of J items for all judges. $\mathbf{a}$ is still the factor loading, and the factors $\mathbf{z}_1, \cdots, \mathbf{z}_n$ follow i.i.d. standard normal distribution.

Let $\boldsymbol{r}_i = (r_{i,1}, \cdots, r_{i,J})^T$ be the ranking vector, and $r_{i,j}$ is the rank of item j for judge i. Then the ranking vector $\mathbf{r}$ must be consistent with the utility vector $\mathbf{x}$. This constraint makes the factor models for ranking data more difficult to be fitted than the traditional factor models. Yu et al. (2005) solved this problem by implementing the Gibbs sampling method in the E-step of the EM algorithm (Dempster et al., 1977). After a factor model for ranking data has been built, we can estimate the unobserved factors which can determine the ranking behaviors of the judges.

1.3 Organization of this thesis

This thesis is composed of two parts. The first part, Chapter 2 to Chapter 4, discusses three supervised learning models for ranking data. The main purpose of supervised learning models is to predict the rankings of judges. The performance of supervised learning models are often evaluated by the prediction accuracy. The second part, Chapter 5, is an unsupervised learning model for ranking data. The main purpose of unsupervised learning models is to understand the structure of rankings, such as clustering and common factor structures.

In Chapter 2, a model based on boosting algorithm is proposed to predict rankings. The AdaBoost model is an ensemble model which combines some weak classifiers to form a more powerful classifier. AdaBoost has been widely used in two-class and multi-class classification problems. I build an ensemble model for ranking data by applying the idea of boosting. Although the new model also decompose a ranking into pairs like the pairwise comparison model, it takes the ranking as a whole during boosting. It is better than modeling each pair separately. From the experimental results we can see that this model outperforms many traditional ranking models, including the pairwise comparison model.

In Chapter 3, The instance-based ranking model is enhanced by assigning weight to each covariate for judges. ReliefF is a widely used feature weighting algorithm. However, in ranking data problem, if each possible ranking is treated as a different class, the large number of classes will lead ReliefF fail to work. Considering the distance between rankings, a revised version of ReliefF is proposed for ranking data. Results show that after assigning weights to covariates by the new ReliefF algorithm, the new instance-based ranking model performs better.

In Chapter 4, The error-correcting output codes (ECOC) are used to built ranking data models. I proposed some ECOC coding schemes for ranking data. Each coding scheme will map a ranking into a binary vectors. I try these coding schemes with some popular binary classifiers, such as logistic regression, decision tree and bagging. The experimental results show that, by applying appropriate ECOC scheme for ranking and using bagging as the binary classifier, we can predict rankings with very high accuracy.

In Chapter 5, we extend the mixtures of factor analyzers (MFA) to accommodate for ranking data. The new methods are more realiable for analyzing heterogeneous data, and they can handle both complete and incomplete/partial ranking data. For all the methods, both simulated and real examples are evaluated to demonstrate the effectiveness of the proposed methods.

Chapter 2

Analyzing ranking data by boosting

2.1 Introduction

Learning ranking data is to induce models from empirical ranking data, and use the models to predict preferences for a new dataset. For example, how to predict the ordering of a set of items, such as the preferences of some political candidates based on the demographic status of a voter, or the preferences of a set of computer games according to player's characteristics.

Suppose a group of judges is asked to rank J items, labeled by $a_1, \ldots, a_J$. A ranking of these J items, denoted by $\boldsymbol{\pi}$, is defined as a permutation of 1 to J. We denote $\pi(i)$ the rank of item i. For example, when $J = 3$ and the preference ordering is: $(a_1 \succ a_3 \succ a_2)$, then $\pi(1) = 1, \pi(2) = 3, \pi(3) = 2$. The ranking data problem is: giving a judge's characteristics, how should we predict his/her ranking?

Over the past decade, many models have been proposed, a review of the literature can be found in Marden (1995); Yu (2003); Vembu and Gärtner (2010). The most traditional model is the order-statistics models (Thurstone, 1927), which assumes that a ranking of J items is determined by the ordering of J

random utilities, but computing the probability of the ordering needs a high dimensional integration which could be time consuming. An alternative approach is the distance-based models (Mallows, 1957), its basic idea is that, the closer a ranking $\boldsymbol{\pi}$ to its true ranking $\boldsymbol{\pi}_0$, the more likely $\boldsymbol{\pi}$ is observed. Another approach is based on pairwise comparison (David, 1988) such that the process is decomposed into a number of independent pairwise comparisons. However, the ranking probability under this model requires the computation of the proportionality constant which may not have a closed form.

Most of the ranking models assume a homogenous population and could not be used to make prediction of ranking outcomes. However, in the real world, heterogeneity always exists among different judges. The heterogeneity problem can be overcome by the decision tree models, which are nonparametric statistical methodology designed for classification and prediction. Decision tree technique is widely used in machine learning and data mining because of its ease of interpretability and its ability of handling variables of different measurement types.

Yu et al. (2009) show how decision tree algorithms such as CART (Breiman et al., 1984; Quinlan, 1986) can be adapted to ranking data by extending the impurity functions for ranking data. They modified the traditional impurity functions, Gini and Entropy, to ranking data, and predicted the ranking by aggregating the rankings of all training examples in each terminal node.

The number of possible rankings ($J!$) might be very large when J increases.

Making prediction by a single decision tree will encounter one problem: If the number of terminal nodes is less than the number of possible rankings, some rankings will never be predicted; If we increase the number of terminal nodes, the size of some terminal nodes may be too small, and it is possibly meaningless to make a prediction from such small samples. In this chapter, we consider building more trees and making predictions by combining them, which is the principle idea of boosting.

Boosting has been proved to be a very successful tool for solving the classification problem. It is an iterative procedure which will combine many weak classifiers into a stronger one. It was first proposed for two-class classification problem, and then extended to multi-class problem. In this chapter, we will develop efficient boosting algorithms to improve model accuracy.

The rest of this chapter is organized as follows: In Section 2.2, a review of the AdaBoost algorithm is given. In Section 2.3, we propose new boosting algorithms for ranking data and the procedure of justification will be presented. In Section 2.4, some ranking datasets will be tested on the new algorithms and the conclusion will be given in Section 2.5.

2.2 AdaBoost algorithm

2.2.1 Two-class AdaBoost

The AdaBoost algorithm (Freund and Schapire, 1997) aims to combine many weak classifiers to get a stronger one. First of all, a classifier is built on the unweighted training data, and the weight of each data point will be increased (boosted) if it is misclassified by the classifier. At the second step, a new classifier is built on the weights. We will repeat this procedure for many times. Finally the classifiers generated are combined to form a stronger classifier.

Suppose we are given a training dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where the input $\mathbf{x}_n \in \mathbf{X}$ ($\mathbf{X}$ is the feature space with dimension F) and output $y_n \in \{-1, +1\}$, the AdaBoost algorithm is given in Algorithm 1.

Algorithm 1 Pseudo-code of two-class AdaBoost algorithm.

```
1. Initialize the observation weight $w_n = 1/N$,$n = 1, 2, \ldots, N$.
2. for $m = 1$ to $M$ do
3. Fit a classifier $T^{(m)}(\mathbf{x}) : \mathbf{X} \to \{-1, +1\}$ to the training data using weight $w_n$.
4. Compute
   $$err^{(m)} = \sum_{n=1}^{N} w_n \mathbf{I}(y_n \neq T^{(m)}(\mathbf{x}_n)) / \sum_{n=1}^{N} w_n.$$
```

```
5. Choose
   $$\alpha^{(m)} = \frac{1}{2}\log\frac{1-err^{(m)}}{err^{(m)}}.$$
6. Update $w_n \leftarrow w_n \cdot \exp(-\alpha^{(m)} y_n T^{(m)}(\mathbf{x}_n))$ for $n = 1, \ldots, N$.
7. Re-normalize $w_n$, let $\sum_{n=1}^{N} w_n = 1$.
8. end for
9. Output the final prediction:
   $$H(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha^{(m)} T^{(m)}(\mathbf{x})\right)$$
```

Friedman et al. (2000) developed a statistical perspective of the original two-class AdaBoost algorithm, it views the AdaBoost algorithm as a forward stage-wise additive modeling using the exponential loss function:

$$L(y, f) = e^{yf(\mathbf{x})},$$

where $y \in \{-1, 1\}$ for two-class case and f is the classifier. AdaBoost has been proved to be excellent in producing accurate classifiers. Therefore, Breiman called AdaBoost with trees the "best off-the-shelf classifier in the world".

2.2.2 Multi-class AdaBoost

Many algorithms have been proposed to extend the two-class problem into multi-class problem. For instance, AdaBoost.M1, AdaBoost.MH, LogitBoost (Schapire and Singer, 1999; Friedman et al., 2000). Most of them are constructed by treating a multi-class problem as several two-class problems. However, they may not work if the weak classifiers are very weak. Recently, Zhu et al. (2009) proposed a new multi-class AdaBoost algorithm (SAMME). Zhu et al. (2009) showed that SAMME can be derived by minimizing a multi-class exponential loss function by a forward stagewise additive modeling algorithm, which only requires the weak classifiers perform better than random guess and the implementation is very simple and straightforward.

Given $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where the input $\mathbf{x}_n \in \mathbf{X}$ and output y_n is in a finite set $\{1, 2, \ldots, K\}$, the multi-class AdaBoost algorithm is given in Algorithm 2.

Algorithm 2. Pseudo-code of multi-class AdaBoost algorithm

```
1. Initialize the observation weight $w_n = 1/N$,$n = 1, 2, \ldots, N$.
2. for $m = 1$ to $M$ do
3. Fit a classifier $T^{(m)}(\mathbf{x}) : \mathbf{X} \to \{1, 2, \ldots, K\}$ to the training data using
   weight $w_i$.
4. Compute
   $$err^{(m)} = \sum_{n=1}^{N} w_n \mathbf{I}(y_n \neq T^{(m)}(\mathbf{x}_n)) / \sum_{n=1}^{N} w_n.$$
5. Choose
   $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1).$$
```

6. Update $w_n \leftarrow w_n \cdot \exp(-\alpha^{(m)} \cdot \mathbf{I}(y_n \neq T^{(m)}(\mathbf{x}_n))$ for $n = 1, \ldots, N$.

7. Re-normalize w_n, let $\sum_{n=1}^{N} w_n = 1$.

8. **end for**

9. Output the final prediction:

$$H(\mathbf{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbf{I}(T^{(m)}(\mathbf{x}) = k).$$

2.3 Boosting model for ranking data

2.3.1 Ranking data problem

In modeling ranking data, it is assumed that each observation $\mathbf{x}$ from $\mathbf{X}$ will be associated with a ranking $\boldsymbol{\pi}$ of J items, and we would like to develop a boosting algorithm that can be used to predict the ranking $\boldsymbol{\pi}$ for a new observation $\mathbf{x}$.

The most direct approach to tackle this problem is to take each possible ranking of J items as a different class, resulting in a $J!$-class classification problem. However, this approach has two drawbacks: first, it ignores the ordinal structure of ranking, and it will not distinguish whether the predicted ranking slightly or largely differs from the observed ranking. Such information should be very useful in enhancing the algorithm; Second, when the number of items (J) increases, the possible number of rankings ($J!$) will increase rapidly and make the classification difficult. Take the data Housing example from Section 2.4, there are $J = 6$ items

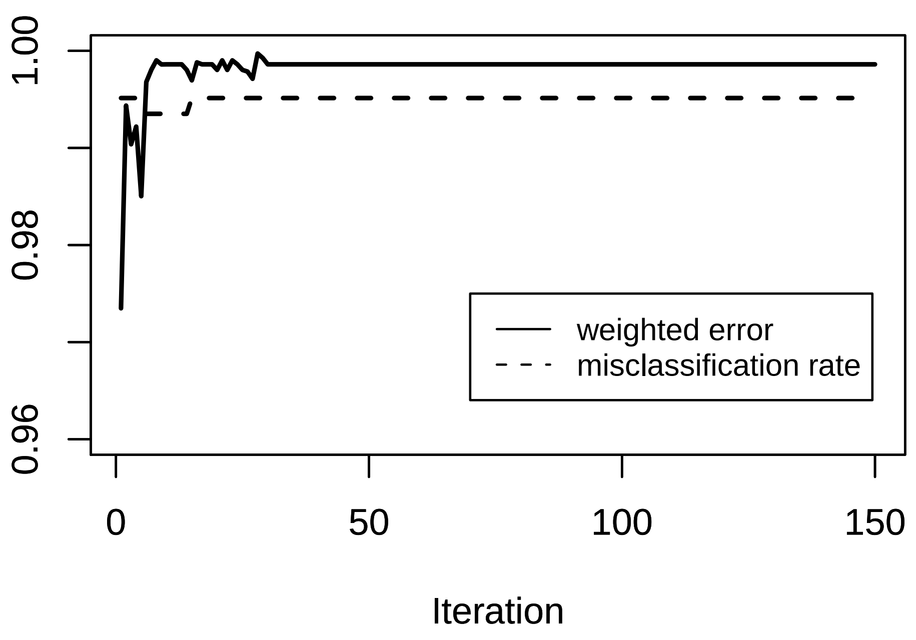

Figure 2.1: Weighted error and misclassification rate for Housing data (J=6)

and $J! = 720$ possible rankings. Applying the multi-class AdaBoost algorithm with the decision tree as a weak classifier. Figure 2.1 shows the misclassification rate over the time of iterations, we can see that misclassification rate remains at a very high level. The reason is that since the number of class $J!$ is too large, the weighted error ($err^{(m)}$ in Algorithm 2) approaches to one quickly, hence the weights do not change anymore and the boosting algorithm fails.

In this chapter, we suggest decomposing the ranking data into a number of pairs and applying forward stagewise additive model to derive a boosting algorithm for ranking data.

2.3.2 Model evaluation

In multi-class classification problem, people usually use misclassification rate (or error rate) to evaluate the classifier. However, in the problem of ranking, using the 0-1 evaluation (0 for correct classification, 1 for misclassification) may not be appropriate. The reason is that when the number of items increases, the number of possible rankings will be large and this will make the misclassification rate very high. When we rank 10 items, it is almost impossible to correctly predict the ordering of all ten items, even though some rankings may be just slightly different from their true rankings. For example, the observed ranking is $\boldsymbol{\pi}_0 : (a_1 \succ a_2 \succ a_3 \succ a_4)$, the predicted rankings for two classifiers T_1 and T_2 are: $\boldsymbol{\pi}_1 : (a_1 \succ a_2 \succ a_4 \succ a_3)$, $\boldsymbol{\pi}_2 : (a_4 \succ a_3 \succ a_2 \succ a_1)$. Note that both of them are misclassified. It is obviously that $\boldsymbol{\pi}_1$ is better than $\boldsymbol{\pi}_2$ as $\boldsymbol{\pi}_1$ correctly predicts the top two favorite choices, but $\boldsymbol{\pi}_2$ is in reverse order.

Therefore, it is more appropriate to evaluate the ranking performance by a distance which measures how close the predicted rankings to the observed ranking.

To do this, we have to define a distance function $d(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$ between two rankings which satisfies the following properties: (i) $0 \leq d(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) \leq 1$, (ii) $d(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = 0$ if and only if $\boldsymbol{\pi}_1 = \boldsymbol{\pi}_2$, (iii) $d(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = 1$ if the two rankings are reverse to each other. Many ranking distances have been proposed, the most widely used one is the Spearman distance. The Spearman distance between two rankings $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$ is defined as follows

$$d_s(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \frac{3\sum_{i=1}^{J}[\pi_1(i) - \pi_2(i)]^2}{J(J^2-1)}, \tag{2.3.2.1}$$

which is a linear transformation of the sum of squared rank distances. Another frequently used ranking distance is the Kendall distance. The Kendall distance between two rankings $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$ is defined as follows

$$d_k(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \frac{2\sum_{i<j}\mathbf{I}\{[\pi_1(i) - \pi_1(j)][\pi_2(i) - \pi_2(j)] < 0\}}{J(J-1)}, \tag{2.3.2.2}$$

which measures the proportion of pairwise disagreement between two rankings. The larger the Spearman/Kendall distance, the more dissimilar the two rankings are. Given a testing dataset $\{(\mathbf{x}_i, \boldsymbol{\pi}_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ and $\boldsymbol{\pi}_i$ are the characteristics and ranking of the i-th observation. Using the above ranking distances, we can evaluate the performance of a classifier T for the testing data by the so called Average Spearman/Kendall Distance:

$$\text{Average Spearman Distance} = \frac{\sum_{i=1}^{N} d_s(\boldsymbol{\pi}_i, \mathrm{T}(\mathbf{x}_i))}{N}$$

$$\text{Average Kendall Distance} = \frac{\sum_{i=1}^{N} d_k(\boldsymbol{\pi}_i, \mathrm{T}(\mathbf{x}_i))}{N}.$$

We have already normalized the Spearman/Kendall distance to $[0, 1]$. Then the Average Spearman/Kendall Distance will also between 0 and 1, the smaller the value is, the better the model performs.

2.3.3 Forward stagewise additive modeling for ranking data

In a ranking $\boldsymbol{\pi}^n = (\pi^n(1), \ldots, \pi^n(J))$ of J items: $\{a_1, a_2, \ldots, a_J\}$ given by the n-th observation, we can obtain $K(= J(J-1)/2)$ item pairs: $(a_i, a_j), i < j, 1 \leq i, j \leq J$. For each observation and its item pairs $(a_i, a_j), i < j$, we define $\mathbf{y}_n = \{y^n_{i,j}\}_{i<j}, n = 1, \ldots, N, 1 \leq i, j \leq J$, where:

$$y^n_{i,j} = \begin{cases} 1 & \text{if } \pi^n(i) > \pi^n(j) \\ -1 & \text{if } \pi^n(i) < \pi^n(j). \end{cases}$$

In order to build a forward stagewise additive model for ranking data, we propose the following exponential loss function for the pairwise comparison $\mathbf{y}$ declared from a ranking $\boldsymbol{\pi}$:

$$L(\mathbf{y}, \mathbf{f}(\mathbf{x})) = \sum_{i<j} \exp(-y_{i,j} f_{i,j}(\mathbf{x})), \tag{2.3.3.1}$$

where $\mathbf{f}(\mathbf{x}) = \{f_{i,j}(\mathbf{x})\}_{i<j}$ and its meaning will be explained later. Given the training data, our purpose is to find the $\mathbf{f}$ in order to minimize

$$\sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n)).$$

Similar to the derivation of two-class AdaBoost, we consider $\mathbf{f(x)}$ as the linear combination of basic classifiers:

$$\mathbf{f(x)} = \sum_{m=1}^{M} \beta^{(m)} \mathbf{T}^{(m)}(\mathbf{x}), \tag{2.3.3.2}$$

where the basic classifier $\mathbf{T}(\mathbf{x}) = \{\mathrm{T}_{i,j}(\mathbf{x})\}_{i<j}$ are defined as:

$$\mathrm{T}_{i,j}(\mathbf{x}) = \begin{cases} 1 & \text{if classifier predicts that } \pi(i) > \pi(j), \\ -1 & \text{if classifier predicts that } \pi(i) < \pi(j). \end{cases}$$

Forward stagewise modeling will find $\mathbf{f(x)}$ by sequentially adding new β and $\mathbf{T}$ to it without adjusting the parameters and classifiers that have already been added. In other words, the algorithm (Algorithm 3) starts with $\mathbf{f}^{(0)}(\mathbf{x}) = 0$ and add one new classifier to it in each stage.

Algorithm 3 Pseudo-code of the forward stagewise additive modeling for ranking data

```
1. Initialize $\mathbf{f}^{(0)}(\mathbf{x}) = 0$.
2. for $m = 1$ to $M$
3. Find $\left(\beta^{(m)}, \mathbf{T}^{(m)}(\mathbf{x})\right)$, where
       $\left(\beta^{(m)}, \mathbf{T}^{(m)}(\mathbf{x})\right) = \arg\min_{\beta,\mathbf{T}} \sum_{n=1}^{N} L(y_n, \mathbf{f}^{(m-1)}(\mathbf{x}_n) + \beta\mathbf{T}(\mathbf{x}_n))$.
4. Set $\mathbf{f}^{(m)}(\mathbf{x}) = \mathbf{f}^{(m-1)}(\mathbf{x}) + \beta^{(m)}\mathbf{T}^{(m)}(\mathbf{x})$.
5. end for
```

The essential part of this algorithm is how to find $\beta^{(m)}$ and $\mathbf{T}^{(m)}$ in step 3.

Theorem 1. The solutions to step 3 in Algorithm 3 are:

$$\mathbf{T} = \arg\min_{\mathbf{T}} \sum_{n=1}^{N} \sum_{i<j} w_{i,j}^{n,m} \mathbf{I}(y_{i,j}^n \neq T_{i,j}(\mathbf{x}_n)), \tag{2.3.3.3}$$

$$\beta^{(m)} = \frac{1}{2} \log \frac{\sum_{i<j} \sum_{y_{i,j}^n = T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}}{\sum_{i<j} \sum_{y_{i,j}^n \neq T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}}, \tag{2.3.3.4}$$

$$w_{i,j}^{n,m} = \exp(-y_{i,j}^n f_{i,j}^{(m-1)}).$$

Proof. By the definition of loss function given in (2.3.3.1),

$$\begin{aligned}
&\Big(\beta^{(m)}, \mathbf{T}^{(m)}(\mathbf{x})\Big) \\
&= \arg\min_{\beta,\mathbf{T}} \sum_{n=1}^{N} \sum_{i<j} \exp(-y_{i,j}^n (f_{i,j}^{(m-1)} + \beta T_{i,j})) \\
&= \arg\min_{\beta,\mathbf{T}} \sum_{n=1}^{N} \sum_{i<j} w_{i,j}^{n,m} \exp(-y_{i,j}^n \beta T_{i,j}) \\
&= \arg\min_{\beta,\mathbf{T}} \sum_{i<j} \left[e^{-\beta} \sum_{y_{i,j}^n = T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m} + e^{\beta} \sum_{y_{i,j}^n \neq T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m} \right] \quad (\star) \\
&= \arg\min_{\beta,\mathbf{T}} \left[e^{-\beta} \sum_{n=1}^{N} \sum_{i<j} w_{i,j}^{n,m} + (e^{\beta} - e^{-\beta}) \sum_{n=1}^{N} \sum_{i<j} w_{i,j}^{n,m} \mathbf{I}(y_{i,j}^n \neq T_{i,j}(\mathbf{x}_n)) \right],
\end{aligned}$$

where $w_{i,j}^{n,m} = \exp(-y_{i,j}^n f_{i,j}^{(m-1)})$.

