A Survey of Ranking Algorithms

Kishor Jothimurugan
(PennKey: `kishor`; Email: `kishor@seas.upenn.edu`)
Nicolas Koh
(PennKey: `pennkey2`; Email: `email2@xxx.upenn.edu`)
Mingyang Liu
(PennKey: `mingyal`; Email: `mingyal@sas.upenn.edu`)

Abstract

Abstract text goes here.

1 Introduction to Ranking

In this project, we study the general problem of supervised ranking and survey work in the area, paying attention to consistency results and algorithms that perform empirically well but lack theoretical backing.

Roughly, the goal of supervised ranking is to learn from appropriate samples in some way so as to learn how to rank, i.e., given a new set of n items, determine how they can be ordered so as to do well on some performance measure.

The canonical application is web search. Here, the search engine is given a query q and may find n documents $v_1, \ldots, v_n$, but has to determine an order in which they should be presented to a user. Early on, this ordering was determined by hand-tuned models [**?**], but there is good reason to believe that models adapting to collected data (samples) would perform better.

A sample is a (multi)set of instance-label pairs. Some appropriate types of samples are:

- (Binary relevance) $S = \{((q_i, (v_{i,j})_{j=1}^n), \mathbf{y}_i)\}_{i=1}^m$, where $\mathbf{y}_i \in \{0,1\}^n$. That is, an instance is a query q_i with a list of n documents; the label for the instance is a vector $\mathbf{y}_i = (y_{i,j})_{j=1}^n$ indicating, for each document, whether it is relevant or not. This may be determined, e.g., by whether there is a user who clicked on the link or not.

- (Relevance scores) $S = \{((q_i, (v_{i,j})_{j=1}^n), \mathbf{y}_i)\}_{i=1}^m$, where $\mathbf{y}_i \in R^n \subseteq \mathbb{R}_+^n$. Same as the previous case, except that each document has a relevance score. This may be determined, e.g., by the number of users who clicked on the link.

- (Preference graphs) $S = \{((q_i, (v_{i,j})_{j=1}^{n}), G_i)\})_{i=1}^{m}$, where G_i is a weighted directed graph on the n_i documents, all weights being non-negative. We can interpret $v_j \overset{w_{j,k}}{\to} v_k$ as saying that v_j is preferred to v_k with an importance weight of $w_{j,k}$. Note that this case subsumes the first two: a label for binary relevance can be coded as a directed bipartite graph in the obvious way with all weights 1, and a label for relevance scores may be coded as a DAG with weights being the difference in scores. An important special case is when the graph consists of just a single edge, expressing a single pairwise preference. Preference graphs allow for cases where complete information on relevance is hard to obtain; for instance, an e-commerce ranking system may obtain S from a consumer survey, and surveyees should be allowed to specify preferences for only a small set of items. Ranking from preference graphs also have applications beyond just information retrieval, e.g., in recommendation systems and social choice theory, where we may want to aggregate preferences from different agents into a global ranking [].

One may view relevance scores as a natural multiclass generalization of binary relevance labels, and preference graphs as the structured-prediction-generalization of relevance scores.

In each case, the goal is for the learning algorithm to learn from these samples and output a ranking function $f : Q \times D^n \to S_n$, where Q is the set of queries, D is the set of documents, and S_n is the set of permutations on $[n]$. The formalism can be simplified by thinking of query-document pairs as feature vectors in some space $\mathcal{X}$ (which is also done in practice), so that dependence on queries can be dropped.

Formally, the supervised ranking problem can be framed as follows. There is an instance space $\mathcal{X}$, a label space $\mathcal{Y}$ (which varies as above), and the prediction space is $\widehat{\mathcal{Y}} = S_n$. A permutation $\sigma \in \widehat{\mathcal{Y}}$ is interpreted such that $\sigma(i)$ is the position of the ith item. A learning algorithm takes $S = (\mathbf{x}_i, y_i)_{i=1}^{m}$ as input, where $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ is a feature vector and $y_i \in \mathcal{Y}$ is a label following the cases above, and outputs a function $f_S : \mathcal{X} \to S_n$. There is no loss of generality, since the feature space can in principle include Q itself.