First, we need to find $\mathbf{T(x)}$. Since only the second term in brackets depends on $\mathbf{T}$, so when β is fixed, the optimal $\mathbf{T}(x)$ is given by:

$$\mathbf{T} = \arg\min_{\mathbf{T}} \sum_{n=1}^{N} \sum_{i<j} w_{i,j}^{n,m} \mathbf{I}(y_{i,j}^n \neq T_{i,j}(\mathbf{x}_n)).$$

After obtaining $\mathbf{T}$, the optimal β can be determined by taking derivative of $(\star)$ with respect to β and setting it to 0:

$$\frac{\partial\left(e^{-\beta^{(m)}}\sum_{i<j}\sum_{y_{i,j}^n=T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m} + e^{\beta^{(m)}}\sum_{i<j}\sum_{y_{i,j}^n\neq T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}\right)}{\partial\beta^{(m)}} = 0.$$

Solving this gives:

$$\beta^{(m)} = \frac{1}{2}\log\frac{\sum_{i<j}\sum_{y_{i,j}^n=T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}}{\sum_{i<j}\sum_{y_{i,j}^n\neq T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}}.$$

□

According to (2.3.3.3), if $J = 3$, we need to find a classifier $\mathbf{T}$ which can minimize:

$$\sum_{n=1}^{N}\left[w_{1,2}^n\mathbf{I}(y_{1,2}^n \neq T_{1,2}(\mathbf{x}_n)) + w_{1,3}^n\mathbf{I}(y_{1,3}^n \neq T_{1,3}(\mathbf{x}_n)) + w_{2,3}^n\mathbf{I}(y_{2,3}^n \neq T_{2,3}(\mathbf{x}_n))\right].$$

We find that what we want to minimize is a weighted version of Kendall distance. When the weights $w_{i,j}^n$ are the same, it degenerates to the standard Kendall distance. So we call it weighted-Kendall distance. In the m-th stage of the boosting, we want to find a classifier $\mathbf{T}^{(m)}$ which minimizes the sum of weighted-Kendall distance for all training data.

The model will treat a ranking as a whole unity by finding the basic classifier in this way. The traditional pairwise comparison methods usually treat each pairs independently. From the results in Section 2.4 we can see that, the boosting method will not only reduce the number of cycles caused by pairwise comparison results but also achieve a better performance, compared to traditional pairwise comparison methods.

After we get β from (2.3.3.3), define $\alpha^{(m)} = 2\beta^{(m)} = \log[(1 - err^{(m)})/err^{(m)}]$, where

$$err^{(m)} = \frac{\sum_{i<j} \sum_{y_{i,j}^n \neq T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}}{\sum_{i<j} \sum_{n=1}^{N} w_{i,j}^{n,m}}. \tag{2.3.3.5}$$

Then it is clear that the weights should be updated by:

$$\begin{aligned} w_{i,j}^{n,m+1} &= \exp(-y_{i,j}^n f^{(m)}) & (2.3.3.6) \\ &= w_{i,j}^{n,m} \cdot \exp(-\beta^{(m)} y_{i,j}^n T_{i,j}(\mathbf{x}_n)) & (2.3.3.7) \\ &= w_{i,j}^{n,m} \cdot \exp(\alpha^{(m)} I(y_{i,j}^n \neq T_{i,j}(\mathbf{x}_n)). & (2.3.3.8) \end{aligned}$$

Therefore, the forward stagewise additive model for ranking data can start with $\mathbf{f}^{(0)}(\mathbf{x}) = 0$, and it can be updated by

$$\mathbf{f}^{(m)}(\mathbf{x}) = \mathbf{f}^{(m-1)}(\mathbf{x}) + \beta^{(m)} \mathbf{T}^{(m)}(\mathbf{x}). \tag{2.3.3.9}$$

2.3.4 Prediction of ranking

In each stage, we train a classifier based on the weighted training data. Each classifier $\mathbf{T}^{(m)}$ consists of K sub-classifiers: $\mathbf{T}^{(m)} = \{T_{i,j}^{(m)}\}_{i<j}$, while each sub-classifier compares a pair of items and tells people whether $\pi(i) > \pi(j)$ or $\pi(j) >$

$\pi(i)$. Then the forward stagewise additive model is a linear combination of these classifiers.

Theorem 2. For a new observation $\mathbf{x}$, the pairwise comparison probabilities for pair (a_i, a_j) can be estimated by:

$$\Pr(y_{i,j} = 1|\mathbf{x}) = \Pr(a_i \succ a_j) \qquad = \frac{e^{f_{i,j}(\mathbf{x})}}{e^{f_{i,j}(\mathbf{x})} + e^{f_{j,i}(\mathbf{x})}} \tag{2.3.4.1}$$

$$\Pr(y_{i,j} = -1|\mathbf{x}) = \Pr(a_j \succ a_i) \qquad = \frac{e^{f_{j,i}(\mathbf{x})}}{e^{f_{i,j}(\mathbf{x})} + e^{f_{j,i}(\mathbf{x})}} \tag{2.3.4.2}$$

where $f_{i,j} = \frac{1}{2}\sum_{m=1}^{M} \alpha^{(m)}\mathrm{T}_{i,j}^{(m)}(\mathbf{x})$.

Proof. I will investigate what the exponential loss function estimates. For pair (a_i, a_j), we want to find:

$$\arg\min_{\mathbf{f(x)}} E\left[\exp(-\frac{1}{2}(y_{i,j}f_{i,j} + y_{j,i}f_{j,i}))\right]$$

$$\text{subject to: } f_{i,j} + f_{j,i} = 0.$$

Using the Lagrange method, the minimization problem can be written as:

$$\exp\left(-\frac{f_{i,j}(\mathbf{x})}{2}\right)\Pr(y_{i,j} = 1) + \exp\left(-\frac{f_{j,i}(\mathbf{x})}{2}\right)\Pr(y_{i,j} = -1) - \lambda\left(f_{i,j}(\mathbf{x}) + f_{j,i}(\mathbf{x})\right),$$

where λ is the Lagrange multiplier. Taking derivatives with respect to $f_{i,j}$,$f_{j,i}$ and λ, we can obtain the population minimizer for each pair:

$$f_{i,j}(\mathbf{x}) = \left[\log\Pr(y_{i,j} = 1) - \log\Pr(y_{i,j} = -1)\right]/2 \tag{2.3.4.3}$$

$$f_{j,i}(\mathbf{x}) = \left[\log \Pr(y_{i,j} = -1) - \log \Pr(y_{i,j} = 1)\right]/2. \tag{2.3.4.4}$$

Thus, the probabilities (2.3.4.1) and (2.3.4.2) can be estimated from (2.3.4.3) and (2.3.4.4). $\square$

In order to assign a ranking for a new observation, we can derive the ranking from all K pairwise comparison relations. However, these relations will cause cycles sometimes. For example, if the model predicts that $a_1 \succ a_2, a_2 \succ a_3$ and $a_3 \succ a_1$, then we cannot derive a ranking of a_1, a_2 and a_3. I minimize the weighted-Kendall distance and treat the ranking as a whole unity during the training process, which can make cycles fewer in some sense, but cycles cannot be totally avoided.

If cycles exist, how to induce a ranking from pairwise relations has received a lot of attention. A simple though effective method is the voting strategy. Each item a_i is evaluated by the sum of pairwise relations (voting method):

$$S_i = \sum_{j \neq i} \mathbf{I}(a_i \succ a_j), \tag{2.3.4.5}$$

or by the sum of probabilities (weighted voting method):

$$S_i = \sum_{j \neq i} \Pr(a_i \succ a_j). \tag{2.3.4.6}$$

We call S_i the evaluation variable. Then all items are ordered according to these evaluation variables and a ranking is obtained. (In case of some $S_i = S_j$, we just assign the ordering of a_i and a_j by the more frequent pattern of $(a_i \succ a_j)$ and

$(a_j \succ a_i)$ from the training rankings).

$$(a_i \succ a_j) \Leftrightarrow S_i \geq S_j. \tag{2.3.4.7}$$

Although this ranking procedure appears a little ad-hoc, Hüllermeier et al. (2008) gave some theoretical proofs to support this method. They assumed that for any individual $\mathbf{x}$, there is a probability distribution of rankings $P(\cdot|\mathbf{x})$. The performance of a ranking model should be evaluated by the distance between real ranking and predict ranking, and it can also be evaluated by the expected distance between real ranking and predict ranking:

$$\mathbf{E}(d(\boldsymbol{\pi}, T(\mathbf{x}))|\mathbf{x}),$$

where $d(\cdot)$ is the distance between rankings, $\boldsymbol{\pi}$ is the real ranking and $T(\mathbf{x})$ is the predict ranking. The expectation is taken over all possible $J!$ rankings.

Hüllermeier et al. (2008) proved that if $d(\cdot)$ is the Spearman distance and $\boldsymbol{\pi}_0$ is the real ranking, the expected distance between real and predict rankings $\mathbf{E}(d(\boldsymbol{\pi}, \boldsymbol{\pi}_0)|\mathbf{x})$ becomes minimal by choosing $\boldsymbol{\pi}$ such that:

$$\pi(i) \leq \pi(j) \text{ whenever } S_i \geq S_j,$$

where $S_i = \sum_{j \neq i} Pr(a_i \succeq a_j)$. This result shows that a ranking predicted by (2.3.4.5) and (2.3.4.6) can achieve the minimal expected Spearman distance to real ranking.

When $d(\cdot)$ is the Kendall distance, the expected distance between real and predict rankings is:

$$\begin{aligned}
\mathbf{E}(d(\boldsymbol{\pi}, \boldsymbol{\pi}_0)|\mathbf{x}) &= \sum_{\pi} \Pr(\boldsymbol{\pi}|\mathbf{x}) \times d(\boldsymbol{\pi}, \boldsymbol{\pi}_0) \\
&= \sum_{\pi} \sum_{i<j, \pi(i)<\pi(j)} \Pr(\boldsymbol{\pi}|\mathbf{x}) \times \mathbf{I}(\pi(i) > \pi(j)) \\
&= \sum_{i<j, \pi(i)<\pi(j)} \Pr(a_i \succ a_j) \qquad (2.3.4.8)
\end{aligned}$$

In this case, the ranking which can minimize the expected Kendall distance to real ranking can only be determined by:

$$\widehat{\boldsymbol{\pi}} = \arg\min_{\pi} \sum_{i<j, \pi(i)<\pi(j)} \Pr(a_i \succ a_j). \qquad (2.3.4.9)$$

2.3.5 The weak classifier

In each step of the boosting algorithm, a classifier will be trained which will minimize the sum of weighted-Kendall distance for all training data. I will use decision tree as the basic classifier, the splitting criterion is to minimize the sum of weighted-Kendall distance for both left and right child nodes. Therefore, we need to prove that this splitting criterion is valid, i.e. a valid impurity function can be found.

Theorem 3. If ϕ is defined as:

$$\phi = \sum_{i<j} \phi_{i,j}(p_{i,j}^{+1}, p_{i,j}^{-1}) = \sum_{i,j} \min(p_{i,j}^{+1}, p_{i,j}^{-1}), \qquad (2.3.5.1)$$

where $p_{i,j}^{+1} = \sum_{y_{i,j}^n=1} w_{i,j}^n$ and $p_{i,j}^{-1} = \sum_{y_{i,j}^n=-1} w_{i,j}^n$. Then ϕ is a valid impurity function.

Proof. Since all (i, j) pairs are independent, ϕ achieves maximum only when all $\phi_{i,j}$ achieve maximum, and it only happens when $p_{i,j}^{+1} = p_{i,j}^{-1}$ for all (i, j) pairs. Therefore ϕ reaches maximum only if all the p_i's are equal.

By similar method, ϕ achieves minimum only at these points: $(1, 0, \cdots, 0)$, $(0, 1, \cdots, 0)$, $\cdots$, $(0, 0, \cdots, 1)$, i.e. when the probability of being in some class is 1 and 0 for all other classes. Finally, it is trivial to prove that ϕ is symmetric function of $p_1, \cdots, p_J$, and ϕ remains constant if we permuted p_js. □

2.3.6 AdaBoost algorithm for ranking data

Finally, I build a boosting model for ranking data based on forward stagewise additive method and predict the ranking through pairwise comparison relations. If cycles exist, the ranking can be predicted by (2.3.4.7) and (2.3.4.9) respectively, for Spearman or Kendall distance criterion. I summarize the algorithm (AdaBoostRanking) in Algorithm 4.

In Algorithm 4, the classifiers $\mathrm{T}_{i,j} \in \{-1, +1\}$ in each boosting iteration. We can extend this binary result to a probability between 0 and 1, and the probability can be estimated from training data:

$$\Pr(y_{i,j}^n \neq T_{i,j}^{(m)}(x_n)) = \min(p_{i,j}^{+1}, p_{i,j}^{-1}), \tag{2.3.6.1}$$

where $p_{i,j}^{+1} = \sum_{y_{i,j}^n=1} w_{i,j}^{n,m}$ and $p_{i,j}^{-1} = \sum_{y_{i,j}^n=-1} w_{i,j}^{n,m}$. This is similar to the process of extending the original AdaBoost algorithm to Real AdaBoost algorithm. The

Real version of AdaBoostRanking is given in Algorithm 5, I modify step 3, 6 and 9 to the probability version.

2.4 Applications

I use the datasets from the Label Ranking Datasets (Cheng et al., 2009), these ranking datasets are synthetic from some multi-class category datasets from the UCI-KDD Archive.

For each dataset, I apply the AdaBoostRanking and AdaBoostRanking.Real algorithms. I boost one hundred steps to monitor the change of Average Spearman/Kendall Distance. In each step of boosting, a weak classifier will be fit to training data so that the sum of weighted-kendall is minimized.

I use decision tree as the weak classifier, and the splitting criterion is to minimize the sum of weighted-kendall for both left and right nodes. The splitting will stop if the node is pure or the size of the node is smaller than one tenth of the size of the training data. In the prediction part, if there are no cycles, a ranking is predicted from pairwise comparison relations, otherwise the ranking is predicted by (2.3.4.6) and (2.3.4.9) for Spearman and Kendall criterions.

For benchmark comparisons, I use decision tree model for ranking (Yu et al., 2009), rank-order-logit model (Beggs et al., 1981), and pairwise comparison model (Hüllermeier et al., 2008). For the decision tree model for ranking, since we

Algorithm 4. Pseudo-code of AdaBoostRanking algorithm

```
1: Initialize weights $w_{i,j}^{n,1} = 1/NK$.
2: for $m = 1$ to $M$ do
3:     Fit a classifier $\mathbf{T}^{(m)}(\mathbf{x}) = \{\mathrm{T}_{i,j}^{(m)}(\mathbf{x})\}_{i<j}$ to the training data
       which should minimize the sum of the weighted-Kendall:
       $$\sum_{n=1}^{N}\sum_{i<j} w_{i,j}^{n,m}\mathbf{I}(y_{i,j}^{n} \neq \mathrm{T}_{i,j}^{(m)}(\mathbf{x}_n))$$
4:     Compute
       $$err^{(m)} = \frac{\sum_{i<j}\sum_{y_{i,j}^{n}\neq T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}}{\sum_{i<j}\sum_{n=1}^{N} w_{i,j}^{n,m}}, \ \alpha^{(m)} = \log\frac{1-err^{(m)}}{err^{(m)}}$$
5:     Update $w_{i,j}^{n,m+1} = w_{i,j}^{n,m} \cdot \exp(\alpha^{(m)}\mathbf{I}(y_{i,j}^{n} \neq \mathrm{T}_{i,j}(\mathbf{x}_n))$
6:     Re-normalize $w_{i,j}^{n,m+1}$, make sure $\sum_{n=1}^{N}\sum_{i<j} w_{i,j}^{n,m+1} = 1$
7: end for
8: For a testing data $\mathbf{x}_{test}$, compute
   $$f_{i,j} = \sum_{m=1}^{M} \alpha^{(m)}\mathrm{T}_{i,j}^{(m)}(\mathbf{x}_{test}),$$
   $\mathbf{I}(a_i \succ a_j) = 1$ if $f_{i,j} \geq 0$, and $\mathbf{I}(a_i \succ a_j) = 0$ if $f_{i,j} < 0$
9: If cycles exist, for Spearman distance, compute $S_i = \sum_{j\neq i}\mathbf{I}(a_i \succ a_j)$, the predicted ranking $\widehat{\boldsymbol{\pi}}$ is obtained by sorting $S_i$ by descending sequence.
10: If cycles exist, for Kendall distance, the predicted ranking is obtained by
   $$\widehat{\boldsymbol{\pi}} = \arg\min_{\pi} \sum_{i<j,\pi(i)<\pi(j)} \mathbf{I}(a_j \succ a_i)$$
```

Algorithm 5. Pseudo-code of AdaBoostRanking.Real algorithm

```
1: Initialize weights $w_{i,j}^{n,1} = 1/NK$.
2: for $m = 1$ to $M$ do
3:     Fit a classifier $\mathbf{T}^{(m)}(\mathbf{x}) = \{\mathrm{T}_{i,j}^{(m)}(\mathbf{x})\}_{i<j}$ to the training data which should minimize the sum of the weighted-Kendall:
       $$\sum_{n=1}^{N}\sum_{i<j} w_{i,j}^{n,m}\Pr(y_{i,j}^{n} \neq T_{i,j}^{(m)}(\mathbf{x}_n))$$
       $$\Pr(y_{i,j}^{n} \neq T_{i,j}^{(m)}(x_n) = \min(\sum_{y_{i,j}^{n}=1} w_{i,j}^{n,m}, \sum_{y_{i,j}^{n}=-1} w_{i,j}^{n,m})$$
4:     Compute
       $$err^{(m)} = \frac{\sum_{i<j}\sum_{y_{i,j}^{n}\neq T_{i,j}(\mathbf{x}_n)} w_{i,j}^{n,m}}{\sum_{i<j}\sum_{n=1}^{N} w_{i,j}^{n,m}}, \ \alpha^{(m)} = \log\frac{1-err^{(m)}}{err^{(m)}}$$
5:     Update $w_{i,j}^{n,m+1} = w_{i,j}^{n,m} \cdot \exp(\alpha^{(m)}\Pr(y_{i,j}^{n} \neq T_{i,j}(\mathbf{x}_n))$
6:     Re-normalize $w_{i,j}^{n,m+1}$, make sure $\sum_{n=1}^{N}\sum_{i<j} w_{i,j}^{n,m+1} = 1$
7: end for
8: For a testing data $\mathbf{x}_{test}$, compute
       $$\Pr(a_i \succ a_j) = \frac{e^{f_{i,j}}}{e^{f_{i,j}} + e^{f_{j,i}}}; \Pr(a_j \succ a_i) = 1 - \Pr(a_i \succ a_j), i < j$$
   where $f_{i,j} = \sum_{m=1}^{M} \alpha^{(m)}\mathrm{T}_{i,j}^{(m)}(\mathbf{x}_{test}), 0 \leq T_{i,j}^{(m)}(\mathbf{x}_{test}) \leq 1$
9: If cycles exist, for Spearman distance, compute $S_i = \sum_{j\neq i}\Pr(a_i \succ a_j)$, the predicted ranking $\widehat{\boldsymbol{\pi}}$ is obtained by sorting $S_i$ by descending sequence.
10: If cycles exist, for Kendall distance, the predicted ranking is obtained by
       $$\widehat{\boldsymbol{\pi}} = \arg\min_{\pi} \sum_{i<j,\pi(i)<\pi(j)} \Pr(a_j \succ a_i)$$
```

evaluate the performance of classifier by Average Spearman/Kendall Distance, so I assign a ranking to each leaf node which can minimize the total Spearman/Kendall distance for all rankings in that leaf node. For the rank-ordered-logit model, I normalize each covariate to $[-1, 1]$. For the pairwise comparison model, I build $K(= L(L-1)/2)$ binary classifiers independently for K pairs. Two-class AdaBoost model is used as the binary classifier (boost 100 steps), and the ranking is predicted by weighted voting strategy.

For AdaBoostRanking, I average the results through four-fold cross-validation. Finally I plot the boosting results in Figures 2.2-2.9, and list the numerical results of Average Spearman/Kendall distance in Table 2.1.

For all algorithms, I compute the proportion of test instances that pairwise relations do not cause cycles. I also compute the proportion of pairs which are correctly predicted. The results are summarized in Table 2.2.

2.5 Conclusion

From the plots and results in Table 2.1, we can see that the AdaBoostRanking algorithm can reduce the Average Spearman/Kendall Distance. In most cases, the performance become stable within 100 iterations. For these datasets, both AdaBoostRanking and AdaBoostRanking.Real can obtain smaller Average Spearman/Kendall Distance than pairwise comparison model, decision tree

Table 2.1. Summary of datasets properties and Average Spearman/Kendall Distance of simulation results. J is the number of items, F is the number of characteristics and N is the number of observations. adaRank and adaRank.R means the original and real versions of AdaBoost.Ranking algorithm, PWC is the pairwise comparison model, RankTree is the ranking tree model, ROL is the rank-ordered-logit model. The best results are in bold font.

Dataset	J	F	N	distance	adaRank	adaRank.R	PWC	RankTree	ROL
Vehicle	4	18	846	(Spearman)	0.058	**0.051**	0.066	0.080	0.094
				(Kendall)	0.071	**0.063**	0.081	0.091	0.111
Calhouse	4	4	10000	(Spearman)	**0.275**	0.283	0.315	0.296	0.321
				(Kendall)	0.313	**0.303**	0.345	0.361	0.412
Authorship	4	70	841	(Spearman)	0.047	**0.038**	0.052	0.065	0.083
				(Kendall)	0.059	**0.047**	0.066	0.081	0.103
Stock	5	5	950	(Spearman)	0.093	**0.085**	0.132	0.109	0.128
				(Kendall)	0.072	**0.066**	0.163	0.133	0.165
Fried	5	9	10000	(Spearman)	0.135	**0.118**	0.170	0.188	0.213
				(Kendall)	0.172	**0.156**	0.201	0.211	0.245
Glass	6	9	214	(Spearman)	0.043	**0.042**	0.054	0.061	0.091
				(Kendall)	0.058	**0.057**	0.081	0.101	0.132
Housing	6	4	506	(Spearman)	0.262	**0.231**	0.311	0.319	0.333
				(Kendall)	0.275	**0.251**	0.312	0.321	0.352
Segment	7	18	2310	(Spearman)	0.046	**0.044**	0.065	0.068	0.091
				(Kendall)	0.059	**0.056**	0.088	0.091	0.103

	adaBoostRanking.Real		adaBoostRanking		Pairwise Comparison	
Dataset	Non-cycle	Right-pair	Non-cycle	Right-pair	Non-cycle	Right-pair
Vehicle	98.4%	93.7%	97.7%	93.6%	96.1%	93.1%
Calhouse	98.2%	69.6%	89.5%	69.0%	99.5%	69.4%
Authorship	99.5%	95.0%	99.3%	93.9%	94.7%	94.0%
Stock	94.7%	90.0%	94.1%	89.4%	75.4%	89.0%
Fried	96.0%	83.0%	79.3%	82.0%	81.2%	79.1%
Glass	95.0%	94.0%	94.8%	93.1%	79.4%	93.7%
Housing	63.9%	74.3%	64.3%	72.2%	44.8%	72.2%
Segment	90.3%	94.4%	87.1%	93.8%	69.6%	93.5%

Table 2.2: Pairwise comparison and cycles information for different models. Non–cycle means the proportion of testing instances that pairwise relations do not cause cycles, Right-pair means the proportion of pairs which are correctly predicted.

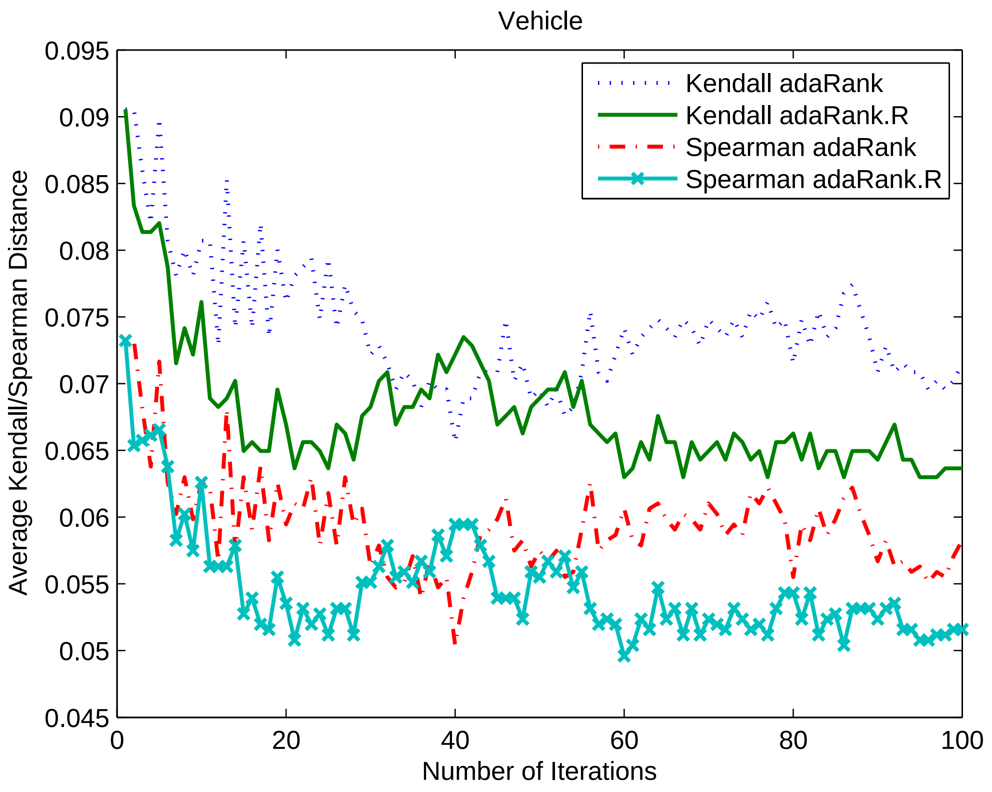

Figure 2.2: Vehicle. Changes of Average Spearman/Kendall Distance after boosting

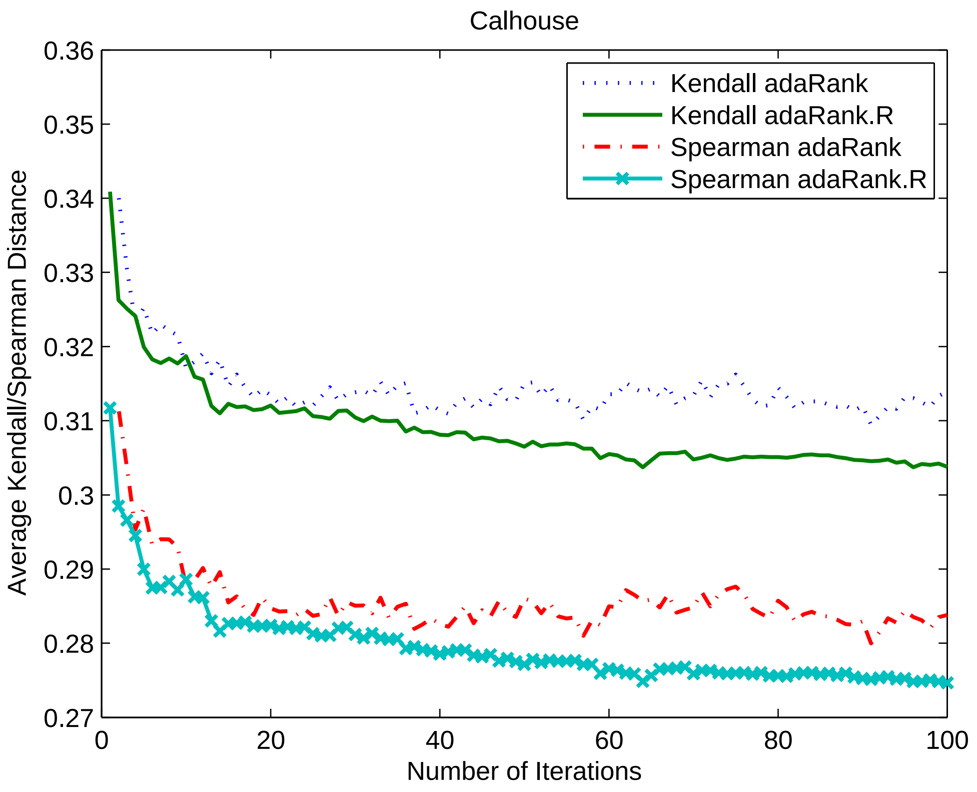

Figure 2.3: Calhouse. Changes of Average Spearman/Kendall Distance after boosting

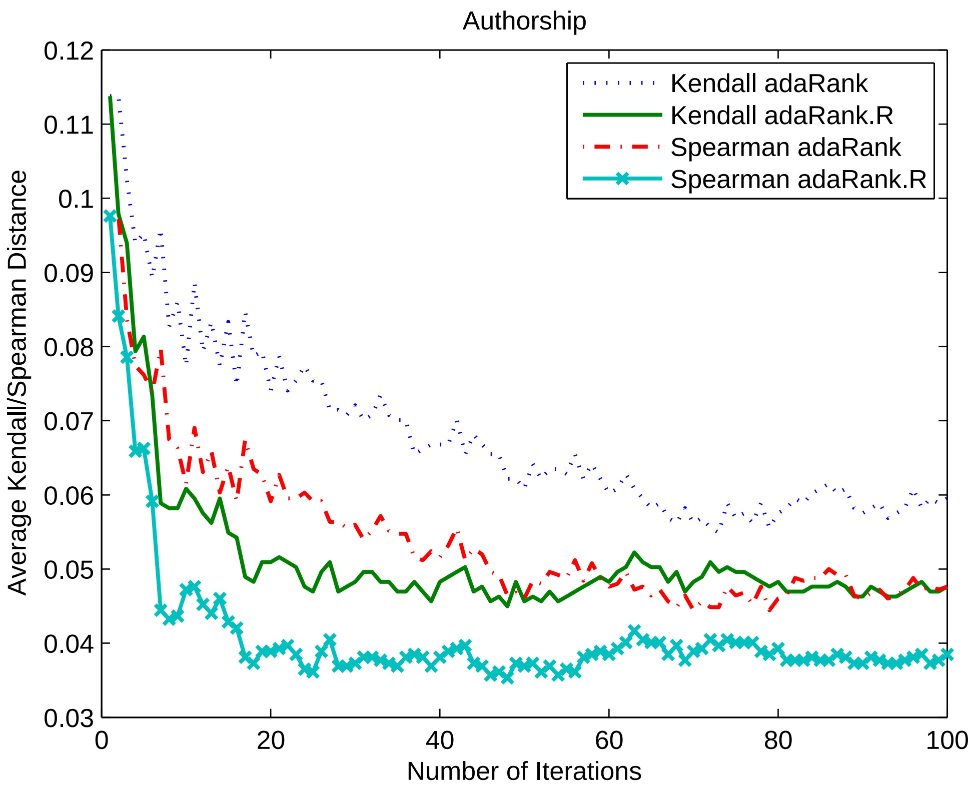

Figure 2.4: Authorship. Changes of Average Spearman/Kendall Distance after boosting

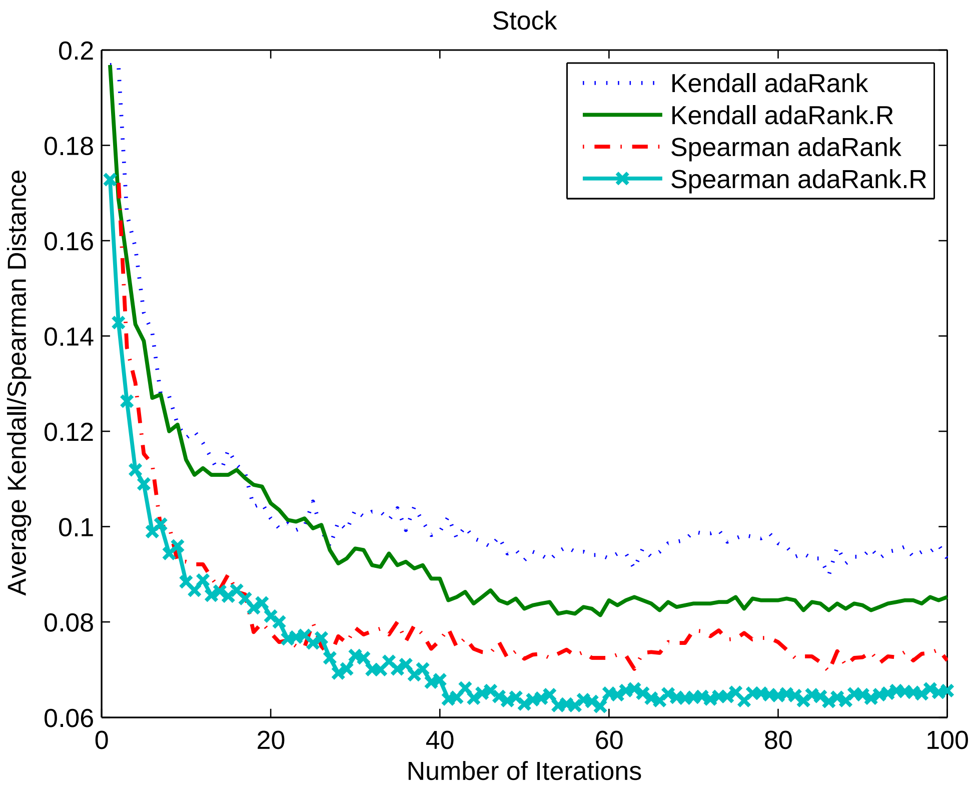

Figure 2.5: Stock. Changes of Average Spearman/Kendall Distance after boosting

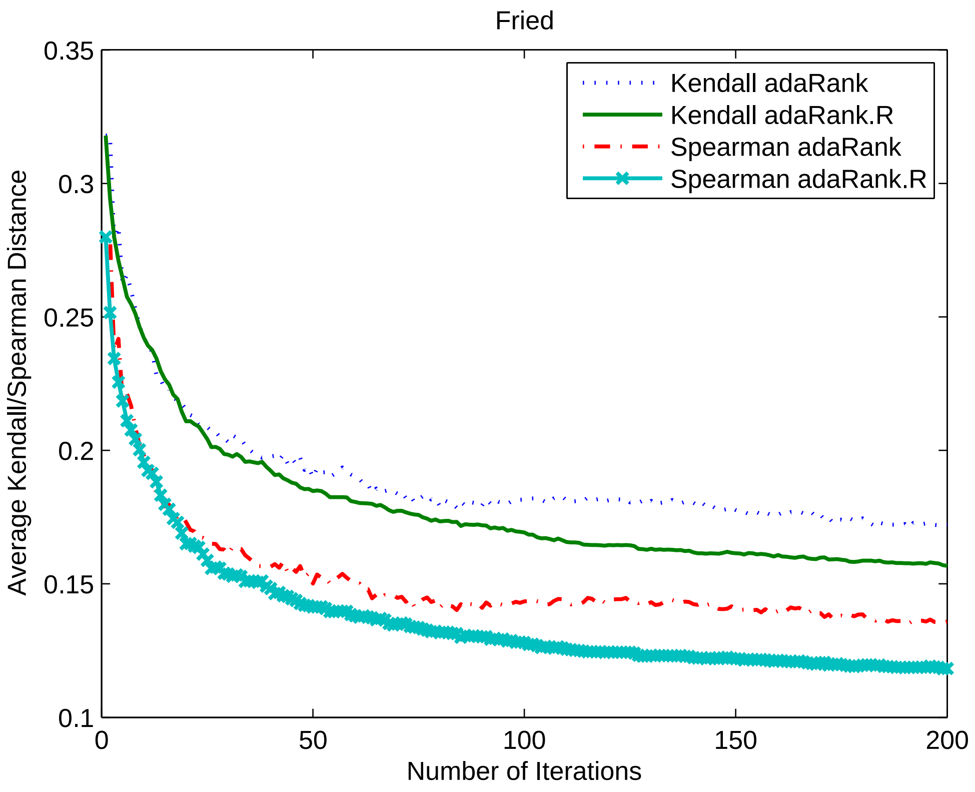

Figure 2.6: Fried. Changes of Average Spearman/Kendall Distance after boosting

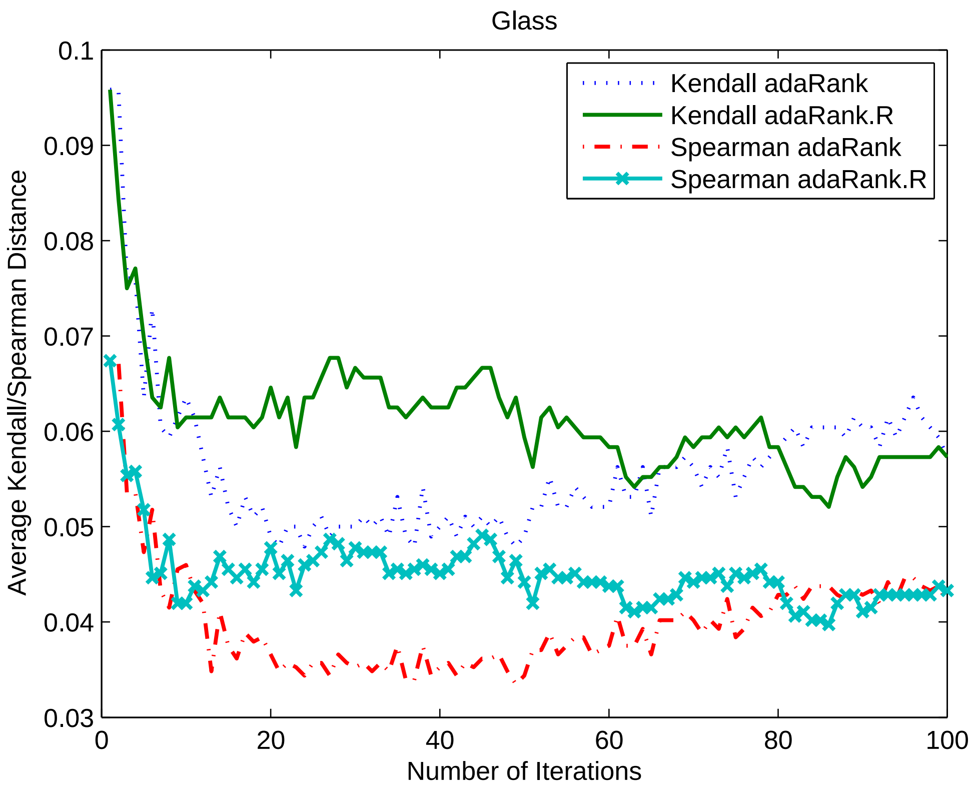

Figure 2.7: Glass. Changes of Average Spearman/Kendall Distance after boosting

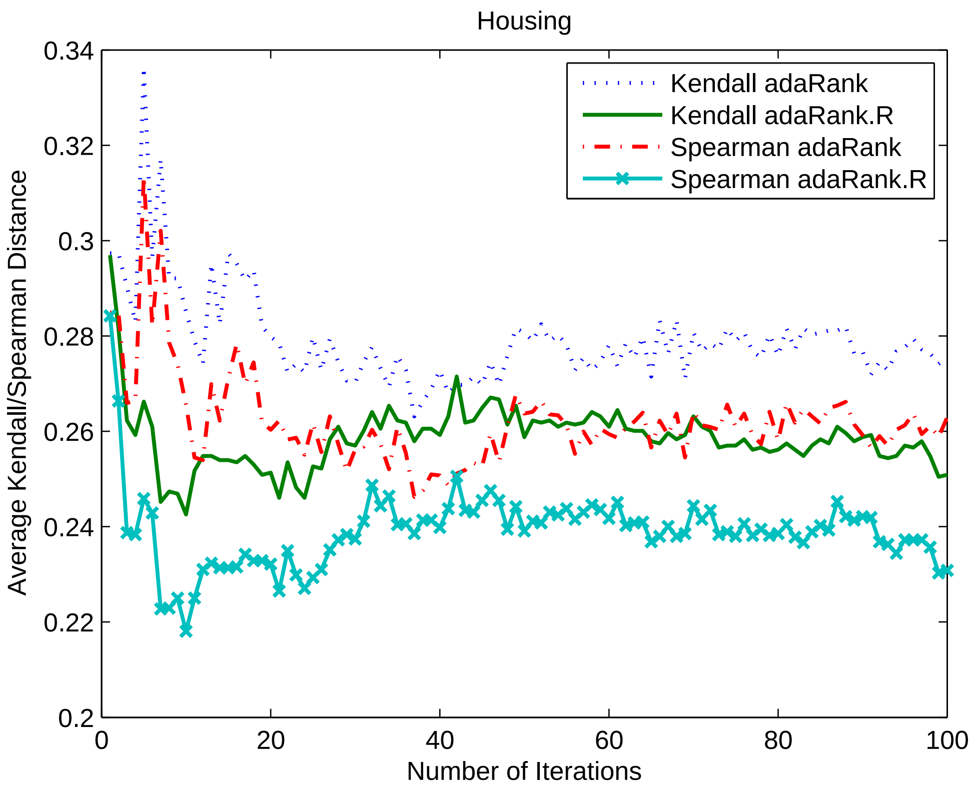

Figure 2.8: Housing. Changes of Average Spearman/Kendall Distance after boosting

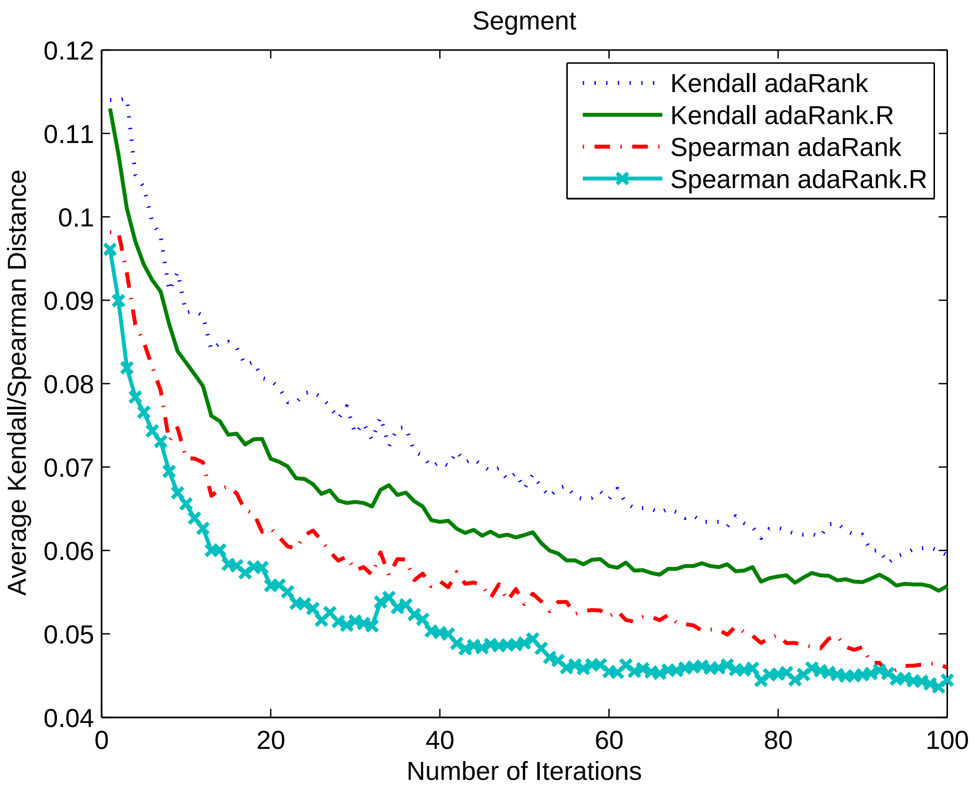

Figure 2.9: Segment. Changes of Average Spearman/Kendall Distance after boosting

model and rank-ordered-logit model.