Performance of the learning algorithm is measured by some performance measure $M : \widehat{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R}_+$ or loss function $\ell : \widehat{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R}_+$. The goal is to minimize the ℓ-risk

$$\operatorname{err}_D^{\ell}[f_S] = \mathbf{E}_{(X,Y)\sim D}[\ell(f_S(X), Y)],$$

for some unknown distribution D on $\mathcal{X} \times \mathcal{Y}$ from which the samples are drawn i.i.d. The Bayes ℓ-risk for a given distribution D is the minimum possible ℓ-risk, denoted $\operatorname{err}_D^{l,*} = \inf_{f:\mathcal{X}\to\widehat{\mathcal{Y}}} \operatorname{err}_D^{l}[f]$. A learning algorithm is said to be ℓ-consistent if for any distribution D,

$$\operatorname{err}_D^{l}[f_S] \xrightarrow{\Pr} \operatorname{err}_D^{l,*} \quad \text{as} \quad m \to \infty,$$

where $m = |S|$ is the sample size and the probability is over samples S of size m which is drawn i.i.d. from D. Typically, minimizing the empirical loss $\sum_{i=i}^{m} \ell(f(\mathbf{x}_i), y_i)$ over some suitable function class $\mathcal{F}_m$ gives a consistent algorithm but is computationally hard. Therefore surrogate losses are used. Instead of learning a function from $\mathcal{X} \to \widehat{\mathcal{Y}}$, one typically learns a function $h_S : \mathcal{X} \to \mathcal{C}$ where the surrogate space $\mathcal{C}$ is a convex set in $\mathbb{R}^d$ for some d. In the case of ranking, $\mathcal{C}$ is usually $\mathbb{R}^n$ or $\mathbb{R}^{(n)(n-1)/2}$ which correspond to learning pointwise and pairwise scores respectively. The learned

function is composed with a prediction map $\mathtt{pred} : \mathcal{C} \to \widehat{\mathcal{Y}}$ to map instances to permutations, $f_S = \mathtt{pred} \circ h_S$.

The function h_S is obtained by minimizing a surrogate loss $\psi : \mathcal{C} \times \mathcal{Y} \to \mathbb{R}_+$ on the samples in S, $h_S \in \operatorname{argmin}_{h \in \mathcal{H}_m} \sum_{i=1}^{m} \psi(h(\mathbf{x}_i), y_i)$. This gives consistency w.r.t. ψ. Given a distribution D over $\mathcal{X} \times \mathcal{Y}$, define ψ-risk as follows:

$$\mathrm{err}_D^{\psi}[h_S] = \mathbf{E}_{(X,Y)\sim D}[\psi(h_S(X), Y)]$$

The learning algorithm is ψ-consistent if $\mathrm{err}_D^{\psi}[h_S]$ converges in probablity to $\mathrm{err}_D^{\psi,*}$ as $m = |S|$ goes to infinity, where $\mathrm{err}_D^{\psi,*} = \inf_{h:\mathcal{X}\to\mathcal{C}} \mathrm{err}_D^{\psi}[h]$. A surrogate ψ along with a prediction mapping $\mathtt{pred}$ is said to be ℓ-consistent if ψ-consistency of a learning algorithm that outputs h_S on input S implies ℓ-consistency of the algorithm that ouputs $f_S = \mathtt{pred} \circ h_S$ on input S.

Ramaswamy and Agarwal [2] showed that a property of surrogates called ℓ-calibration is sufficient for consistency of general multiclass losses by extending the result for 0-1 loss in [4]. They also define the notion of Convex Calibration dimension which is the smallest d for which there is a calibrated convex surrogate with a surrogate space $\mathcal{C} \subseteq \mathbb{R}^d$. For most ranking problems, this is bounded above by n or n^2.

There are many surrogates that are used in practice that are non-convex. In such cases, one has to analyze the consistency and also the computational complexity of emprical loss minimization. Some of the surrogates used in practice are proven to be consistent whereas some are proven to be inconsistent. There are a few practical approaches for which there are no consistency results giving oppurtunity for further work in this area. The surrogates for some the loss functions studied can be classified as pointwise, pairwise and listwise. We do not categorize the surrogates presented in this article. More information can be found in [1].

The rest of the report is organized as follows. In Section 2 we look at the standard 0-1 loss on permutations. In Section 3, we consider surrogates for losses defined for binary relevance labels. In Section 4, we consider surrogates for NDCG. In Section **??**, we consider surrogates for PD loss. In Section 6, we look at extensions to ranking and other problems considered in this project.