We can also see that for most of the datasets, the real version is better than the original version, and the boosting curve for real version is always less frustrating than the original version. Because the real version can obtain more information from the probabilities, and the changes of weights are more smooth in each iteration.

From the results in Table 2.2 we can see, AdaBoostRanking model can reduce the number of cycles compared to the pairwise comparison model. For the dataset Segment, around 30% data have cycles for pairwise comparison model, but only 10% data have cycles for boosting model. The reason is that I treat the ranking as a whole unity during the training process, while the pairwise comparison model treat each pair independently.

Unlike the Multi-Class AdaBoost algorithm which would not work for large J as mentioned in Section 2.2, the proposed AdaBoostRanking algorithm can provide obvious improvement even when $J = 6$ or 7. For future investigation, I hope to extend the algorithm to deal with incomplete ranking and the tie situation in ranking.

Chapter 3

Predicting ranking by local distance learning

3.1 Introduction

Distance-based models (Mallows, 1957) have the advantages of being simple and elegant. The basic idea of distance-based models is: the closer a ranking $\boldsymbol{\pi}$ to its true ranking $\boldsymbol{\pi}_0$, the more likely that $\boldsymbol{\pi}$ being observed. Cheng et al. (2009) built an instance-based model to predict rankings by combining distance-based model and the k nearest neighbors (kNN) method. However, this model does not allow different weights of the k neighbors combined in prediction, and it does not consider the relative importance of each feature in determining the distance between two instances either. In this chapter, I will improve this model by assigning weights to both instances and features.

This chapter is organized as follow: In Section 3.2 will illustrate how to use distance-based model and kNN method to build an instance-based model for ranking data. In Section 3.3 will present the detail of weighted kNN method and kernel method, In Section 3.4 will propose the algorithm for assigning weights to features. Experiment studies and conclusions will be given in Section 3.5.

3.2 Instance-based model for ranking data

Based on the classical distance-based model introduced by Mallows (1957), the probability that a ranking $\boldsymbol{\pi}$ is observed equals to:

$$\Pr(\boldsymbol{\pi}|\theta, \boldsymbol{\pi}_0) = \frac{\exp(-\theta d(\boldsymbol{\pi}, \boldsymbol{\pi}_0))}{\phi(\theta)}, \tag{3.2.0.1}$$

where the ranking $\boldsymbol{\pi}_0$ is the location parameter (central ranking) and θ is a spread parameter. $d(\cdot)$ measures the difference between two rankings and $\phi(\theta)$ is the normalization factor. Distance-based model assigns the maximum probability to the central ranking $\boldsymbol{\pi}_0$. The larger the distance $d(\boldsymbol{\pi}, \boldsymbol{\pi}_0)$, the smaller probability that $\boldsymbol{\pi}$ will be observed.

Fligner and Verducci (1988) showed that, when Kendall distance is used as the ranking distance $d(\cdot)$, the normalization constant is given by:

$$\phi(\theta) = \prod_{j=1}^{J} \frac{1 - \exp(-j\theta)}{1 - \exp(-\theta)}. \tag{3.2.0.2}$$

In a ranking dataset $\{\mathbf{x}_i, \boldsymbol{\pi}_i\}_{i=1}^{N}$, where $\mathbf{x}_i$ is the feature vector for the i-th instance and $\boldsymbol{\pi}_i$ is the corresponding ranking for J items. For a new instance $\mathbf{x}$, we can predict its ranking based on its nearest neighbors. Assume $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ are the k nearest neighbors of $\mathbf{x}$, each of them associates with a ranking $\boldsymbol{\pi}_i$. By the assumption that the probability $\Pr(\cdot|\mathbf{x})$ is locally constant around the query instance $\mathbf{x}$, the probability to observe all k rankings $\{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_k\}$ of these k-nearest neighbors is:

$$P(\{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_k\}|\theta, \boldsymbol{\pi}) = \prod_{i=1}^{k} P(\boldsymbol{\pi}_i|\theta, \boldsymbol{\pi}) = \frac{\exp(-\theta \sum_{i=1}^{k} d(\boldsymbol{\pi}_i, \boldsymbol{\pi}))}{\prod_{i=1}^{k} \phi(\theta)}$$

$$= \frac{\exp(-\theta \sum_{i=1}^{k} d(\boldsymbol{\pi}_i, \boldsymbol{\pi}))}{(\prod_{j=1}^{n} \frac{1-\exp(-j\theta)}{1-\exp(-\theta)})^k}.$$

It is easy to prove that the MLE of $\boldsymbol{\pi}$ is:

$$\hat{\boldsymbol{\pi}} = \arg\min_{\pi} \sum_{i=1}^{k} d(\boldsymbol{\pi}_i, \boldsymbol{\pi}). \tag{3.2.0.3}$$

Therefore, we can predict the ranking of $\mathbf{x}$ by finding its k nearest neighbors and using formula (3.2.0.3) .

3.3 Weighted kNN method and kernel function

The k-nearest-neighbor(kNN) method (Hodges, 1951) represents one of the simplest and most intuitive techniques in the field of data mining. It assumes that probabilities of classes are locally constant. The classical kNN method neglects the fact that neighbors who are close to the query instance should contribute more than neighbors who are far away on the classification problem. Therefore, kNN can be improved by assigning a weight to each of the k nearest neighbors. The weights of neighbors who are closer to the query instance should be bigger than the weights of neighbors who are farer away from the query instance.

The weighted kNN method is in fact nor new. Dudani (1976) first introduced a distance-weight kNN rule (DWKNN), which assigns a weight w_i to the i-th nearest neighbor with feature $\mathbf{x}_i$ for a query $\mathbf{x}$:

$$w_i = \begin{cases} \dfrac{D(\mathbf{x},\mathbf{x}_k) - D(\mathbf{x},\mathbf{x}_i)}{D(\mathbf{x},\mathbf{x}_k) - D(\mathbf{x},\mathbf{x}_1)} & \text{, if } D(\mathbf{x},\mathbf{x}_k) \neq D(\mathbf{x},\mathbf{x}_1) \ ; \\ 1 & \text{, if } D(\mathbf{x},\mathbf{x}_k) = D(\mathbf{x},\mathbf{x}_1) \ . \end{cases}$$

Some theoretical analysis (Bailey and Jain, 1978) showed that, given infinite training samples, the asymptotic error rate of unweighted kNN is always lower than that of any weighted kNN. However, the conclusion would not apply when the number of training samples is finite, and a number of experimental results have found that: weighted kNN can achieve better performance than unweighted kNN when the sample size is finite.

More general, the transition from metric distance to weight can be done by kernel methods (Hofmann et al., 2008). The kernels are functions $K(\cdot)$ of distances D, and these functions should satisfy the following properties:

1. $K(D) \geq 0$ for all $D \in \mathbf{R}$,

2. $K(D)$ gets its maximum at $D = 0$,

3. $K(D)$ descents monotonously for $D \to \pm\infty$.

Some widely used kernel examples are the following:

- Rectangular kernel: $K(D) = \frac{1}{2} \cdot \mathbf{I}(D \leq 1)$
- Triangular kernel: $K(D) = (1 - D) \cdot \mathbf{I}(D \leq 1)$
- Epanechnikov kernel: $K(D) = \frac{3}{4}(1 - D^2) \cdot \mathbf{I}(D \leq 1)$
- Quartic kernel: $K(D) = \frac{15}{16}(1 - D^2)^2 \cdot \mathbf{I}(D \leq 1)$
- Gaussian kernel: $K(D) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{D^2}{2\sigma^2})$
- Inversion kernel: $K(D) = \frac{1}{D}$

Except the Gaussian and inversion kernels, all these kernels require a window width: the weight will become zero if $D \geq 1$. In order to avoid this problem, we can standardize all distances by the distance from the $(k+1)^{th}$ neighbor. For a given query $\mathbf{x}$, let $\{\mathbf{x}_1, .., \mathbf{x}_k, \mathbf{x}_{k+1}\}$ be the nearest $k+1$ neighbors, and $\{D(\mathbf{x}, \mathbf{x}_1), ..., D(\mathbf{x}, \mathbf{x}_k), D(\mathbf{x}, \mathbf{x}_{k+1})\}$ are their distances to $\mathbf{x}$. Define:

$$\Phi(\mathbf{x}, \mathbf{x}_i) = \frac{D(\mathbf{x}, \mathbf{x}_i)}{D(\mathbf{x}, \mathbf{x}_{k+1})} \text{ for } i = 1, 2, .., k \tag{3.3.0.4}$$

After standardization, all distances for the k-nearest neighbors, $\Phi(\mathbf{x}, \mathbf{x}_i), 1 \leq i \leq k$, will lie within the interval $[0, 1]$.

Among the kernel functions we proposed above, the most popular and widely used is the Gaussian Kernel, where the weights will decrease along Gaussian density. In kNN method, finding the optimal value of k is a difficult task. However, if we apply the Gaussian kernel, the weight will decrease rapidly when the instance leaves far away from $\mathbf{x}$. Then we can just give k a relative large value such as $\sqrt{N}$, where N is the size of the training data. Therefore, the only parameter we need to find out is σ, which control the standard deviation of the Gaussian distribution.

In this chapter, we will let $\sigma = \{0.1 \times i\}_{i=1}^{10}$, and set the optimal value of σ to be the one who can achieve the best performance on training dataset. From the experimental results we can see that, in most of the cases, the optimal value of σ will be between 0.2 and 0.4.

3.4 Feature selection by Relief

In kNN method, we need to define the metric distance $D(\cdot)$ between any two instances. In most situations, Euclidean distance is used as the metric distance because its simplicity and wide range suitability. Euclidean distance assigns equal weight to each feature:

$$D(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{I} (\mathbf{x}_1^{(i)} - \mathbf{x}_2^{(i)})^2},$$

where $\mathbf{x}^{(i)}$ is the i^{th} feature of $\mathbf{x}$, and I is the number of features. However, in many cases, some features are irrelevant to the model. Some features are more important than others, and some inferior features will weaken the performance of algorithms. Therefore, we should select good features or assign weights to features before the kNN method.

Relief is considered one of the most successful algorithms for assessing the quality of features, which is proposed by Kira and Rendell (1997). A key idea of the Relief algorithm is to estimate the quality of features according to how well their values distinguish between sample points that are near to each other. Given a randomly selected sample point $\mathbf{x}$, Relief searches for its two nearest neighbors: one is the nearest neighbor from the same class as $\mathbf{x}$, we call it nearest hit: $NH(\mathbf{x})$; another is the nearest neighbor from the different class with $\mathbf{x}$, we call it nearest miss: $NM(\mathbf{x})$. Relief algorithm updates the weights $\{w_i\}_{i=1}^{I}$ for all I features depending on the values of $\mathbf{x}$, $NH(\mathbf{x})$ and $NM(\mathbf{x})$. If $\mathbf{x}$ and $NH(\mathbf{x})$ have different values for feature f, then this feature separates two instances of the same class, which is not desirable and the weight of this feature should decrease. On the other hand, if $\mathbf{x}$ and $NM(\mathbf{x})$ have different values for feature f, then this feature separates two instances of different class, which is desirable and weight of this feature should increase. The algorithm is presented below.

Algorithm 1. Pseudo-code of Relief algorithm (Two-class)

```
1. **Require:** $Data = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, $y_i \in \{-1, +1\}$, set $w_i = 0, 1 \le i \le I$,
   number of iteration $T$.
2. **for** $t = 1$ to $T$ **do**
3. Randomly select an instance $\mathbf{x}$ from $Data$.
```

```
4. Find the nearest hit $NH(\mathbf{x})$ and nearest miss $NM(\mathbf{x})$ of $\mathbf{x}$.
5. **for** $i = 1$ to $I$ **do**
6. Update: $w_i = w_i + |\mathbf{x}^{(i)} - NM^{(i)}(\mathbf{x})| - |\mathbf{x}^{(i)} - NH^{(i)}(\mathbf{x})|$ (where $\mathbf{x}^{(i)}$ means the value of the i-th feature of $\mathbf{x}$).
7. **end for i**
8. **end for t**
```

Therefore, we can obtain the weights of each feature by applying the Relief algorithm to training data, and the distance between two instances can now be computed as:

$$D(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{I} [w_i(\mathbf{x}_1^{(i)} - \mathbf{x}_2^{(i)})]^2} \qquad (3.4.0.5)$$

The ReliefF algorithm (Kononenko, 1994) is proposed as an extension of Relief: from two-class to multi-class. Similar to Relief, ReliefF randomly selects an instance, but then searches for k of its nearest neighbors from the same class, and also k nearest neighbors from each of the different classes.

Algorithm 2. Pseudo-code of ReliefF algorithm (Multi-class)

```
1. **Require:** $Data = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, $y_i \in \{1, 2, \cdots, m\}$, set $w_i = 0, 1 \leq i \leq I$, number of iteration $T$.
2. **for** $t = 1$ to $T$ **do**
```

```
3.  Randomly select an instance $\mathbf{x}$ from $D$.
4.  Find $k$ nearest hit $\{H^{(j)}(\mathbf{x})\}_{j=1}^{k}$
5.  **for** each class $C \neq class(\mathbf{x})$ **do**
6.  from class C find $k$ nearest misses $M_j(C)$
7.  **end for**
8.  **for** $i = 1$ to $I$ **do**
9.  Update: $w_i = w_i - \sum_{j=1}^{k} |\mathbf{x}^{(i)} - H^{(j)}(\mathbf{x})|/(m \cdot k) +$
    $\sum_{C \neq class(x)} \big[\frac{P(C)}{1 - P(class(x))} \sum_{j=1}^{k} |\mathbf{x}^{(i)} - M^{(j)}(C)(\mathbf{x})|\big]/((m \cdot k))$
10. **end for i**
11. **end for t**
```

For ranking data problems, we can treat each ranking as a different class and then apply ReliefF directly. However, when the number of items (J) increases, the number of possible rankings ($J!$) increases as factorial. So in many cases, it might be difficult to find a nearest hit.

In order to solve this problem, we can revise the definitions of nearest hit and nearest miss according to the ordinal structure of ranking. We know that the Kendall distance can measure the difference between two rankings. The normalized Kendall distance equals to 0 when two rankings are identical and equals to 1 when two rankings are reverse to each other.

We define the nearest hit for ranking data to be the nearest neighbor, where the Kendall distance between the rankings of the query instance and the neighbor is less than 0.5. We also define the nearest miss for ranking data to be the nearest neighbor, where the Kendall distance between the rankings of the query instance and the neighbor is greater or equal to 0.5. By these new definitions of nearest hit and miss, we can roughly divide the neighbors into two parts, and it should be more suitable for ranking data problems.

Therefore, we can obtain the weights for features by Algorithm 1 (Relief for two-class), just change the definitions of nearest hit and nearest miss according to Kendall distance. Then a new metric distance can be computed by (3.4.0.5). The weights for each nearest neighbors can also be computed by Gaussian kernel function, and the optimal value of σ will be determined by training data. Finally, we summarize the new algorithm in Algorithm 3.

3.5 Experimental study

3.5.1 Synthetic ranking data

Eleven ranking datasets with different size and different number of ranking items will be used in the experimental study, their properties are summarized in Table

Algorithm 3. Weighted kNN classification for ranking data

1. Let $\{(\mathbf{x}_i, \boldsymbol{\pi}_i)\}, i = 1..N$ be a training ranking dataset, where $\mathbf{x}_i$ is a feature vector and π_i is the corresponding ranking. $\mathbf{x}$ is a new data and its ranking $\boldsymbol{\pi}$ need to be predicted.

2. Apply Relief algorithm to training data as Algorithm 1, the nearest hit and nearest miss is revised according to Kendall distance. The iteration process stops when w_i converges. Then the distance between two instances $\mathbf{x}_1$ and $\mathbf{x}_2$ is computed as:

$$D_w(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{I} [w_i(\mathbf{x}_1^{(i)} - \mathbf{x}_2^{(i)})]^2}$$

3. Let $k = \lfloor \sqrt{N} \rfloor$, find the $k + 1$ nearest neighbor to $\mathbf{x}$ according the distance $D_w(\mathbf{x}, \mathbf{x}_i)$.

4. The $(k+1)th$ neighbor is used for standardization of the k smallest distances via:

$$\Phi(\mathbf{x}, \mathbf{x}_i) = \frac{D_w(\mathbf{x}, \mathbf{x}_i)}{D_w(\mathbf{x}, \mathbf{x}_{k+1})}, 1 \leq i \leq k$$

5. Transform the normalized distances $\Phi(\mathbf{x}, \mathbf{x}_i)$ with Gaussian kernel into weights:

$$\nu_i = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{\Phi(\mathbf{x}, \mathbf{x}_i)^2}{2\sigma^2})$$

The optimal value of σ is obtained by go through the values $0.1, 0.2, \ldots, 0.9, 1.0$, and find the one who can minimize the average Kendall distance between real and predict rankings of training data.

6. The ranking can be predicted by:

$$\hat{\boldsymbol{\pi}} = \arg\min_{\boldsymbol{\pi}} \sum_{i=1}^{k} \nu_i \cdot d(\boldsymbol{\pi}, \boldsymbol{\pi}_i)$$

where $\boldsymbol{\pi}_i$ are rankings of the k nearest instances, $d()$ is the Kendall distance.

3.1. For each dataset, I split it randomly: 70% for training and 30% for testing. I estimate the feature weights w_i from training data by the revised ReliefF algorithm for ranking and find the optimal value of σ. Then I apply Algorithm 3 to predict the ranking for each testing instance, compute the Kendall distance between predict ranking and real ranking. The average Kendall distance will be computed for all testing data, and the whole process will be repeated 50 times (by randomly splitting into training set and testing set). Finally, I use the mean of the average Kendall distances of 50 trials of simulation to evaluate the performance of the model.

I compare the results of five different methods:

1. The original instance-based method described in Section 3.1.
2. Assign weights to features by multi-class ReliefF and then use method 1.
3. Assign weights to features by revised Relief for ranking, and then use method 1.
4. Assign weights to features by revised Relief for ranking, assign weights to neighbors by Guassian Kernel, and then use method 1. (This is the method in Algorithm 3).

The mean of average Kendall distance of 50 trials of simulation for all five methods are listed in Table 3.2. I also list the most frequent value of σ appeared in these 50 trials.

Dataset	item	feature	size
authorship	4	70	841
calhouse	4	4	10000
vehicle	4	18	846
cpu	5	6	8192
fried	5	9	10000
stock	5	5	950
housing	6	6	506
bodyfat	7	7	251
segment	7	18	2310
elevator	9	9	6000
pendigit	10	16	5000
vowel	11	10	528
wisconsin	16	16	194

Table 3.1: The size, number of features and number of ranking items for each dataset

Dataset	knn	ReliefF	Relief.R	Relief.R+Ker	Most frequent σ
authorship	0.0669	0.0679	0.0651	**0.0648**	0.2
calhouse	0.3429	**0.3116**	0.3211	0.3210	0.2
vehicle	**0.0766**	0.0833	0.0797	0.0793	0.3
cpu	0.3596	**0.3530**	0.3559	0.3547	0.2
fried	0.2471	0.2322	0.1669	**0.1660**	0.3
stock	**0.1107**	0.1172	0.1189	0.1171	0.2
housing	0.3173	0.3164	0.3043	**0.3011**	0.4
bodyfat	**0.4502**	0.4597	0.4507	0.4489	0.2
segment	0.0878	0.0873	0.0864	**0.0852**	0.3
elevator	0.2495	0.2831	0.2195	**0.2088**	0.3
pendigit	0.1558	0.1721	0.1550	**0.1486**	0.4
vowel	0.1627	0.1636	0.1574	**0.1501**	0.2
wisconsin	0.4370	0.4456	0.4308	**0.4275**	0.2

Table 3.2: The average Kendall distances for different algorithms (all experiments are repeated 50 times and the results are taken average)

From the result we can see, assigning weights to instances and features can improve the prediction accuracy for most of the ranking data. The revised Relief.R algorithm for ranking data outperforms the multi-class ReliefF algorithm for almost all datasets except calhouse and cpu. For the last four datasets whose number of item J is big (> 8), we see that the multi-class ReliefF algorithm fails to improve from original kNN method. Because the number of possible ranking $J!$ is too large, so it is difficult to find the nearest hit in the original ReliefF algorithm. The revised Relief algorithm for ranking does not have this problem, because the nearest hit and nearest miss can always be found easily.

3.5.2 Real ranking data

In this section I will apply the new model to a real ranking problems. Predicting the expected return of an investment asset is a very difficult task thanks to the complicity of the financial world. However, predicting the ranking/ordering of returns of different asset classes is relative easier. Financial institutions often summarize the historical performance and macroeconomic information to tell investors which asset classes might be better for investing in the following year/season/month.

Data of eight different asset classes are collected from 2001 Jan to 2012 Sept. The eight asset classes are: Asia Pacific Equities, Emerging Market Equities, Cash, US Bonds, Developed Market Equities, Small-Cap Equities, Emerging

Market Bonds and High Yield Bonds. At the end of each month, I use some macroeconomic and market data as factors, to predict the ranking of returns of these eight asset classes for the following month. The twelve factors are: market risk premium, size effect, value effect, momentum effect, risk-free rate, credit premium, aggregate government bond return of US, Canada, Hungary, Germany, Italy and Denmark.

I use the data from 2001 Jan to 2009 Dec for training, and the data from 2010 Jan to 2012 Sept for testing. For each testing month, I apply the original kNN, ReliefF, Relief.R as classifiers to predict the ranking of returns of eight assets for the following month. I list the average Kendall distance in the first column of Table 3.3. Furthermore, I construct five different portfolios: take position of the top-1, top-2, top-3, top-4 and all eight assets with equal weights each month. The average monthly return of all five portfolios are computed, and the results are listed in Table 3.3.

From the results we can see, Relief.R method can achieve a lower average Kendall distance than ReliefF and original knn. For all top $k(k = 1, 2, 3, 4)$ portfolios, the Relief.R method can achieve a higher monthly return than original kNN and ReliefF methods. The average monthly return of eight assets is only 0.67%, or 8.06% annually. But if an investor use the results from Relief.R model to built his investment portfolios of top k(k=1,2,3,4) assets, the annual returns will be $13.32\%, 10.95\%, 11.25\%, 13.04\%$ respectively. Investing by instructions

Algorithm	Ave.Kendall	Top 1	Top 2	Top 3	Top 4	Full Portfolio
knn	0.412	0.9813	0.8389	0.8642	0.7095	0.6717
ReliefF	0.393	1.0396	0.5088	0.9212	0.8539	0.6717
Relief.R	0.376	1.1102	0.9122	0.9379	1.0867	0.6717

Table 3.3: Average monthly returns of different portfolios for different ranking algorithms

from quantitative models can bring more profit than investing blindly.

3.6 Conclusion

In this chapter we improve the instance-based ranking model by assigning weights to neighbors according to their distances and assigning weights to features according to their relative importance. In order to assign weights to features for ranking data, we modify the original Relief algorithm to a new version which is more suitable for ranking data.

From the results of synthetic datasets, we can see that the new algorithm outperforms original methods. Especially when the number of items to be ranked is large, the improvement of the new algorithm is significant. We also apply the algorithm to a real financial ranking data problem. By constructing portfolios according the predict results of the ranking model, investors can get a higher return.

For further research, We will investigate some clustering or adaptive nearest neighbors methods in order to predict rankings more accurately. Many thanks to Zhangyun Li, a former student in Department of Statistics and Actuarial Science in HKU, who provided all the financial data in the real data example.

Chapter 4

Predicting ranking based on error-correcting output codes

4.1 Introduction

Error-correcting output codes (ECOC) (Dietterich and Bakiri, 1995) have been proved to be a very powerful method to improve the classification accuracy for multi-class supervised learning algorithms. The principle idea of ECOC is to decompose multi-class classification problem into several binary classification problems. These binary problems can be solved separately by some traditional binary classifiers. The ECOC framework is composed of two stages: encoding and decoding. In the encoding stage, each class will be mapped into a binary vector with fixed length. In the decoding stage, a binary vector of the same length will be predicted, and then be mapped back to one of the classes.

The essential part of ECOC is the design of coding scheme. The coding scheme will determine the performance of the model. Many schemes have been proposed (Dietterich and Bakiri, 1995) for multi-class classification problems. One of the advantages of ECOC is that the binary problems are relative easier to solve, another advantage is that ECOC can correct some errors caused by the learning algorithms. Therefore, if people could find a good coding scheme, a better prediction can be made to compare to many traditional multi-class classifiers.

In ranking data problems, each class is a ranking/ordering of several items. If we can find a mapping from ranking to binary vector, then ECOC can be applied to solve ranking data problems. In this chapter, I will try several mapping methods for ranking data, and test them with different binary classifiers on different ranking datasets.

4.2 Error-Correcting Code Design

The most important part of ECOC is the design of coding scheme. The error-correcting code is usually defined as a matrix of binary digits, we name this matrix $\mathbf{C}$. The number of columns L is the length of the code; the number of rows K is the number of classes for the multi-class problem. For example, Table 4.1 shows a possible coding matrix for $K = 3$ and $L = 8$.

In matrix $\mathbf{C}$, each row is a code vector: $\mathbf{c}_i = (c_{i1}, c_{i2}, \cdots, c_{iL})$, where $1 \leq i \leq K$ and $c_{ij} = 1$ or 0.

The Hamming distance between two codes $\mathbf{c}_i$ and $\mathbf{c}_j$ are:

$$d_H(\mathbf{c}_i, \mathbf{c}_j) = \sum_q \mathbf{I}(c_{iq} \neq c_{jq}), \tag{4.2.0.1}$$

	Code Matrix							
Class	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
$class1$	0	0	0	0	1	1	1	1
$class2$	0	0	1	1	0	0	1	1
$class3$	0	1	0	1	0	1	0	1

Table 4.1: Example of a coding scheme (code matrix) for a three-class problem, the code for each class contains 8 binary digits.

and the minimum Hamming distance of a code matrix $\mathbf{C}$ is defined as the smallest distance between two different codes in $\mathbf{C}$:

$$d_{min} = \min\{d(\mathbf{c}_i, \mathbf{c}_j) | \mathbf{c}_i \neq \mathbf{c}_j, \mathbf{c}_i, \mathbf{c}_j \in \mathbf{C}\} \tag{4.2.0.2}$$

If a code $\mathbf{v}$ is transmitted and code $\mathbf{w}$ is received, then $\mathbf{w}$ will be decoded to the code in $\mathbf{C}$ whose Hamming distance to $\mathbf{w}$ is smallest. We call this method the nearest neighbor decoding.

Theorem 1. Let $\mathbf{C}$ be a coding matrix with minimum Hamming distance d, assume that the nearest neighbor decoding is used, then:

If $2t < d$, $\mathbf{C}$ can correct t errors caused by transition.

Proof. For any code $\mathbf{w}$ in $\mathbf{C}$, define the r-ball $B_r(\mathbf{w})$ of radius r about $\mathbf{w}$ as:

$$B_r(\mathbf{w}) = \{\mathbf{c} \in \mathbf{C} | d_H(\mathbf{c}, \mathbf{w}) \leq r\}.$$

When $2t < d$, we only need to prove that the t-balls about any two codes are disjoint. If $\mathbf{c}_1 \neq \mathbf{c}_2$ and $\mathbf{w} \in B_t(\mathbf{c}_1) \cap B_t(\mathbf{c}_2)$, then:

$$d_H(\mathbf{c}_1, \mathbf{c}_2) \leq d_H(\mathbf{c}_1, \mathbf{w}) + d_H(\mathbf{w}, \mathbf{c}_2) \leq t + t = 2t < d,$$

which contradicts to the definition of minimum Hamming distance. $\square$

From the above theorem we can see, the ECOC can correct at least $\lfloor \frac{d_{min}-1}{2} \rfloor$ errors. For multi-class classification problems, if the number of binary classifiers that make the wrong prediction is less than $\lfloor \frac{d_{min}-1}{2} \rfloor$, the final prediction will still be correct, because the ECOC has the ability to correct some errors.

Kong and Dietterich (1995) explained why the ECOC method works. They decomposed error rate into two parts: the Bias and the Variance. The Bias is the systematic error, and the Variance is the non-systematic error. Breiman (1994) showed that decision tree models such as C4.5 and CART have high variance, sometimes we say that the decision tree model is unstable. A model is unstable or with high variance means that the model will change significantly if small changes are made to the training data. The Bagging method, which is an ensemble method for decision tree, can reduce the variance. However, it cannot reduce bias because we use the same decision tree model for each Boostrap in bagging. Kong and Dietterich (1995) showed that ECOC can reduce both bias and variance, and they also declared that the ECOC method can be combined with bagging to achieve even better performance.

A good error-correcting output code should satisfy the following two properties:

- Row separation: each code should be well-separated from other codes in the measurement of Hamming distance.
- Column separation: each binary position function f_i should be independent with other binary position functions $f_j, j \neq i$. This property can be evaluated by checking whether the Hamming distance between column i and other columns is large and whether the Hamming distance between column i and the complement of other columns is also large.

The row separation will determine the power of a code to correct errors. The reason that we require column separation is: if two columns i and j are too similar, then any binary classifier can be applied to these two columns will make similar results (mistakes). If two columns are complementary to each other, these binary classifiers will also make similar results (mistakes) because of the symmetry between 0 and 1. And we also don't like the columns in which all digits are 0s or 1s.

Different coding schemes have been proposed in the literature over the years, the choice of different method is mainly based on the number of classes K in the multi-class problem. The most widely used two schemes, for small and large value of K, are exhaustive codes and randomized hill climbing codes respectively.

- **Exhaustive Code:** When K is relative small, usually when $K \leq 8$, we can construct a code of length $2^{K-1} - 1$. Row 1 is all ones. First half of row

	Code Matrix															
Class	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
c_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
c_2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
c_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
c_4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
c_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 4.2: Example of a exhaustive codes for a five-class problem

2 are zeros and the following are ones. Row 3 have 2^{K-3} zeros, followed by 2^{K-3} ones, and followed by 2^{K-3} zeros, and followed by 2^{K-3} ones. In row i, there are alternating runs of 2^{K-i} zeros and ones. Table 4.2 shows the exhaustive codes for a five-class problem.

- **Randomized Hill Climbing Code:** When K is bigger, the exhaustive code will generate a very long binary vector. In this case, randomized hill climbing coding is proposed. We can draw K random codes of a desired length L. Any pair of these random codes should be separated by a Hamming distance which follows a binomial distribution with mean equals $L/2$. In order to do this, the algorithm constantly finds the pair of rows which have the smallest Hamming distance (too close) and the pair of columns which have the biggest Hamming distance (too far apart). Then the algorithm finds out the four bits where the two rows and two columns intersect, changes them to improve the row and column separations. In most of the cases, people first set the length of code to a moderate size, increase it step by step until the performance do not change too much. Allwein et al. (2002) suggested that $L = 10 \log K$ should be a good choice.

Many other coding schemes have been proposed, more details can refer to Escalera et al. (2010). Many new ECOC methods have been proposed for different purposes, Bagheri et al. (2013) used a three-dimension code-matrix to improve the predicting accuracy, Bautista et al. (2012) proposed a minimal design of ECOC in order to save the computing time.

After encoding the multi-class to binary vector with length L, we can use any binary classifiers to solve these binary classification problems. For example, decision tree, logistic regression, neural network and so on. For any testing instance, we use the binary classifier to compute the probabilities that each bit equals to one, call the probability vector $\mathbf{p} = \langle p_1, p_2, \ldots, p_L \rangle$. In order to make prediction, the probability vector need to compare with the codes for K classes, $\mathbf{c}_i (i = 1, 2, \ldots, K)$. The most straightforward way is to compute the L^1 distance between $\mathbf{p}$ and $\mathbf{c}_i$:

$$L^1(\mathbf{p}, \mathbf{c}_i) = \sum_{j=1}^{L} |p_j - c_{i,j}|$$

The class who has the smallest L^1 distance to $\mathbf{p}$ is assigned to the testing data. When tie appears, just randomly choose one class.

4.3 Error-correcting code for ranking data

In ranking data problem, the output of an instance is not a class, but a ranking vector $\boldsymbol{\pi}$. In order to apply ECOC methods, we need to design the coding scheme for ranking data. We can apply the two coding schemes mentioned in last section directly to form four ranking data coding schemes:

1. **Multi-class randomized hill climbing code**

   The most straightforward method is treating the ranking problem as a multi-class problem. If we rank J items, there will be $K = J!$ possible rankings. We can design a $J!$-classes coding scheme. Since $J!$ is always large, the randomized hill climbing codes should be a good choice. One weakness of this method is that the number $J!$ will be extremely large when J increases, another weakness is that it ignores the structure of ranking.

2. **Rank position code**

   For a ranking of J items, we can map each rank to a binary vector by exhaustive codes, and connect these binary vectors together to form the code for a ranking. For example, when $J = 5$ and $\boldsymbol{\pi} = (2, 3, 1, 5, 4)$, the corresponding binary code has length $L = 15 \times 5$. The first 15 bits is the second line (rank2) in Table 4.2, the following 15 bits is the third line (rank3) in Table 4.2, and the following three 15 bits are the first line, fifth line and forth line in Table 4.2 respectively. Then we will build a coding scheme with length $L = J(2^{J-1} - 1)$.

When J is large, the length L is also very large.

3. **Pairwise comparison code**

Another coding method is using the pairwise comparison model. To rank J items, we can construct $J(J-1)/2$ item pairs. Therefore, each bit in the code is just the relation between two items. The code $B = \langle b_1, \ldots, b_L \rangle$ where $b_n = 1$ if $\pi(i) > \pi(j)$ and $b_n = 0$ if $\pi(i) < \pi(j)$, and $L = J(J-1)/2$.

4. **Revised rank position code**

Ranking data has a special property that the J items are ordinal structured: rank 1 is the best and rank J is the worst. Also, the difference between rank 1 and rank J is bigger than the difference between rank $J-1$ and rank J. Therefore I propose to revise the rank position code in order to reflect the ordinal structure property. The new binary vector for each rank has length $J-1$. The binary vector for rank 1 is all composed of ones, the vector for rank 2 composed of $J-2$ ones and 1 zero, and so on. Finally the vector for rank J is all composed of zeros.

I proposed two coding schemes which satisfy the above criterion, and I list them in table 4.3 and 4.4 (for example when $J = 6$).

The advantage of the first coding scheme is: the Hamming distance between and two ranks is just the difference between two ranks. However, the column 1 contains one 1 and five 0s. This imbalanced data will make binary classification difficult, but we can use some over-sampling techniques to solve this problems.

	Code Matrix				
Rank	f_1	f_2	f_3	f_4	f_5
$rank = 1$	1	1	1	1	1
$rank = 2$	0	1	1	1	1
$rank = 3$	0	0	1	1	1
$rank = 4$	0	0	0	1	1
$rank = 5$	0	0	0	0	1
$rank = 6$	0	0	0	0	0

Table 4.3: Rank position coding scheme 1 for J=6

	Code Matrix				
Rank	f_1	f_2	f_3	f_4	f_5
$rank = 1$	1	1	1	1	1
$rank = 2$	1	1	0	1	1
$rank = 3$	1	0	1	0	1
$rank = 4$	0	1	0	1	0
$rank = 5$	0	0	1	0	0
$rank = 6$	0	0	0	0	0

Table 4.4: Rank position coding scheme 2 for J=6

The advantage of the second coding scheme is: since 0 and 1 are interlapped distributed, the number of 0 and 1 in each column are the same, which makes binary classification easier. But the Hamming distance between ranks does not equal to the difference between ranks.

The two coding schemes for any number of items J can be derived as following:

- **Coding scheme 1.** The code for rank i consists of $J-1$ digits. The first $i-1$ digits are 0s, and the following $J-i$ digits are 1s.

- **Coding scheme 2.** The code for rank 1 consists of $J-1$ 1s. The $\lfloor \frac{J}{2} \rfloor$-th digit for rank 2 is 0, and others are 1s. For rank 3, the middle 3 digits are complementary of the middle 3 digits of rank 2, and other digits are the same as rank 2. For rank 4, the middle 5 digits are complementary of the middle 5 digits of rank 3, and other digits are the same as rank 3. In general, when $i \leq \lceil \frac{J}{2} \rceil$, the middle $2i-3$ digits are the complementary of the middle $2i-3$ digits of rank $i-1$, and other digits are the same as rank $i-1$. When $i > \lceil \frac{J}{2} \rceil$, the code of rank i is just the complementary of code of rank $J+1-i$.

For multi-class problems, we analyze what will happen if some errors exist. In ranking data problem, since we will evaluate the accuracy of prediction by Spearman/Kendall distance, then it is more suitable to analyze the change of Spearman/Kendall distance if some errors happen.

Theorem 1. We assume that exactly one digit is wrongly predicted for each rank and the nearest Hamming distance decoding method is used, then the expected Spearman distance between predict ranking and real ranking for coding scheme 1 and 2 are:

$$\mathbf{E}(\text{Spearman distance}) = \frac{12(2J-3)^2}{(J-1)^3J^2(J+1)} \text{ for Code 1}$$
$$\mathbf{E}(\text{Spearman distance}) = \frac{12(2J-1)^2}{(J-1)^3J^2(J+1)} \text{ for Code 2}$$

Proof. For code scheme 1. We can see the last line (rank=6), if the last digit is wrongly predicted, then the code [00001] will be wrongly predicted to rank=5, so the difference of rank is 1. If the 5-th digit is wrongly predicted, then the code [00010] will be predicted to rank=4 with 0.5 probability and be predicted to rank=6 with 0.5 probability, and we can calculate that the expected difference of rank is $0.5 \times 2 + 0.5 \times 0 = 1$. If any of the first three digits is wrong predicted, the rank=6 will be predicted, and there is no mistake for this. So, for the last line of the code matrix, if one digit is wrong, the expected difference of rank should be:

$$\frac{1}{5}(0+0+0+1+1) = \frac{2}{J-1}.$$

We can do this for all J rows, we find that, for row 1 and row J, the expected difference of rank are $\frac{2}{J-1}$. For row 2 and row $J-1$, the expected difference of rank are $\frac{3}{J-1}$. For all other rows, the expected difference of rank are $\frac{4}{J-1}$. And since all J ranks will appear with equal probabilities, the expected difference of the rank for one position is:

$$\frac{2+2+3+3+4(J-4)}{(J-1)J} = \frac{4J-6}{J(J-1)}.$$

From the definition of Spearman distance (normalized):

$$d_s(\pi_1, \pi_2) = \frac{3\sum_{i=1}^{J}[\pi_1(i) - \pi_2(i)]^2}{J(J^2-1)}$$

We can compute that the expected Spearman distance between real ranking and predict ranking should be:

$$\frac{3 \times J \times (\frac{4J-6}{J(J-1)})^2}{J(J^2-1)} = \frac{12(2J-3)^2}{(J-1)^3 J^2 (J+1)}.$$

A similar method can be used to prove the formula for code scheme2. $\square$

We plot the expected Spearman distance for these two code schemes in Figure 4.1. From the figure we can see, code 1 is better than code 2 by theory. But when $J > 10$, the difference between code 1 and code 2 becomes very small. We assume that exactly one error happens in one rank position, but the number of errors is difficult to control. From the experiment studies in next section, we can see that, in most of the cases, code scheme 2 is better than code scheme 1. I think the main reason is that in code 2, each column is balance, so the binary models will be more accurate.

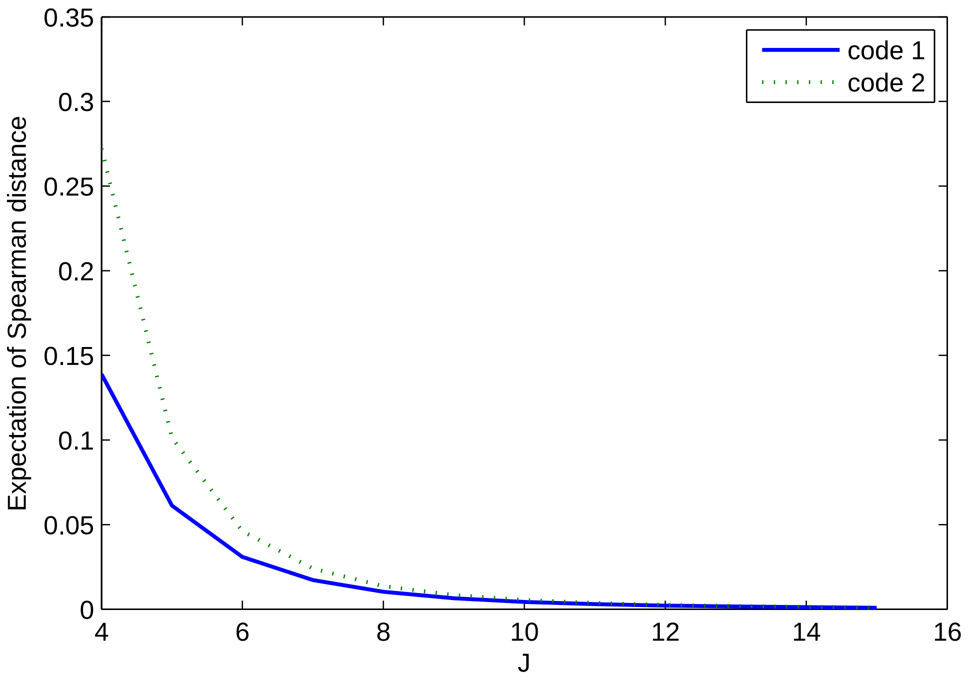

Figure 4.1: Expected Spearman distance for two coding schemes

4.4 Experimental study

Ten ranking datasets with different sizes and different number of items will be used in the experimental study. For each dataset, we split it randomly: 70 percent for training and 30 percent for testing. We use the five decoding schemes we proposed in last Section: multi-class randomized hill climbing codes, Rank position codes, Pairwise codes, and the revised rank position code schemes 1 and 2. And we use three different binary classifiers: C4.5(Decision Tree), logistic regression, and bagging with trees. The three classifiers are typical models for three model categories: nonlinear, linear and ensemble.

For each dataset, we try five coding schemes and three binary classifiers. For each combination, we repeat the training/testing splitting 20 times, and compute the mean of average Kendall distance between real and predicted rankings for testing data. The results are summarized in Tables 4.5 and 4.6.

From the results we can see, the revised rank position coding scheme 2 with Bagging is the best combination for all datasets except Authorship. Bagging is better than C4.5 and logistic regression.

We also compare the ECOC results (using the revised rank position coding scheme 2 and Bagging) with the results of some traditional ranking models. The first one is the instance-based model, which finds the nearest neighbors and uses the rankings of these neighbors and distance-based model to make prediction.

Dataset	Classifier	MC.code	RP.code	PW.code	RP1.code	RP2.code
Authorship	C4.5	0.108	0.104	0.081	0.094	0.092
	Logit.R	0.102	0.099	0.079	0.092	0.090
	Bagging	0.093	0.089	0.066	0.084	0.080
Vehicle	C4.5	0.095	0.083	0.079	0.083	0.080
	Logit.R	0.093	0.082	0.081	0.079	0.078
	Bagging	0.091	0.081	0.080	0.072	0.069
Cpu	C4.5	0.360	0.358	0.353	0.351	0.350
	Logit.R	0.363	0.358	0.355	0.353	0.353
	Bagging	0.361	0.353	0.350	0.343	0.341
Fried	C4.5	0.211	0.191	0.154	0.123	0.121
	Logit.R	0.242	0.223	0.219	0.190	0.184
	Bagging	0.231	0.213	0.201	0.103	0.098
Stoke	C4.5	0.209	0.181	0.165	0.123	0.119
	Logit.R	0.181	0.178	0.168	0.130	0.123
	Bagging	0.178	0.166	0.162	0.099	0.090

Table 4.5: Average Kendall Distance for different coding schemes and different binary classifiers. MC.code is the multi-class Randomized Hill Climbing Codes, RP.code is the Rank Position Codes, PW.code is the Pairwise codes, RP1.code and RP2.code are the two revised rank position codes.