2 0-1 Loss

3 Subset Ranking

Binary relevance, also known as subset ranking, is the case where the true labels are subsets of $[n]$, i.e., $\mathcal{Y} = \{0,1\}^n$. There are various loss functions in this setting most of which are low rank meaning that the loss function $\ell : \widehat{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R}_+$ when viewed as a $|\widehat{\mathcal{Y}}| \times |\mathcal{Y}|$ matrix has small rank. Hence the quadratic surrogate loss for low rank losses defined in [3] can be applied directly to get consistent algorithms for these problems. Here, we look at a few important loss functions.

3.1 Precision@q

The precision@q loss is a measure of how (im)precise the prediction is, with respect to the true label when we look at the prediction σ as a map from $[n] \to \{0,1\}$ which sends the top q positions to 1 and the rest to 0. More precisely, the loss $\ell_{P@q} : \{0,1\}^n \times S_n \to \mathbb{R}_+$ is defined by,

$$\ell_{P@q}(\sigma, \mathbf{y}) = 1 - \frac{1}{q} \sum_{i:\sigma(i)\leq q} y_i$$

A consistent surrogate loss $\psi_{P@q} : \mathbb{R}^n \times \mathcal{Y} \to \mathbb{R}_+$ and its corresponding prediction mapping $\mathbf{pred}_{P@q} : \mathbb{R}^n \to S_n$ are given by:

$$\psi_{P@q}(\mathbf{u}, \mathbf{y}) = \|\mathbf{u} - \mathbf{y}\|_2^2$$

$$\mathbf{pred}_{P@q}(\mathbf{u}) \in \text{argmax}_{\sigma \in S_n} \sum_{i:\sigma(i)\leq q} u_i$$

In this case, the prediction mapping just amounts to sorting the indices in decreasing order of the $\mathbf{u}$ values.

3.2 MAP

The mean average precision, as the name suggests, is a loss based on the avergae of precision at positions where the permutation maps the documents labelled 1 in $\mathbf{y}$. The loss $\ell_{\text{MAP}} : \{0,1\}^n \times S_n \to \mathbb{R}_+$ is given by,

$$\ell_{\text{MAP}} = 1 - \frac{1}{\|\mathbf{y}\|_1} \sum_{i:y_i=1} \frac{1}{\sigma(i)} \sum_{j:\sigma(j)\leq\sigma(i)} y_j$$

where $\|\mathbf{y}\|_1 = \sum_{i=1}^n y_i$ is the number relevant documents according to $\mathbf{y}$. It is known that there cannot be an n-dimensional convex calibrated surrogate for this loss. [3] gives an $O(n^2)$ dimensional surrogate for this loss. The consistent (convex) surrogate $\psi_{\text{MAP}} : \mathbb{R}^{n(n+1)/2} \times \{0,1\}^n \to \mathbb{R}_+$ and the corresponding $\mathbf{pred}$ mapping is as follows:

$$\psi_{\text{MAP}}(\mathbf{u}, \mathbf{y}) = \sum_{i=1}^n \sum_{j=1}^i \left(u_{ij} - \frac{y_i y_j}{\|\mathbf{y}\|_1} \right)^2$$

$$\mathbf{pred}_{\text{MAP}(\mathbf{u})} \in \text{argmax}_{\sigma \in S_n} \sum_{i=1}^n \sum_{j=1}^i \frac{u_{ij}}{\max(\sigma(i), \sigma(j))}$$

This method involves learning a score for every unordered pair of documents/indices where a higher score indicates that both elements of the pair should be higher in rank. Unfortunately, there is no known polynomial time algorithm to compute the $\mathbf{pred}$ function. Therefore, one has to make lwo noise assumptions on the distribution D to get fast algorithms. Under one such condition which assumes that the expected values of $y_{ii}/\|\mathbf{y}\|_1$ given any $x \in \mathcal{X}$ are more relevant than the cross terms $y_{ij}/\|\mathbf{y}\|_1$ where $i \neq j$, one can simply sort the indices in decreasing order of the diagonal terms u_{ii} to compute the permutation from scores. In that scenario, it is also enough to just learn the diagonal terms giving an n-dimensional surrogate.