Dataset	Classifier	MC.code	RP.code	PW.code	RP1.code	RP2.code
Housing	C4.5	0.336	0.327	0.319	0.286	0.279
	Logit.R	0.339	0.328	0.327	0.280	0.271
	Bagging	0.332	0.331	0.312	0.267	0.259
Bodyfat	C4.5	0.457	0.454	0.449	0.446	0.446
	Logit.R	0.458	0.452	0.446	0.444	0.442
	Bagging	0.452	0.449	0.441	0.440	0.438
Segment	C4.5	0.113	0.100	0.098	0.071	0.062
	Logit.R	0.110	0.099	0.091	0.072	0.069
	Bagging	0.101	0.093	0.088	0.066	0.052
Elevator	C4.5	-	-	0.234	0.201	0.199
	Logit.R	-	-	0.229	0.199	0.191
	Bagging	-	-	0.202	0.175	0.169
Pendigit	C4.5	-	-	0.179	0.151	0.143
	Logit.R	-	-	0.172	0.160	0.141
	Bagging	-	-	0.154	0.123	0.113

Table 4.6: Average Kendall Distance for different coding schemes and different binary classifiers. MC.code is the multi-class Randomized Hill Climbing Codes, RP.code is the Rank Position Codes, PW.code is the Pairwise codes, RP1.code and RP2.code are the two revised rank position codes. The MC.code and RP.code for datasets Elevator and Pendigit are not computed because that big J will cost too much computing time for these two codes.

Dataset	ECOC	Instance-Based	ROL	RankTree
authorship	0.080	**0.066**	0.103	0.081
vehicle	**0.069**	0.076	0.111	0.091
cpu	**0.341**	0.359	0.371	0.360
fried	**0.098**	0.243	0.245	0.211
stock	**0.090**	0.111	0.165	0.133
housing	**0.259**	0.317	0.352	0.321
bodyfat	**0.438**	0.450	0.461	0.455
segment	**0.052**	0.087	0.103	0.091
elevator	**0.169**	0.249	0.301	0.281
pendigit	**0.113**	0.155	0.203	0.158

Table 4.7: Average Kendall Distance for different ranking models. The best results are in bold numbers.

The second one is the rank-ordered-logit model. Rank-ordered-logit (Beggs et al., 1981) is a very traditional ranking model, the model assumes that judges rate each item with a measurable utility, and the ordering of a judge's utilities determines the ranks of items assigned by the judge. The third one is Yu's (2009) decision tree model for ranking. Since we evaluate the performance of classifier by average Kendall distance, we assign a ranking to each leaf node which can minimize the total Kendall distance in that node. All results are summarized in Table 4.7.

From the results we can see, the ECOC model outperforms other models significantly for most of the datasets.

4.5 Conclusion

In this chapter we propose some mapping from ranking to binary vector. Then we can solve the ranking data problem via Error-correcting output codes, which is a widely used algorithm for multi-class problems. The revised version of rank position coding scheme can reflect the ordinal structure of ranking and its relative short compare to other coding schemes.

From the experiment results we can see, ECOC can achieve a good prediction accuracy compare to some existing ranking models. The revised rank position coding scheme with Bagging as binary classifier achieves the best results for most of the datasets. This method works for ranking data with different size, different number of features and different number of items.

For further investigation, we hope to find some other coding schemes which are more suitable for ranking data. And we also hope to find some methods to deal with partial ranking problems.

Chapter 5

Mixtures of factor analyzers for heterogeneous ranking data

5.1 Introduction

Over the past decade, many models for ranking data have been proposed, see (Marden, 1995; Yu, 2003) for a review of the literature. In many situation, the purpose for analyzing ranking data is to identify factors which will affect judges' preference behavior. Factor analysis model (Thurstone, 1947) is widely used in marketing researches and social science. This model can find the unobserved common characteristics among a set of variables. Yu et al. (2005) used the factor model to modeling ranking data, and applied it to a real survey data in order to investigate factors which affect people's attitudes to job hunting.

Most of the ranking models assume a homogeneous population of judges. However, in many cases, the population of judges is more likely to be heterogeneous. Mixture model can solve this problem by assuming that the population is composed of several homogeneous sub-populations. The distribution within a sub-population can be modeled by any traditional ranking model. More flexibility to ranking data models can be obtained by using mixture models. Marden

(1995) first applied mixture models to ranking data, Murphy and Martin (2003) applied mixture models to distance-based model and explained the heterogeneity among judges. Gormley and Murphy (2008) applied mixture models to experts model for analyzing the Irish president election. Lee and Yu (2012) proposed the mixture models to weighted distance-based models.

Partial ranking appears with higher frequency than full ranking, because judges usually only give their top k choices. Busse et al. (2007) extended (Murphy and Martin, 2003)'s mixture model to deal with partial rankings, Yu et al. (2005)'s factor model can also deal with partial rankings.

Mixture of factor analysis (MFA) models are widely applied in recent years (Ghahramani and Hinton, 1996; McLachlan et al., 2003, 2007). MFA can perform clustering and dimension reduction simultaneously, and meanwhile provides some descriptions of factors in each sub-population. Most of the mixture models are fitted by EM algorithm (Dempster et al., 1977). Zhao and Yu (2008) proposed a fast expectation conditional maximization (ECM) algorithm which can converge much faster than classical EM algorithm.

It is natural to apply MFA for ranking data, and it will be more appropriate when the population is heterogeneous. However, MFA cannot be directly used for ranking data due to the discrete ordinal structures of rankings. In this chapter, we will propose a MFA model for ranking data and show the advantages of the model.

This chapter is organized as follows. Section 5.2 gives the notations of ranking data and a brief introduction of factor analysis model. Section 5.3 proposes the MFA model for ranking data. Section 5.4 illustrates how to fit the model by MCEM approach. In Section 5.5, I will give some simulation studies results. Two real ranking datasets will be tested in the new model in Section 5.6. Section 5.7 will give the conclusions and talk about some future works.

5.2 Factor analysis for ranking data

Suppose a judge from the population will be asked to rank J items according to his/her preference. In the factor model, we assume that the ordering of the J items are determined by the ordering of J latent utilities $\mathbf{x} = (x_i, \cdots, x_J)$. The J-dimensional utility vector $\mathbf{x}$ should satisfy the q-factor model:

$$\mathbf{x} = \mathbf{A}\mathbf{y} + \boldsymbol{\mu} + \boldsymbol{\epsilon}, \tag{5.2.0.1}$$

where $\boldsymbol{\mu}$ is a J-dimensional mean vector, $\mathbf{A}$ is a $J \times q$ factor loading matrix, $\mathbf{y} \sim \mathbf{N}(\mathbf{0}, \mathbf{I})$ is a q-dimensional latent factor vector, and $\boldsymbol{\epsilon} \sim \mathbf{N}(\mathbf{0}, \boldsymbol{\Psi})$ is the error vector where $\boldsymbol{\Psi}$ is a diagonal matrix.

A ranking of these J items, denoted by $\boldsymbol{\pi} = (\pi_i, \cdots, \pi_J)$, is defined as a permutation of 1 to J. We denote π_i the rank of item i. Smaller ranks corresponding to more preferred items and have bigger utilities. For example, if $\boldsymbol{\pi} = (2, 3, 1)$, it corresponds to an unobserved utility vector: $\mathbf{x} = (x_1, x_2, x_3)$ with $x_3 > x_1 > x_2$.

The traditional factor model can be solved quickly by EM algorithm. However, if we apply it to ranking data, the utility vector $\mathbf{x}$ must be consistent with the ranking $\boldsymbol{\pi}$, and it will bring more difficulty during the E-step. Yu et al. (2005) solved this by applying the Gibbs sampling method in the E-step of the EM algorithm.

5.3 MFA model

The MFA model is a mixture of M factor models with mixture proportions $\{p_j\}_{j=1}^{M}$, and $\sum_{j=1}^{M} p_i = 1$. After the component label j and the ranking $\boldsymbol{\pi}$ are given, we can generate $\mathbf{x}$ from a q_j-factor FA model with parameter $\boldsymbol{\theta}_j = (\boldsymbol{\mu}_j, \mathbf{A}_j, \boldsymbol{\Psi}_j)$:

$$\begin{cases} \mathbf{x}|\boldsymbol{\pi}, j = \mathbf{A}_j\mathbf{y}_j + \boldsymbol{\mu}_j + \boldsymbol{\epsilon}_j; \\ \mathbf{y}_j \sim \mathbf{N}(\mathbf{0}, \mathbf{I}), \boldsymbol{\epsilon}_j \sim \mathbf{N}(\mathbf{0}, \boldsymbol{\Psi}_j) \end{cases} \tag{5.3.0.2}$$

where $\boldsymbol{\mu}_j$ is a J-dimensional mean vector, $\mathbf{A}_j$ is the $J \times q_j$ factor loading matrix, $\mathbf{y}_j$ is a q_j-dimensional factor vector and $\boldsymbol{\Psi}_j$ is a positive diagonal matrix.

Both the probability density distribution of $\mathbf{x}$ and the conditional probability density distribution of $\mathbf{y}_j$ given $\mathbf{x}$ and label j are the multivariate normal distributions:

$$\mathbf{x}|j \sim \mathbf{N}_J(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{5.3.0.3}$$

$$\mathbf{y}_j|\mathbf{x}, j \sim \mathbf{N}_{q_j}(\mathbf{A}_j'\boldsymbol{\Sigma}_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j), \mathbf{M}_j^{-1}), \tag{5.3.0.4}$$

where

$$\mathbf{\Sigma}_j = \mathbf{A}_j\mathbf{A}_j' + \mathbf{\Psi}_j, \mathbf{M}_j = \mathbf{I} + \mathbf{A}_j'\mathbf{\Psi}_j^{-1}\mathbf{A}_j \tag{5.3.0.5}$$

The log likelihood of the model is:

$$L(\{\boldsymbol{\pi}_n\}|\boldsymbol{\theta}) = \sum_n \ln\big[\sum_j p_j \mathrm{Pr}(\boldsymbol{\pi}_n|\boldsymbol{\theta}_j)\big], \tag{5.3.0.6}$$

where $\boldsymbol{\theta}_j = (\mathbf{A}_j, \boldsymbol{\mu}_j, \mathbf{\Psi}_j)$.

5.4 A two-stage EM algorithm for finding the parameters

In this section, I propose an EM-like algorithm to find the MLE of the parameter $\boldsymbol{\theta}$ in the MFA model. Let $\mathbf{\Pi} = \{\boldsymbol{\pi}_n\}_{n=1}^N$, $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$, $\mathbf{Y} = \{\mathbf{y}_n\}_{n=1}^N$, and $\mathbf{Z} = \{\mathbf{z}_n\}_{n=1}^N$, where $\mathbf{z}_n = (z_{n1}, \ldots, z_{nM})$ and z_{nj} is an indicator variable taking the value 1 if $\mathbf{x}_n$ comes from the j-th component, and 0 otherwise.

We use a two-stage EM algorithm. In the first stage of EM, only latent label vectors $\mathbf{Z}$ and unobserved data $\mathbf{X}$ are treated as missing data and the complete log likelihood of the augmented complete data $(\mathbf{\Pi}, \mathbf{X}, \mathbf{Z})$, is given by:

$$L_1(\mathbf{\Pi}, \mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \sum_{n=1}^N \sum_{j=1}^M z_{nj} \ln\{p_j \mathrm{Pr}(\mathbf{x}_n|\boldsymbol{\theta}_j)\}. \tag{5.4.0.7}$$

- **E-step of stage 1:** Compute the expected L_1 given $\mathbf{\Pi}$ and $\boldsymbol{\theta}_0$:

$$Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}_0) = \sum_{j=1}^{M}\sum_{i=1}^{N} R_{nj}(\boldsymbol{\theta}_0)\mathbf{E}[\ln\{p_j\Pr(\mathbf{x}_n|\boldsymbol{\theta}_{0j})\}|\boldsymbol{\pi}_n, j]. \tag{5.4.0.8}$$

Here, $R_{nj}(\boldsymbol{\theta}_0)$ is the posterior possibility of data point $\mathbf{x}_n$ belonging to component j given $\boldsymbol{\theta}_0$:

$$R_{nj}(\boldsymbol{\theta}_0) \triangleq \frac{p_j\Pr(\mathbf{x}_n|\boldsymbol{\theta}_{0j})}{\sum_{k=1}^{M} p_k\Pr(\mathbf{x}_n|\boldsymbol{\theta}_{0k})} \tag{5.4.0.9}$$

- **M-step of stage 1:** Maximize Q_1 w.r.t. p_j's and $\boldsymbol{\mu}_j$'s given $(\mathbf{A}_j, \boldsymbol{\Psi}_j)$'s under the restriction $\sum_{i=1}^{M} p_j = 1$, which is easy and similar to that in conventional gaussian mixture model, leading to the following updating equations:

$$\begin{aligned} \widetilde{p}_j &= \frac{1}{N}\sum_{n=1}^{N} R_{nj}(\boldsymbol{\theta}_0), \\ \widetilde{\boldsymbol{\mu}}_j &= \frac{1}{N\widetilde{p}_j}\sum_{n=1}^{N} R_{nj}(\boldsymbol{\theta}_0)\mathbf{E}(\mathbf{x}_n|\boldsymbol{\pi}_n, j). \end{aligned} \tag{5.4.0.10}$$

In the second stage of our EM, we consider Q_1 as our interested log likelihood. We wise to increase Q_1, rather than maximize it. This corresponds to a generalized EM algorithm. To this end, we introduce $\mathbf{Y}$ as missing data. The complete log likelihood L_2 of the complete data $(\mathbf{\Pi}, \mathbf{X}, \mathbf{Y}, \mathbf{Z})$ is given by:

$$L_2(\mathbf{\Pi}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}|\boldsymbol{\theta}) = \sum_{n=1}^{N}\sum_{j=1}^{M} R_{nj}(\boldsymbol{\theta}_0)\ln\{p_j\Pr(\mathbf{x}_n, \mathbf{y}_n|\boldsymbol{\theta}_j)\}. \tag{5.4.0.11}$$

- **E-step of stage 2:** Given $\boldsymbol{\Psi}$ and $(\widetilde{p}_j, \widetilde{\boldsymbol{\mu}}_j, \mathbf{A}_j, \boldsymbol{\Psi}_j)$'s, compute L_2 w.r.t. the distribution of $\Pr(\mathbf{X}, \mathbf{Z}|\mathbf{\Pi})$:

$$Q_2(\boldsymbol{\theta}|\boldsymbol{\theta}_0) = \sum\nolimits_{j=1}^{M} \sum\nolimits_{n=1}^{N} R_{nj}(\boldsymbol{\theta}_0)\mathbf{E}\big[\mathbf{E}[\ln\{p_j \Pr(\mathbf{x}_n, \mathbf{y}_n|\boldsymbol{\theta}_{0j})\}|\mathbf{x}_n, j]\big|\boldsymbol{\pi}_n, j], \tag{5.4.0.12}$$

where the involved expectations $\mathbf{E}[\mathbf{y}_n|\mathbf{x}_n, j]$ and $\mathbf{E}[\mathbf{y}_n\mathbf{y}'_n|\mathbf{x}_n, j]$ can be easily obtained from:

$$\mathbf{E}[\mathbf{y}_n|\mathbf{x}_n, j] = \mathbf{A}'_j\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_j), \tag{5.4.0.13}$$

$$\mathbf{E}[\mathbf{y}_n\mathbf{y}'_n|\mathbf{x}_n, j] = \mathbf{M}_j^{-1} + \mathbf{E}[\mathbf{y}_n|\mathbf{x}_n, j]\mathbf{E}[\mathbf{y}_n|\mathbf{x}_n, j]'. \tag{5.4.0.14}$$

- **M-step of stage 2:** Maximize Q_2 w.r.t. $\mathbf{A}_j$'s and $\boldsymbol{\Psi}_j$'s, and we can get:

$$\widetilde{\mathbf{A}_j} = \mathbf{S}_j\boldsymbol{\Psi}_j^{-1}\mathbf{A}_j(\mathbf{I} + \mathbf{A}'_j\boldsymbol{\Sigma}_j^{-1}\mathbf{S}_j\boldsymbol{\Psi}_j^{-1}\mathbf{A}_j)^{-1}, \tag{5.4.0.15}$$

$$\widetilde{\boldsymbol{\Psi}_j} = \mathrm{diag}\{\mathbf{S}_j - \mathbf{S}_j\boldsymbol{\Sigma}_j^{-1}\mathbf{A}_j\mathbf{A}'_j\}, \tag{5.4.0.16}$$

where $\boldsymbol{\Sigma}_j$ is given by (2.3.0.5) and $\mathbf{S}_j$ is given by:

$$\mathbf{S}_j = \frac{1}{N\widetilde{p}_j}\sum\nolimits_{n=1}^{N} R_{nj}(\boldsymbol{\theta}_0)\mathbf{E}[(\mathbf{x}_n - \boldsymbol{\mu}_j)(\mathbf{x}_n - \boldsymbol{\mu}_j)'|\boldsymbol{\pi}_n, j] \tag{5.4.0.17}$$

It can be seen that in this two-stage EM algorithm, the E-step to compute the first and second moments of $\mathbf{x}_n$ given $\boldsymbol{\pi}_n$ and j is required to perform once only. As this step is expensive to compute, compared to the AECM algorithm that performs this step twice in each iteration, our two-stage EM is more efficient. All the expectations within the computation of $\mathbf{S}_j, \widetilde{\mathbf{A}_j}$ and $\widetilde{\boldsymbol{\Psi}_j}$ can be obtained by GHK simulators (Hajivassiliou and McFadden, 1998) with higher efficiency, the details of GHK is given in Appendix of this chapter.

5.5 Simulation Studies

In this section, we will use simulation ranking data to demonstrate that: The BIC criteria is effective in selecting the number of components and number of factors; The MFA model can increase the prediction accuracy as training data size increases; The MFA model outperforms MDB (mixture of distance-based) model for both full and partial rankings.

5.5.1 Model selection by different criteria

First, we will generate a ranking dataset of 15 items by factor model. We generate the ranking dataset with 2 components and 2 factors for each component, the parameters of the factor model are listed in Table 5.1 (the factor loadings are the same for two components).

Then, we will find out which criteria can effectively select the true parameters of a MFA model. I generate 500 rankings for each component. For the simulation data, I fit MFA models with different number of components and number of factors, and select the best combination by three different criteria: Bayesian information criterion (BIC), Akaike information criterion (AIC), Integrated complete likelihood (ICL). Definitions of these criterions are:

$$\mathrm{BIC} = 2L - v\ln(n),$$

$$\mathrm{AIC} = 2L - 2v,$$

Item	$\mathbf{A_{11}}$	$\mathbf{A_{12}}$	$\mathbf{A_{21}}$	$\mathbf{A_{22}}$	$\mathbf{b_1}$	$\mathbf{b_1}$
1	2.8	0.8	2.8	0.8	0.2	3.0
2	2.8	-0.8	2.8	-0.8	0.4	2.8
3	2.8	0.8	2.8	0.8	0.6	2.6
4	2.8	-0.8	2.8	-0.8	0.8	2.4
5	2.8	0.8	2.8	0.8	1.0	2.2
6	2.8	-0.8	2.8	-0.8	1.2	2.0
7	2.8	0.8	2.8	0.8	1.4	1.8
8	2.8	-0.8	2.8	-0.8	1.6	1.6
9	2.8	0.8	2.8	0.8	1.8	1.4
10	2.8	-0.8	2.8	-0.8	2.0	1.2
11	2.8	0.8	2.8	0.8	2.2	1.0
12	2.8	-0.8	2.8	-0.8	2.4	0.8
13	2.8	0.8	2.8	0.8	2.6	0.6
14	2.8	-0.8	2.8	-0.8	2.8	0.4
15	2.8	0.8	2.8	0.8	3.0	0.2

Table 5.1: Simulation parameters of factor model

$$\text{ICL} = \text{BIC} + 2\sum_{i=1}^{n}\sum_{j=1}^{M} \widehat{z}_j^{(i)} \ln \widehat{z}_j^{(i)},$$

where L is the log-likelihood, v is the number of free parameters, n is the data size and the $\widehat{z}$ values are the latent variables generated in the E-step of the algorithm. I repeat the simulation and model selection process 100 times, then count the frequency of best number of components and factors for all three criterions. The results are listed in Table 5.2. From the results we can see, BIC criterion select the true model with the highest accuracy, which is 74%. AIC and ICL criterions almost never choose the right model, AIC criterion prefers models with more components and ICL criterion prefers models with less components. Therefore, the BIC criterion is effective for selecting the true MFA model.

5.5.2 Prediction accuracy for full/partial ranking

We will verify the prediction accuracy of MFA by two approaches. Firstly we will show that the log-likelihood of testing data will increase if we increase the size of the training data; secondly we will compare the log-likelihood of testing data for MFA and MDB models. Furthermore, we demonstrate that our MFA model can also deal with partial rankings (top-k ranking), and the log-likelihood of testing data of MFA model also outperforms that of MDB model.

We use the same parameters in Section 5.5.1 to generate ranking datasets with 2 components and 2 factors. We use $100, 200, \ldots, 1000$ rankings for training

BIC	factor=0	factor=1	factor=2	factor=3
Component=1	0	0	5	1
Component=2	0	0	74	3
Component=3	0	17	0	0
AIC	**factor=0**	**factor=1**	**factor=2**	**factor=3**
Component=1	0	0	0	0
Component=2	0	0	1	1
Component=3	0	3	68	27
ICL	**factor=0**	**factor=1**	**factor=2**	**factor=3**
Component=1	0	0	78	22
Component=2	0	0	0	0
Component=3	0	0	0	0

Table 5.2: Model selection frequency of different criterions: BIC, AIC and ICL

and 1000 rankings for testing. For both MFA and MDB models, we plot the log-likelihood of testing data for models trained by different number of training data. We do the same thing for partial rankings (top 5 and top 10 rankings) which are derived from the full rankings, and plot the log-likelihood for testing data for two models. All the results are showed in Figure 5.1.

From the results we can see that, for both full and partial rankings, the log-likelihood of testing data will increase if we increase the number of training data. For all the situations, MFA model can obtain a better (higher) likelihood than MDB model.