4 NDCG

If the algorithm is to output a scoring function and has access to relevance scores as labels, perhaps the simplest idea is to minimize similarity between the output and scores. In [**?**], it was established that minimizing the pointwise least-squares surrogate

$$\phi(\vec{s}, y) = \sum_{j=1}^{n} (s_j - y_j)^2$$

actually minimizes $1 - \text{NDCG}(\vec{s}, y)$ because we can upper bound the latter by some multiple of the former. But this upper bound is coarse, and the result also does not tell us if the surrogate is consistent. Similar

4.1 LambdaRank

5 Pairwise Disagreement Loss

[**?**] studies consistency of pdloss. They show two widely used surrogates are not generally consistent. Then they gives low.noise condition on the probability distribution under which we can have calibrated surrogate loss. [3] generalizes low.noise condition and provides another more general condition and surrogate loss.

Unfinished

6 Variations of Ranking and Other Problems

In this section, we give an overview of some problems we considered in the course of the project, motivated by papers we were reading [].

One direction was along ranking. In the ranking problem, the prediction space is the set of permutations, which one can think of as the set of total orders. We considered generalizing this to partial orders with possibly some restriction, and identified task scheduling as an application domain. We outline how ranking algorithms may be applied to solve some problems in these domains.

Another direction was along online expert advice. TODO.

These are discussed in the sections below.

6.1 Task scheduling

Consider a cloud service that offers distributed infrastructure to process jobs, or a modern data processing system that processes queries on large databases. In such systems, queries may be passed to an execution planner which uses statistics, current workloads, and cost models to choose

an optimal plan among a set of candidate execution plans. These plans are in the form of directed acyclic graphs with edges indicating task dependencies. Some metrics for choosing plans include makespan (time to finish executing the DAG) and resource consumption (e.g., to obtain fairness between queries). One challenge, however, is that available resources change with time, so an execution plan deemed optimal before execution can become sub-optimal during its lifetime [**?**].

With this as setting, one may consider learning a decision rule for scheduling tasks. We consider $\mathcal{X}$ to be some set of feature vectors (extracted from the query, tasks, resources at time of execution, time-of-day, and so on), and $\mathcal{Y}$ to be the set of preference graphs with appropriate augmentation depending on the metric of interest. Samples $S = \{((x_{i,j})_{j=1}^{n_i}, y_i)_{i=1}^{m}\}$ may come from computing the best DAG in hindsight and collecting statistics. We want the learning algorithm to output a DAG that minimizes ℓ-risk for some suitable loss ℓ.

If we make the simplification that execution of plans proceed in rounds (e.g. MapReduce), ranking techniques can be brought to bear. Specifically, let the prediction space be $\widehat{\mathcal{Y}} = \{f : \bigcup_{i=1}^{n} \mathcal{X}^n \to \mathbb{Z}_+\}$, i.e. functions which stratifies jobs into rounds. This is no different from functions that output relevance scores.

6.1.1 Minimizing Job Completion Time and Parallel Processing Cost?

Here, we want the scoring function to output as few rounds as possible. A way to do this is as follows. For r items, let

$$\mathcal{Y}_r = \{(G, \vec{c}) \mid G \text{ is a weighted DAG on } [r] \text{ and } \vec{c} = (c_1, \ldots, c_r), c_1 < \ldots < c_r\}.$$

The label space is $\mathcal{Y} = \mathcal{Y}_r$ is a set of finite number of weighted DAGs over r items. We interpret each c_i to be the unit cost incurred for scheduling a task past round i. The prediction space $\mathcal{T}$ has size r^r

The loss ℓ can then be defined as:

$$\ell(f, (G, \vec{c})) = \sum_{i,j:i \neq j} w_{i,j}^{G} \mathbf{1}_{f(i) \geq f(j)} + \sum_{i=1}^{n} c_i \sum_{j=1}^{n} \mathbf{1}_{f(j) \geq i}$$

Mingyang: The loss defined below eq(1) is easier to analyze and have the same property to penalize depth.

6.1.2 Mingyang's draft for Minimizing Job Completion Time

$$\text{Target Loss: } l : \mathcal{Y} \times \mathcal{T}, \text{ with } l((\mathbf{y}, \mathbf{c}), m) = \sum_{i \neq j} y_{ij} 1_{\{m(i) \geq m(j)\}} + \sum_{i=1}^{r} c_i 1_{\{(\max_j m(j)) \geq i\}} = (\mathbf{y}', \mathbf{c}') \begin{pmatrix} \mathbf{\Phi}_1(m) \\ \mathbf{\Phi}_2(m) \end{pmatrix} \tag{1}$$

, where $\mathbf{\Phi}_1(m)$ is $r(r-1)$ length vector with entries $1_{\{m(i) \geq m(j)\}}$, and $\mathbf{\Phi}_2(m)$ is r length vector with

entries $1_{\{(\max_j m(j))\geq i\}}$. By Theorem 3 [**?**], we have l-calibrated surrogate $(\psi_l^*, \text{pred}_l^*)$, where

$$\psi_l^*((\mathbf{y},\mathbf{c}),(\mathbf{u},\mathbf{v})) = \sum_{i\neq j}^{r}(u_{ij}-y_{ij})^2 + \sum_{k=1}^{r}(v_k - c_i)^2, \quad \text{pred}_l^*(\mathbf{u},\mathbf{v}) \in \text{argmin}_{m\in\mathcal{T}}\left(\mathbf{u}'\mathbf{\Phi}_1(m) + \mathbf{v}'\mathbf{\Phi}_2(m)\right)$$

It is easy to see that solving pred_l^* is same as solving original PD-loss problem.

Let $m^* : [r] \to [r]$ be the optimal solution of eq(1), and let $|m| = \max_j m(j)$. We notice that m must satisfies the below two rules.

Lemma 1. *Fact1(no gap): For any $k \in [|m|]$, $m^{-1}(k) \neq \emptyset$.*

Proof. If there exists such a k, then we can move all the items ranked lower than k w.r.t m by one level, i.e, set $m'(l) = m(l) - 1, \forall l, m(l) > k$. We can reduce the target loss by $c_{|m|} > 0$, which contradicts that m is optimal. □

Before deriving the second fact, we first re-examine the target loss. The target loss can be written as three parts: pdloss, clusterloss, depthloss

$$\sum_{i\neq j} y_{ij}1_{\{m(i)>m(j)\}} + \sum_{i\neq j} y_{ij}1_{\{m(i)=m(j)\}} + \sum_{i=1}^{r} c_i 1_{\{(\max_j m(j))\geq i\}}$$

Assumption 1. *There is a unique total order σ^*. For any $i \neq j$, wlog $m(i) < m(j)$*

$$y_{ij} + y_{ji} \geq \sum_{k\in\{l:\sigma^*(l)\in[m(i),m(j)]\},l=(\sigma^*)^{-1}(\sigma^*(k)+1)\}} y_{kl} + y_{lk}$$

It means under total ranking, clustering two nodes will incur strict bigger loss than sum of clustering loss of any two items between them.

Lemma 2. *Fact2(respect DAG): If $\sigma^*(k) < \sigma^*(l)$, then $m(k) \leq m(l)$.*

Proof. If not, then for m, there must exist two clusters which do not respect DAG. That is to say, for some $k < l$, and let the first cluster $C_1 = \{i : m(i) = k\}$, the second $C_2 = \{j : m(j) = l\}$, $\exists$ pair $(i,j) \in (C_1, C_2)$, $\sigma^*(i) > \sigma^*(j)$. Now consider m' which flip all these pairs of m until we can not find more such pairs. and keep the rest of clusters unchanged. When we cannot find more such pairs, let $C_{1,u}$ be items previously in C_1 and now still in C_1; $C_{1,d}$ be items previously in C_1 but now in C_2; et $C_{2,u}$ be items previously in C_2 but now in C_1; $C_{2,d}$ be items previously in C_2 and now still in C_2. Since we only do flip pairs, $|C_{1,d}| = |C_{2,u}|$.

$$\begin{aligned} &C_1 = C_{1,u} \cup C_{1,d} \\ &C_2 = C_{2,u} \cup C_{2,d} \\ &m : m(C_1) = k; m(C_2) = l \\ &m' : m'(C_{1,u} \cup C_{2,u}) = k; m'(C_{1,d} \cup C_{2,d}) = l \end{aligned} \tag{2}$$

- pdloss: m' will have no greater pdloss than m, because m' respect more ranking than m.