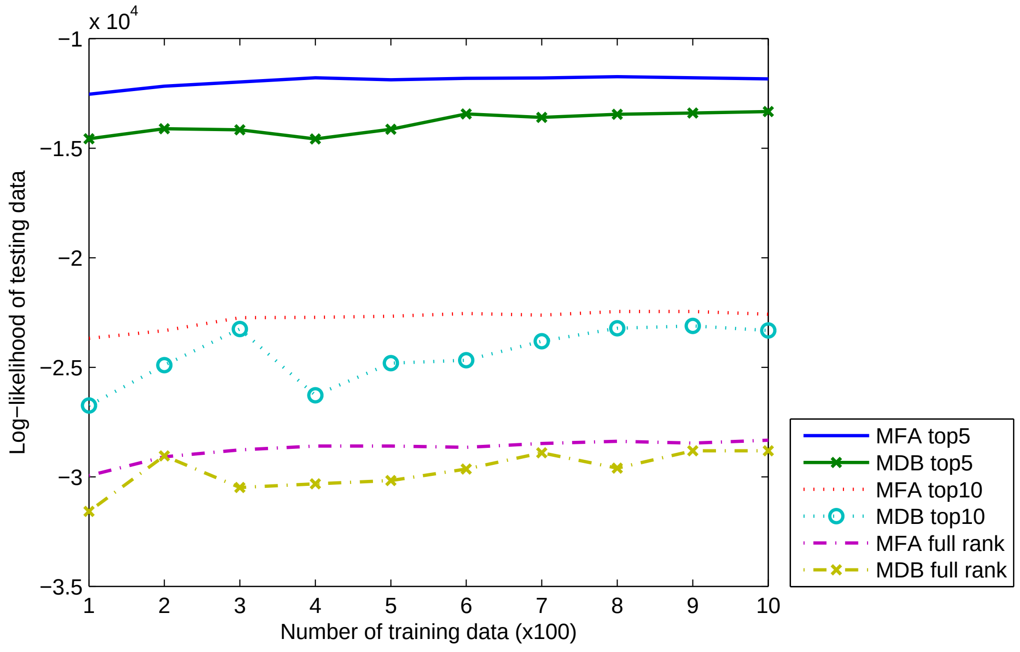

Figure 5.1: Testing log-likelihood for MFA and MDB models, for top5/top10/full rankings.

5.5.3 Synthetic ranking from real data

We will apply the MFA model for six synthetic datasets with 4-7 items to rank. These are the ranking datasets we used in Chapter 5.2-5.4, they are synthetic from UCI dataset.

For each ranking dataset, we compare the performance of MFA with the mixture of distance-based model(MDB) (Murphy and Martin, 2003). We split each dataset into two halves, fifty percents for training and fifty percents for testing. For training data, we compute the BIC for MDB and MFA for different number of components and different number of factors. For testing data, we use the model and parameters obtained from training data to compute the log-likelihood.

The results of BIC are given in Table 5.3, and results for log-likelihood of testing data are given in Table 5.4. The highest BIC and biggest Log-likelihood for each dataset are in bold type. From Table 5.3 we can see, by BIC criterion, MFA models outperform MDB models for all datasets. When the number of items increases, the differences of BIC between two models also increase. From Table 5.4 we can see, if we evaluate the performance of models by the log-likelihood of testing data, the MFA models also outperform MSB models (except for Dataset Housing). Therefore, the MFA can provide a better fit and better prediction than the MDB model.

Table 5.3: BIC for MDB and MFA models with different parameters.

Dataset	♯G	MDB	MFA factor=1	MFA factor=2	MFA factor=3
Vehicle	2	-2190.93	**-1818.70**	-1827.01	
	3	-2025.61	-1832.76	-1834.92	
	4	-1962.89	-1846.83	-1863.04	
	5	-2038.57	-1867.09	-1893.75	
Calhouse	2	-62173.67	**-53318.85**	-53332.11	
	3	-66015.90	-53357.59	-53362.47	
	4	-69817.55	-53376.53	-53405.04	
	5	-63388.39	-53411.67	-53439.20	
Authorship	2	-1089.92	-894.64	-906.93	
	3	-922.19	**-921.07**	-936.61	
	4	-1008.66	-947.31	-966.89	
	5	-1208.93	-971.34	-997.57	
Cpu-small	2	-41484.06	-34106.10	-33975.98	
	3	-40329.10	-34059.66	**-33934.38**	
	4	-44089.54	-33943.03	-33981.17	
	5	-40560.97	-33989.99	-34033.17	
Housing	2	-3106.22	-2425.29	-2486.52	-2499.94
	3	-3007.82	**-2372.34**	-2422.18	-2447.81
	4	-2950.63	-2397.99	-2468.24	-2466.48
	5	-2984.64	-2395.84	-2448.70	-2475.95
Segment	2	-18768.11	-10170.32	-9888.94	-9898.04
	3	-16778.90	-9909.70	-9523.44	-9613.59
	4	-16399.49	-9441.29	-9407.95	-9493.75
	5	-16358.48	**-9209.48**	-9276.25	-9401.10

Table 5.4: Log-likelihood for MDB and MFA models with different parameters.

Dataset	♯G	MDB	MFA factor=1	MFA factor=2	MFA factor=3
Vehicle	2	-1093.90	-857.80	-857.70	
	3	-1004.60	-855.00	-849.20	
	4	-934.00	-848.70	-843.60	
	5	-968.00	-844.00	**-843.50**	
Calhouse	2	-31023.80	-26854.40	-26924.00	
	3	-32866.30	**-26801.70**	-26877.30	
	4	-35040.90	-26898.50	-26952.90	
	5	-32195.90	-26960.80	-26903.80	
Authorship	2	-2934.40	-2709.60	-6137.10	
	3	-3240.30	**-2602.00**	-3208.10	
	4	-3265.40	-2960.30	-3831.50	
	5	-3241.50	-2972.10	-3851.90	
Cpu-small	2	-20842.30	-17211.90	-16988.30	
	3	-20214.50	-17055.70	-16963.70	
	4	-22105.60	-16950.80	-16952.60	
	5	-20328.70	-16975.90	**-16942.90**	
Housing	2	**-1906.10**	-3618.40	-2510.40	-2614.32
	3	-1948.60	-4830.30	-3326.60	-3774.64
	4	-1889.20	-3611.10	-3529.90	-3837.56
	5	-2017.60	-4154.20	-4510.70	-3837.61
Segment	2	-9347.50	-5154.90	-4950.50	-4952.58
	3	-8454.20	-4923.50	-4687.00	-4793.12
	4	-8292.60	-4688.80	-4639.60	-4926.66
	5	-8307.40	**-4564.30**	-4585.50	-4572.64

5.6 Applications to real data

In this section, we will apply MFA model to two real ranking datasets. The first dataset is a mixture of partial rankings with different length, and the second dataset only contains full rankings. We will show that MFA model can fit both partial and full ranking data sets well. Furthermore, we will select the best model by BIC criterion and list out the parameter estimate of MFA. An explanation of factors will be given for both datasets.

5.6.1 APA election

The APA dataset (1980 American Psychological Association presidential election) is first used by Diaconis (1989) and has since been studied by a number of ranking models. In this election, 15449 ballots were collected, each of them is a ranking of five candidates. However, only 5738 ballots gave the full ranking, and the rest of them gave only top-k ranking (k=1,2,3). Busse et al. (2007) analyzed this dataset by a mixture of distance-based model which can combine rankings of different lengths.

We use MFA model to fit this data, and we also modify the MFA model to deal with rankings with different length. The BIC for different number of components and factors are computed and plotted in Figure 5.2. From the heat map we can see that, MFA model with 2 components and 1 factor fit the APA

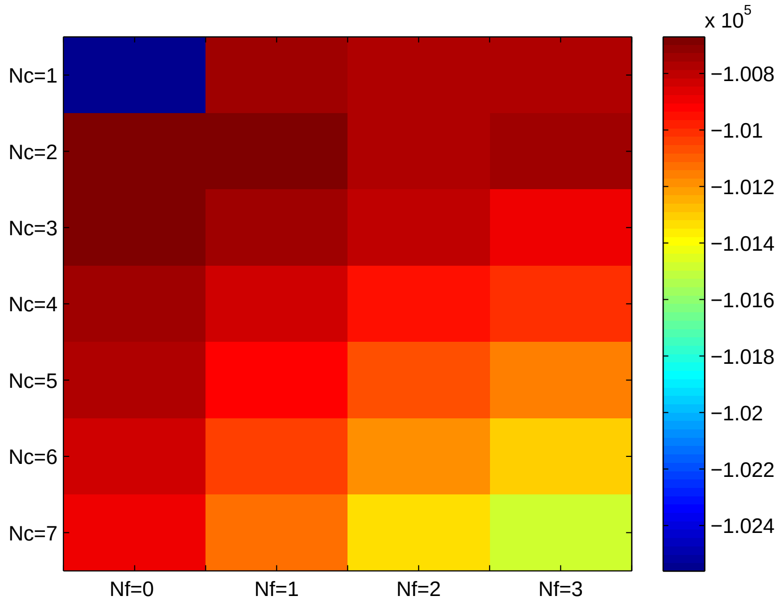

Figure 5.2: BIC for different number of components and factors, APA dataset

dataset best. We also compare the best BIC of MFA and MDB and list the results in Table 5.5. We can see that the BIC of MFA model is higher than the BIC of MDB model. Furthermore, we list the means and factor loadings of each components for MFA results in Table 5.6.

Best BIC (MFA)	Best BIC (MDB)
-100670.2 (2 components 1 factor)	-103224.5 (1 component)

Table 5.5: BIC for APA data

Among the five candidates, candidates A and C are research psychologists, candidates D and E are clinical psychologists, candidate B is a community psychologist. From the results in Table 5.6 we can see that, component 1 takes

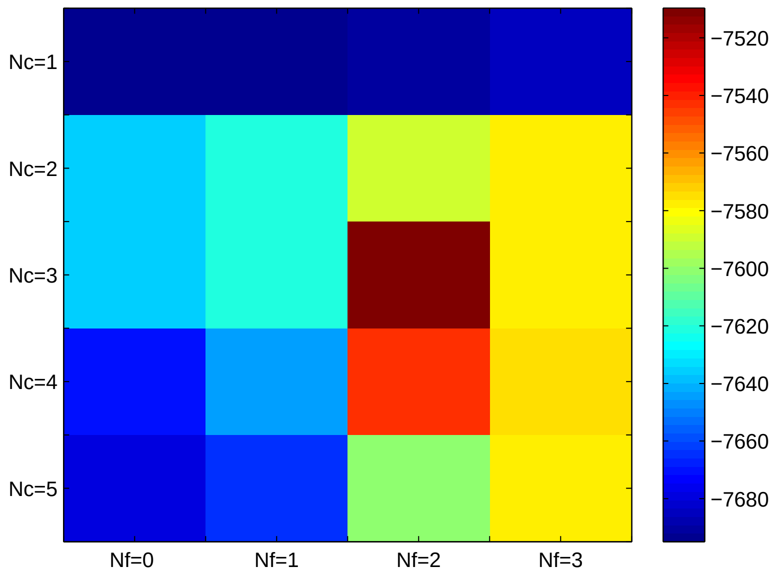

Figure 5.3: BIC for different number of components and factors, physician survey dataset

Component 1 (71.4%)	Factor 1	Mean
Candidate A	-0.755	2.225
Candidate B	-0.263	2.432
Candidate C	-0.582	2.595
Candidate D	0.453	3.698
Candidate E	0.815	4.234

Component 2 (28.6%)	Factor 1	Mean
Candidate A	0.287	2.399
Candidate B	-0.645	2.677
Candidate C	0.515	1.939
Candidate D	-0.812	2.811
Candidate E	0.781	4.393

Table 5.6: Factors and means for APA data

around 70% of the whole population. In component 1, there is a big contrast between clinical and research/community psychologists: people support clinical psychologists will less support research/community psychologist and vice versa. This contrast makes sense because in most of the academic areas, there exist some arguments between theory based and application based researches. And people in this group preferred candidates D and E most. In component 2, people still prefer candidate E but do not like candidate C very much, and there is a big contrast between two clinical psychologists: D and E.

5.6.2 A population-based physician survey

A group of 949 doctors in Hong Kong was asked by a postal survey to rank a list of seven incentives or catalysts that they think may influence the computerization process of hospitals in Hong Kong (Leung et al., 2003). Table 5.7 describes the seven incentives/catalysts in this survey.

566 responses with full ranking are collected. I apply the MFA model to this ranking data. We try different number of groups and different number of factors, for each combination we compute the BIC and plot the results in Figure 5.3. By the BIC criterion, the best model is composed with 3 groups and 2 factors in each group.

The estimated factor loadings were rotated by varimax rotation, and these values were expressed by the correlation between factors and utilities. The factors and mean value for each ranking item and each group are summarized in Table 5.8, relative important factor loading values (absolute value is grater than 0.7) are in bold type. We use the model to calculate the posterior probabilities, the group each doctor should belong to is determined by these probabilities. I summarize the demographic information for doctors in each group in Table 5.9.

First we analyze the results of MFA model. Doctors in Group 1 care more about increased saving and government financial incentives, the first factor can be regarded as a concern of working-improvement: improve efficiency and quality

Incentives/catalysts	
1. Improved efficiency	Improved efficiency with real-time electronic access to laboratory results and hospital inpatient data.
2. Improved quality care	Improved quality care through better access to medical evidence and information electronically.
3. Competitive pressures	Competitive peer pressure in terms of more practices becoming computerized.
4. Increased savings	Increased savings from efficiency gains in office operations (e.g., billing and inventory control).
5. Financial incentives	Government financial incentives to support the computerization of medical practices.
6. Patient demand	Increased public or patient demand for a portable electronic medical record and more generally for a computerized practice.
7. Government regulation	Government regulation requiring mandatory reporting of patient information.

Table 5.7: Descriptions of seven incentives/catalysts

Group 1	Factor 1	Factor 2	Mean
1. Improved efficiency	**-0.975**	-0.117	4.666
2. Improved quality care	**-0.870**	-0.122	2.430
3. Competitive pressures	-0.153	**-0.733**	3.138
4. Increased savings	0.465	0.220	5.113
5. Financial incentives	0.631	0.303	5.195
6. Patient demand	0.619	0.071	4.232
7. Government regulation	0.074	0.416	3.254
Group 2	**Factor 1**	**Factor 2**	**Mean**
1. Improved efficiency	**-0.810**	0.577	5.443
2. Improved quality care	-0.371	0.347	2.074
3. Competitive pressures	0.522	-0.273	3.853
4. Increased savings	0.020	-0.521	3.095
5. Financial incentives	0.010	**0.831**	3.212
6. Patient demand	0.669	0.165	3.940
7. Government regulation	**0.871**	0.209	4.543
Group 3	**Factor 1**	**Factor 2**	**Mean**
1. Improved efficiency	**-0.963**	0.067	6.806
2. Improved quality care	**-0.976**	-0.028	4.311
3. Competitive pressures	**0.861**	-0.335	4.101
4. Increased savings	**0.904**	-0.381	3.177
5. Financial incentives	**0.775**	-0.386	2.649
6. Patient demand	-0.113	**0.894**	1.021
7. Government regulation	0.621	-0.104	3.648

Table 5.8: Factors and means for three Groups

	Group 1	Group 2	Group 3
	42.84%	32.30%	24.86%
Gernder			
Male	83.3%	77.4%	79.6%
Femaile	16.7%	22.6%	20.4%
Years of work experience			
0-10	27.27%	29.61%	23.57%
11-20	37.60%	39.81%	40.00%
21-30	21.07%	21.36%	20.71%
>30	14.05%	9.22%	15.71%
Work setting			
Individual	40.60%	26.67%	39.42%
Corporate	59.40%	73.33%	60.58%
Monthly income (HKD)			
<100,000	57.27%	52.30%	48.91%
100,000-149,999	24.36%	25.13%	33.58%
150,000-199,999	8.12%	14.36%	10.22%
>200,000	10.26%	8.20%	7.30%
Qualified specialist			
Yes	55.98%	65.13%	61.31%
No	44.02%	34.87%	38.69%
Work place			
Public	56.00%	73.11%	59.12%
Private	44.00%	26.89%	40.88%
Work hours			
Mean	50.43	51.42	50.98

Table 5.9: Summary of the demographic information for doctors in three Groups

care. The second factor represents the competitive peer pressures.

Doctors in Group 2 care more about improved efficiency of the government regulation, the first factor shows a contract between improved efficiency and government regulation: doctors care more about working improvement will care less about regulation and vice versa. The second factor financial incentive is also a major concern in this group.

Doctors in Group 3 care more about working-improvement, but care less about financial incentive and patient demands. The first factor represents a contrast between working-related and non-working-related: doctors concern improving efficiency and quality care will care less about financial and regulation stuff. The second factor tell us the patient demands is a high loading factor.

Then we analyze the results from demographic information of doctors in three groups and compare them with the MFA results. Summary statisitcs are shown in Table 5.9. Group 1 is the largest group, which contains more male individual doctors, lower percentage of qualified specialist, and higher percentage doctors from private. This can explain why doctors in this group care more about financial related issues.

Group 2 is the second largest group. This group contains obviously more doctors working for corporate and public, so this explain why their concerns to government regulation are relative higher and why they care less about financial issues.

Group 3 is the smallest group, there is no obvious characteristic in this group through demographic analysis, so we think the factors and bias in the group might be explained by some unknown features.

5.7 Conclusion and Future Work

In this chapter we propose a mixture of factor analyzers (MFA) model for analyzing ranking data from heterogeneous population. A two-stage EM algorithm with high efficiency is introduced to fit the MFA model. We test the new model for both simulation and real ranking datasets. In our simulation studies, we demonstrate that MFA can deal with both full and partial rankings, BIC is a good criterion for model selection, and MFA model outperforms MDB model via BIC for testing data and log-likelihood for testing data.

Two real ranking datasets are tested and explained in detail. APA is a partial ranking dataset. The MFA model can deal with partial rankings with different lengths, it can also obtain a higher BIC then MDB model. The hospital survey is a full ranking dataset, after clustering the population into three groups, we can find some different factors in each group and give some explanations. Moreover, by comparing the MFA results with the real demographic information, we can see that some explanations of these factors do make sense.

In the future, we might consider MFA models for ranking data with different number of factors in each group, and MFA models with covariant for both judges and ranking items.

5.8 Appendix: The moments of rank truncated normal distribution

We first introduce some notations that are used by (Hajivassiliou et al., 1996). For a vector $\boldsymbol{\eta}$ with elements $(\eta_1, \cdots, \eta_n)$, we use the notation $\boldsymbol{\eta}_{<j}$ to denote the subvector $(\eta_1, \cdots, \eta_{j-1})$ that contains the first $j-1$ elements of $\boldsymbol{\eta}$ and use $\boldsymbol{\eta}_{-j}$ to denote a subvector that excludes component j of $\boldsymbol{\eta}$. Similarly, for a matrix Σ, if $\Sigma_{j.}$ is a vector that stores the j-th row of Σ, then $\Sigma_{j.<j}$ denotes a vector containing the first $j-1$ elements of the j-th row of Σ, and $\Sigma_{-j,-j}$ denotes the submatrix of Σ excluding row j and column j.

Let $\mathbf{y}$ be a $p \times 1$ random vector that is distributed as normal $\mathbf{N}_p(\widetilde{\boldsymbol{\mu}}, \widetilde{\Sigma})$. Let $\mathbf{r} = (r_1, \cdots, r_p)'$ be the rank pattern of $\mathbf{y}$, and $\mathbf{ro} = (ro(1), \cdots, ro(p))'$ be the order of the elements of $\mathbf{y}$ such that $y_{ro(1)} >, \cdots, > y_{ro(p)}$. If $\mathbf{y}$ is subjected to the rank constraint $\mathbf{r}$, we say that $\mathbf{y}$ has a rank truncated normal distribution, and the probability density function of $\mathbf{y}$ is then given by

$$f(\widetilde{\boldsymbol{\mu}}, \widetilde{\Sigma}) \mathbf{I}_{D^*}(\mathbf{y}) / Pr(\mathbf{y} \in D^*), \tag{5.8.0.1}$$

where $f(\widetilde{\boldsymbol{\mu}}, \widetilde{\Sigma})$ is the probability density of the normal distribution with mean

$\widetilde{\boldsymbol{\mu}}$ and variance $\widetilde{\Sigma}$, $D^* = \{\mathbf{y}\ |y_{ro(1)} > y_{ro(2)} > \cdots > y_{ro(p)}\}$, $\mathbf{I}_{D^*}(\mathbf{y})$ is an indicator function whose value is 1 when $\mathbf{y} \in D^*$ and 0 elsewhere, and $Pr(\mathbf{y} \in D^*)$ is the probability that $\mathbf{y}$ falls in D^*.

Now we will derive the first and second moment of $(\mathbf{y} - \boldsymbol{\mu})$ subject to the rank constraints. Without loss of generality, we consider the case $\mathbf{r} = (1, 2, \cdots, p)'$, that is, $D^* = \{y_1 > y_2 > \cdots > y_p\}$. Let $\mathbf{v} = \mathbf{A}\mathbf{y}$ where $\mathbf{A}$ is a $p \times p$ constant matrix whose elements a_{ij} are all equal to zero except that $a_{ii} = 1$ for $i = 1, 2, \cdots, p$, and $a_{i,i+1} = -1$ for $i = 1, 2, \cdots, p-1$. Then, $\mathbf{v} \sim \mathbf{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\mu = \mathbf{A}\widetilde{\mu}$ and $\Sigma = \mathbf{A}\widetilde{\boldsymbol{\Sigma}}\mathbf{A}'$. Let $v_i, i = 1, \cdots, p$ be the elements of $\mathbf{v}$, then the constraint that $\mathbf{y}$ falls in $D^* = \{\mathbf{y}|y_1 > y_2 > \cdots > y_p\}$ is equivalent to the constraint that $\mathbf{v}$ falls in D where $D = \{\mathbf{v}|v_1 > 0, v_2 > 0, \cdots, v_{p-1} > 0, -\infty < v_p < +\infty\}$. As a result, we have

$$E[(\mathbf{y} - \widetilde{\boldsymbol{\mu}})|\mathbf{y} \in \mathbf{D}^*] = \mathbf{A}^{-1}\mathbf{E}[(\mathbf{v} - \boldsymbol{\mu})|\mathbf{v} \in \mathbf{D}], \tag{5.8.0.2}$$

$$E[(\mathbf{y} - \widetilde{\boldsymbol{\mu}})(\mathbf{y} - \widetilde{\boldsymbol{\mu}})'|\mathbf{y} \in \mathbf{D}^*] = \mathbf{A}^{-1}\mathbf{E}[(\mathbf{v} - \boldsymbol{\mu})(\mathbf{v} - \boldsymbol{\mu})'|\mathbf{v} \in \mathbf{D}](\mathbf{A}^{-1})', \tag{5.8.0.3}$$

As $\mathbf{v_p}$ has no constraint, given $v_1, \cdots, v_{p-1}$, its first and the second moment can be obtained in closed form by the conditional distribution of v_p given $\mathbf{v_{-p}}$:

$$v_p|\mathbf{v_{-p}} \sim \mathbf{N}(\boldsymbol{\mu}_\mathbf{p} + \boldsymbol{\Sigma}_{\mathbf{p}.<\mathbf{p}}\boldsymbol{\Sigma}^{-1}_{\mathbf{-p},\mathbf{-p}}(\mathbf{v_{-p}} - \boldsymbol{\mu}_{\mathbf{-p}}), \boldsymbol{\Sigma}_{\mathbf{pp}} - \boldsymbol{\Sigma}_{\mathbf{p}.<\mathbf{p}}\boldsymbol{\Sigma}^{-1}_{\mathbf{-p},\mathbf{-p}}\boldsymbol{\Sigma}'_{\mathbf{p}.<\mathbf{p}}), \tag{5.8.0.4}$$

where $\boldsymbol{\mu}_p$ and Σ_{pp} are respectively the last elements of $\boldsymbol{\mu}$ and Σ . From (5.8.0.4), it can be shown that

$$E[(v_p - \mu_p)|\mathbf{v_{-p}}] = \mathbf{\Sigma_{p.<p}\Sigma^{-1}_{-p,-p}}(\mathbf{v_{-p}} - \mu_{\mathbf{-p}}), \tag{5.8.0.5}$$

$$E[(v_p-\mu_p)^2|\mathbf{v_{-p}}] = \mathbf{\Sigma_{pp}}-\mathbf{\Sigma_{p.<p}\Sigma^{-1}_{-p,-p}}[\mathbf{\Sigma_{-p,-p}}-(\mathbf{v_{-p}}-\mu_{\mathbf{-p}})(\mathbf{v_{-p}}-\mu_{\mathbf{-p}})']\mathbf{\Sigma^{-1}_{-p,-p}\Sigma'_{p.<p}} \tag{5.8.0.6}$$

$$E[(v_p - \mu_p)(\mathbf{v_{-p}} - \mu_{\mathbf{p}})|\mathbf{v_{-p}}] = (\mathbf{v_{-p}} - \mu_{\mathbf{-p}})(\mathbf{v_{-p}} - \mu_{\mathbf{-p}})^{\mathbf{T}}\mathbf{\Sigma^{-1}_{-p,-p}\Sigma^{T}_{p.<p}} \tag{5.8.0.7}$$

$$E(\mathbf{v} - \boldsymbol{\mu}) = E[E[(\mathbf{v} - \boldsymbol{\mu})|\mathbf{v}_{-p}]] = E((\mathbf{v}_{-p} - \mu_{-p})', E[(\mathbf{v}_p - \mu_p)|\mathbf{v}_{-p}])'; \tag{5.8.0.8}$$

$$E(\mathbf{v} - \boldsymbol{\mu})(\mathbf{v} - \boldsymbol{\mu})' = E[E[(\mathbf{v} - \boldsymbol{\mu})(\mathbf{v} - \boldsymbol{\mu})'|\mathbf{v_{-p}}]] \tag{5.8.0.9}$$

$$= \begin{bmatrix} E[(\mathbf{v_{-p}} - \mu_{\mathbf{-p}})(\mathbf{v_{-p}} - \mu_{\mathbf{-p}})'] & E\left[(\mathbf{v}_{-p} - \mu_{-p})E[(\mathbf{v}_p - \mu_p)|\mathbf{v_{-p}}]\right] \\ E\left[(\mathbf{v}_{-p} - \mu_{-p})'E[(\mathbf{v}_p - \mu_p)|\mathbf{v}_{-p}]\right] & E\left[E[(\mathbf{v}_p - \mu_p)^2|\mathbf{v_{-p}}]\right] \end{bmatrix}$$

It can be seen from the above derivation that to calculate $E[(\mathbf{v} - \boldsymbol{\mu})|\mathbf{v} \in \mathbf{D}]$ and $E[(\mathbf{v}-\boldsymbol{\mu})(\mathbf{v}-\boldsymbol{\mu})'|\mathbf{v} \in \mathbf{D}]$, we only need to calculate the $E[(\mathbf{v_{-p}}-\mu_{\mathbf{-p}})|\mathbf{v_{-p}} \in \mathbf{D_1}]$ and $E[(\mathbf{v_{-p}} - \mu_{\mathbf{-p}})(\mathbf{v_{-p}} - \mu_{\mathbf{-p}})'|\mathbf{v_{-p}} \in \mathbf{D_1}]$, where $D_1 = \{v_1 > 0, \cdots, v_{p-1} > 0\}$. It is easy to see that the distribution of $\mathbf{v_{-p}}$ subject to the constraint D_1 is $\mathbf{v_{-p}} \sim \mathbf{N_{p-1}}(\mu_{\mathbf{-p}}, \mathbf{\Sigma_{-p,-p}})\mathbf{I_{D_1}}(\mathbf{v_{-p}})/\mathbf{Pr}(\mathbf{v_{-p}} \in \mathbf{D_1})$. Hence, the moments of rank truncated multinormal distribution with p dimensions can be obtained by the moments of rectangle truncated multinormal distribution with $p-1$ dimensions.

Let $\boldsymbol{\eta} = (\eta_1, \cdots, \eta_{p-1})' \sim \mathbf{N}(0, \mathbf{I}_{p-1})$, where $\mathbf{I}_{p-1}$ is an identity matrix. Let $\phi(x)$ and $G(x)$ be the pdf and cdf of standard normal random variate, respectively. Let Γ be the Cholesky factor of $\Sigma_{-p,-p}$, such that $\Sigma_{-p,-p} = \Gamma\Gamma'$. Then according to (Hajivassiliou et al., 1996), we have

$$E[(\mathbf{v}_{-p} - \boldsymbol{\mu}_{-p})|\mathbf{v}_{-p} \in D_1] = \Gamma \int c_0^{-1} \boldsymbol{\eta} \omega(\boldsymbol{\eta}) \prod_{j=1}^{p-1} \phi(\eta_j|\mathbf{C}_j(\eta_{<j}))d\boldsymbol{\eta}, \qquad (5.8.0.10)$$

$$E[(\mathbf{v}_{-p} - \boldsymbol{\mu}_{-p})(\mathbf{v}_{-p} - \boldsymbol{\mu}_{-p})'|\mathbf{v}_{-p} \in D_1] = \Gamma \left(\int c_0^{-1} \boldsymbol{\eta}\boldsymbol{\eta}' \omega(\boldsymbol{\eta}) \prod_{j=1}^{p-1} \phi(\eta_j|\mathbf{C}_j(\eta_{<j}))d\boldsymbol{\eta} \right) \Gamma', \qquad (5.8.0.11)$$

where

$$\mathbf{C}_j(\eta_{<j}) = \{\eta_j|(-\mu_j - \Gamma_{j.<j}\eta_{<j})/\Gamma_{jj} < \eta_j < +\infty\}, \qquad (5.8.0.12)$$

$$\phi(\eta_j|\mathbf{C}_j(\eta_{<j})) = \phi(\eta_j)\mathbf{I}_{\mathbf{C}_j(\eta_{<j})}(\eta_j)/G(\mathbf{C}_j(\eta_{<j})), \qquad (5.8.0.13)$$

$$G(\mathbf{C}_j(\eta_{<j})) = 1 - G((-\mu_j - \Gamma_{j.<j}\eta_{<j})/\Gamma_{jj}), \qquad (5.8.0.14)$$

$$\omega(\boldsymbol{\eta}) = \prod_{j=1}^{p-1} G(\mathbf{C}_j(\eta_{<j})), \qquad (5.8.0.15)$$

$$c_0 = \int \omega(\boldsymbol{\eta}) \prod_{j=1}^{p-1} \phi(\eta_j|\mathbf{C}_j(\eta_{<j}))d\boldsymbol{\eta}, \qquad (5.8.0.16)$$

and $\phi(\eta_j|\mathbf{C}_j(\eta_{<j}))$ is the conditional density of η_j given the event $\mathbf{C}_j(\eta_{<j})$. An observation $\boldsymbol{\eta}$ with its elements η_j can therefore be drawn recursively from the

one-dimensional conditional distribution $\phi(\eta_j|\mathbf{C}_j(\eta_{<j}))$ by taking

$$\eta_j = G^{-1}(u_j G(-\mu_j - \Gamma_{j.<j}\eta_{<j})/\Gamma_{jj} + 1 - u_j),$$

where u_j are draws from the uniform $[0,1]$ density. The sampling method that draws a sample $\eta^1, \cdots, \eta^m$ with each of these observations drawn using the recursive method is called the Geweke-Hajivassiliou-Keane simulator. Hajivassiliou et al. (1996) shown that

$$\Gamma\frac{\sum_{i=1}^m \eta^i\omega(\eta^i)}{\sum_{i=1}^m \omega(\eta^i)} \text{ and } \Gamma\frac{\sum_{i=1}^m \eta^i\eta^{i'}\omega(\eta^i)}{\sum_{i=1}^m \omega(\eta^i)}\Gamma' \tag{5.8.0.17}$$

are good approximations of the integrals in (5.8.0.10) and (5.8.0.11). As a result, the conditional expectations in the left-hand side of (5.8.0.2) and (5.8.0.3) can be obtained by GHK simulator.

Bibliography

Allison, P. D. and Christakis, N. A. (1994). Logit models for sets of ranked items. *Sociological Methodology*, 24:199–228.

Allwein, E., Schapire, R., and Singer, Y. (2002). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141.

Alvo, M. and Cabilio, P. (1991). On the balanced incomplete block design for rankings. *Annals of Statistics*, 19:1597–1613.

Bagheri, M. A., Gao, Q., and Escalera, S. (2013). A genetic-based subspace analysis method for improving error-correcting output coding. *Pattern Recognition*, 46:2830–2839.

Bailey, T. and Jain, A. K. (1978). A note on distance-weighted k-nearest neighbor rules. *IEEE Trans Syst Man Cybern*, 8:311–313.

Bautista, M. A., Escalera, S., Baro, X., Radeva, P., and Vitria, J. (2012). Minimal design of error-correcting output codes. *Pattern Recognition Letters*, 33:693–702.

Beggs, S., Cardell, S., and Hausman, J. (1981). Assessing the potential demand for electric cars. *Journal of Econometrics*, 16:1–19.

Bockenholt, U. (2001). Mixed-effects analysis of rank-ordered data. *Psychometrika*, 66(1):45–62.

Breiman, L., Friedman, J., Olsen, R., and Stone, C. (1984). Classification and regression trees. Belmont, California: Wadsworth.

Busse, L. M., Orbanz, P., and Buhmann, J. M. (2007). Cluster analysis of heterogeneous rank data. *Proceedings of the 24th International Conferencece on Machine Learning*, pages 113–120.

Chapman, R. G. and Staelin, R. (1982). Exploiting rank ordered choice set data within the stochastic utility model. *Journal of Market Research*, 19:288–301.

Cheng, W. W., Hühn, J., and Hüllermeier, E. (2009). Decision tree and instance-based learning for label ranking. In Bottou, L. and Littman, M., editors, *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, pages 161–168, Montreal, Canada. Omnipress.

Craig, B. M., Busschbach, J. J. V., and Salomon, J. A. (2009). Modeling ranking, time trade-off, and visual analog scale values for eq-5d health states: a review and comparison of methods. *Medical Care*, 47(6):634–641.

Critchlow, D. E. (1985). Matric methods for analyzing partially ranked data. *Lecture Notes in Statistics*, 34.

Critchlow, D. E., Fligner, M. A., and Verducci, J. S. (1991). Probability models on rankings. *Journal of Mathematical Psychology*, 35:294–318.

Critchlow, D. E. and Verducci, J. S. (1992). Detecting a trend in paired rankings. *Applied Statistics*, 41:17–29.

Croon, M. A. (1989). Latent class models for the analysis of rankings. In G. De Soete, H. F. and Klauer, K. C., editors, *New developments in psychological choice modeling*, pages 99–121, North-Holland. Elsevier Science.

David, H. A. (1988). The method of paired comparisons. Oxford University Press.

Decarlo, L. T. and Luthar, S. S. (2000). Analysis and class validation of a measure of parental values perceived by early adolescents: an application of a latent class models for rankings. *Educational and Psychological Measurement*, 60(4):578–591.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data using the em algorithm. *Journal of Royal Statistic Society. B*, 39:1–38.

Diaconis, P. (1988). Group representations in probability and statistics. *Institute of Mathematical Statistics, Hayward.*

Diaconis, P. (1989). A generalization of spectral analysis with applications to ranked data. *Annals of Statistics*, 17:949–979.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Dittrich, R., Katzenbeisser, W., and Reisinger, H. (2000). The analysis of rank ordered preference data based on bradley-terry type models. *OR Spektrum*, 22:117–134.

Dudani, S. (1976). Distance weighted k-nearest-neighbor rule. *In IEEE Trans. on Systems, Man and Cybernetics.*

Escalera, S., Pujol, O., and Radeva, P. (2010). Error-correcting ouput codes library. *Journal of Machine Learning Research*, 11:661–664.

Feigin, P. D. (1993). Modelling and analysing paired ranking data. In Fligner, M. A. and Verducci, J. S., editors, *Probability Models and Statistical Analyses for Ranking Data*, pages 75–91. Springer-Verlag.

Fligner, M. A. and Verducci, J. S. (1986). Distance based ranking models. *Journal of the Royal Statistical Society: Series B*, 48:359–369.

Fligner, M. A. and Verducci, J. S. (1988). Multi-stage ranking models. *Journal of the American Statistical Association*, 83:892–901.

Freund, Y. and Schapire, R. (1997). A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–138.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–407.

Ghahramani, Z. and Hinton, G. E. (1996). The em algorithm for mixtures of factor analyzers. *Department of Computer Science, University of Toronto, Technical Report.*

Gormley, I. C. and Murphy, T. B. (2008). Exploring voting blocs within the irish electorate: A mixture modeling approach. *of American Statistical Association*, 103:1014–1027.

Hajivassiliou, V. and McFadden, D. (1998). The method of simulated scores for the estimation of ldv models. *Econometrica*, 66:863–896.

Hajivassiliou, V., McFadden, D., and Ruud, P. (1996). Simulation of multivariate normal rectangle probabilities and their derivatives theoretical and computational results. *Journal of Econometrics*, 72:85–134.

Hausman, J. and Ruud, P. A. (1987). Specifying and testing econometric models for rank-ordered data. *Journal of Econometrics*, 34:83–104.

Hodges, F. (1951). Discriminatory analysis, nonparametric discrimination: Consistency properties. *Technical Report 4, US Air Force, School of Aviation Medicine, Randolph Field, TX.*

Hofmann, T., Scholkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *The Annals of Statistics*, 36:1171–1220.

Hüllermeier, E., Fürnkranz, J., Cheng, W. W., and Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172:1897–1916.

Kamishima, T. and Akaho, S. (2006). Effecient clustering for orders. *Proceedings of the 2nd International Workshop on Mining Complex Data,*, pages 274–278.

Kira, K. and Rendell, L. A. (1997). A practical approach to feature selection. *Artificial Intelligence*, 97:273–324.

Kong, E. B. and Dietterich, T. G. (1995). Error-correcting output coding corrects bias and variance. *In Proceedings of the Twelfth International Conference on Machine Learning.*

Kononenko, I. (1994). Estimating attributes: analysis and extension of relief. *Machine Learning: ECML-94*, pages 171–182.

Koop, G. and Poirier, D. J. (1994). Rank-ordered logit models: an empirical analysis of ontario voter preferences. *Journal of Applied Econometrics*, 9(4):369–388.

Krabbe, P. F. M., Salomon, J. A., and Murray., C. J. L. (2007). Quantificaition of health states with rank-based nonmetric multidimensional scaling. *Medical Decision Making*, 27:395–405.

Lee, H. P. and Yu, P. L. H. (2012). Mixtures of weighted distance-based models for ranking data with applications in political studies. *Computational Statistics and Data Analysis*, 56:2486–2500.

Leung, G. M., Yu, P. L. H., Wong, I. O. L., Johnston, J. M., and Tin, K. Y. K. (2003). Incentives and barriers that influence clinical computerization in hong kong: A population-based physician survey. *Journal of the American Medical Informatics Association.*

Loh, W. Y. and Shih, Y. S. (1997). Split selection methods for classification trees. *Statistica Sinica*, 7:815–840.

Loh, W. Y. and Vanichsetakul, N. (1988). Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 83:715–728.

Luce, R. D. (1959). Individual choice behavior. *New York: Wiley.*

Mallows, C. L. (1957). Non-null ranking models. *Biometrika*, 30:81–93.

Marden, J. I. (1995). Analyzing and modeling rank data. Chapman and Hall.

Maydeu-Olivares, A. and Bockenholt, U. (2005). Structural equation modeling of paired-comparison and ranking data. *Psychological Methods*, 10(3):285–304.

McCabe, C., Brazier, J., Gilks, P., Tsuchiya, A., Roberts, J., O'Hagan, A., and Stevens, K. (2006). Use rank data to estimate health state utility models. *Journal of Health Economics,*, 25:418–431.

McLachlan, G., Bean, R., and Jones, L. B. T. (2007). Extension of the mixture of factor analyzers model to incorporate the multivariate t-distribution. *Comput. Statist. Data Anal.*, 51:5327–5338.

McLachlan, G. J., Peel, D., and Bean, R. W. (2003). Modelling high-dimensional data by mixtures of factor analyzers. *Comput. Statist. Data Anal.*, 41:379–388.

Mingers, J. (1987). Expert systems - rule induction with statistical data. *Journal of Operational Research Society*, 38:39–47.

Moors, G. and Vermunt, J. (2007). Heterogeneity in post-materialists value priorities, evidence from a latent class discrete choice approach. *European Sociological Review,*, 23:631–648.

Murphy, T. B. and Martin, D. (2003). Mixtures of distance-based models for ranking data. *Computational Statistics and Data Analysis*, 41:645–655.

Niblett, T. and Bratko, I. (1986). Learning decision rules in noisy domains. *Proceedings of Expert Systems*, 86.

Nombekela, S. W., Murphy, M. R., Gonyou, H. W., and Marden, J. I. (1993). Dietary preferences in early lactation cows as affected by primary tastes and some common feed avors. *Journal of Diary Science,*, 77:2393–2399.

Quinlan, J. R. (1986). Introduction of decision tree. *Machine Learning*, 1:81–106.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Machine Studies,*, 27:221–234.

Quinlan, J. R. (1992). C4.5 programs for machine learning. *San Mateo, CA: Morgan Kaufmann.*

Ratcliffe, J., Brazaier, J., Tsuchiya, A., Symonds, T., and Brown, M. (2006). Estimation of a preference based single index from the sexual quality of life questionnaire (sqol) using ordinal data. *Discussion Paper Series, Health Economics and Decision Science, The University of Sheffield*, 6:6.

Regenwetter, M., Ho, M. H. R., and Tsetlin, I. (2007). Sophisticated approval voting, ignorance priors, and plurality heuristics: a behavioral social choice analysis in a thurstonian framework. *Psychological Review*, 114(4):994–1014.

Riketta, M. and Vonjahr, D. (1999). Multidimensional scaling of ranking data for different age groups. *Experimental Psychology*, 46(4):305–311.

Salomon, J. A. (2003). Reconsidering the use of rankings in the valuation of health states: a model for estimating cardinal values from ordinal data. *Population Health Metrics*, 1:1–12.

Schapire, R. and Singer, Y. (1999). Improved boosting algorithms using condifence-rated prediction. *Machine Learning*, 37:297–336.

Skrondal, A. and Rabe-Hesketh, S. (2003). Multilevel logistic regressioin for polytomous data and rankings. *Psychometrika*, 68(2):267–287.

Stern, H. (1993). Probability models on rankings and the electoral process. In Fligner and Verducci, J. S., editors, *Probability Models and Statistical Analyses for Ranking Data,*, pages 173–195. Springer-Verlag.

Thurstone, L. L. (1927). A law of comparative judgement. *Psychological Reviews*, 34:273–286.

Thurstone, L. L. (1947). Multiple factor analysis. Chicago: University of Chicago Press.

van Dijk, B., Fok, D., and Paap, R. (2007). A rank-ordered logit model with unobserved heterogeneity in ranking capabilities. *Econometric Institute Report, 2007-07, Erasmus University Rotterdam.*

Vembu, S. and Gärtner, T. (2010). Label ranking: a survey. in fürnkranz, j. and hüllermeier, e.(eds.), preference learning. Springer-Verlag.

Vermunt, J. K. (2004). Multilevel latent class models. *Sociological Methodology*, 33:213–239.

Yao, G. and Bockenholt, U. (1999). Bayesian estimation of thurstonian ranking models based on the gibbs sampler. *British Journal of Mathematical and Statistical Psychology,*, 52:79–92.

Yu, P. L. H. (2000). Bayesian analysis of order-statistics models for ranking data. *Psychometrika*, 65(3):281–299.

Yu, P. L. H. (2003). Statistical modeling of ranking data. *Computational Mathematics and Modeling*, pages 319–326.

Yu, P. L. H., Lam, K. F., and Lo, S. M. (2005). Factor analysis for ranked data with application to a job selection attitude survey. *Journal of Royal Statistics, Soc. A*, pages 583–597.

Yu, P. L. H. and Lee, P. H. (2010). Distance-based tree models for ranking data. *Computational Statistics and Data analysis*, 54:1672–1682.

Yu, P. L. H., Wan, W. M., and Lee, P. H. (2009). Decision tree modeling for ranking data. In Furnkranz, J. and Hullermeier, E., editors, *Preference Learning*, pages 139–165. Springer.

Zhao, J. H. and Yu, P. L. H. (2008). Fast ml estimation for the mixture of factor analyzers via a ecm algorithm. *IEEE Transactions on neural network*.

Zhu, J., Zou, H., Rosset, S., and Hastie, T. (2009). Multi-class adaboost. *Statistics and its Interface*, 2:349–360.