- depthloss: m and m' have same depthloss.
- clusteringloss: let $C_{cl}(m)$, $C_{cl}(m')$ be clusteringloss for m, m'
  - $C_{cl}(m) = \Big(\sum_{i\in C_{1,u}, j\in C_{1,u}} y_{ij} + \sum_{i\in C_{2,d}, j\in C_{2,d}} y_{ij}\Big) + \Big(\sum_{i\in C_{1,u}, j\in C_{1,d}} (y_{ij}+y_{ji})\Big) + \Big(\sum_{i\in C_{2,u}, j\in C_{2,d}} (y_{ij}+ y_{ij})\Big)$
  - $C_{cl}(m') = \Big(\sum_{i\in C_{1,u}, j\in C_{1,u}} y_{ij} + \sum_{i\in C_{2,d}, j\in C_{2,d}} y_{ij}\Big) + \Big(\sum_{i\in C_{1,u}, j\in C_{2,u}} (y_{ij}+y_{ij})\Big) + \Big(\sum_{i\in C_{1,d}, j\in C_{2,d}} (y_{ij}+ y_{ij})\Big)$

  By our flip rule, $\sigma^*(i) > \sigma^*(j)$ for any $i \in C_{1,d} \cup C_{2,d}, j \in C_{2,u} \cup C_{1,u}$. If not, the fliping process will not stop. Since the first big bracket is same for m and m', we only work on the rest two. For any $i \in C_{1,u}$, $j \in C_{2,d}, k \in C_{2,u}, l \in C_{1,d}$, There are four possibilities:
  - $\sigma^*(k) < \sigma^*(i) < \sigma^*(j) < \sigma^*(l)$
  - $\sigma^*(k) < \sigma^*(i) < \sigma^*(l) < \sigma^*(j)$
  - $\sigma^*(i) < \sigma^*(k) < \sigma^*(j) < \sigma^*(l)$
  - $\sigma^*(i) < \sigma^*(k) < \sigma^*(l) < \sigma^*(j)$

  For the first case
  $$\begin{aligned}(y_{il} + y_{li}) + (y_{kj} + y_{jk}) &\geq (y_{ij} + y_{ji} + y_{jl} + y_{li}) + (y_{ki} + y_{ik} + y_{ij} + y_{ji}) \\ &> y_{ik} + y_{ki} + y_{jl} + y_{lj}\end{aligned}$$
  For the second case
  $$\begin{aligned}(y_{il} + y_{li}) + (y_{kj} + y_{jk}) &\geq (y_{il} + y_{li}) + (y_{ki} + y_{ik} + y_{ij} + y_{ji} + y_{lj} + y_{jl}) \\ &> y_{ik} + y_{ki} + y_{jl} + y_{lj}\end{aligned}$$
  For the third case
  $$\begin{aligned}(y_{il} + y_{li}) + (y_{kj} + y_{jk}) &\geq (y_{kj} + y_{jk}) + (y_{ki} + y_{ik} + y_{kj} + y_{jk} + y_{lj} + y_{jl}) \\ &> y_{ik} + y_{ki} + y_{jl} + y_{lj}\end{aligned}$$
  For the fourth case
  $$\begin{aligned}(y_{il} + y_{li}) + (y_{kj} + y_{jk}) &\geq (y_{kl} + y_{lk} + y_{jl} + y_{li}) + (y_{ki} + y_{ik} + y_{kl} + y_{lk}) \\ &> y_{ik} + y_{ki} + y_{jl} + y_{lj}\end{aligned}$$

Note that the above inequality is from unique ordering and low.noise condition, because for any (i, j), wlog $m(i) < m(j)$, then $y_{ij} > y_{ji} \geq 0$, which implies $y_{ij} + y_{ji} > 0$. Above calculation shows $C_{cl}(m) > C_{cl}(m')$ which contradicts optimality of m.

□

Lemma 3. *The optimal $m^* \in \mathcal{M}$, where $\mathcal{M}$ is a set containing all elements satisfying Fact2.*

Note that if there is a unique total order σ^*, then $\mathcal{M} = \{\sigma^*\}$, and m can be derived by clustering items in m^*_{pd}. This motivates us a naive algorithm for deriving m^*. Basically, we start from σ^*, and find the best way to cluster items, that is, reduce the target loss by

Increase SMALL loss (from y)by clustering, but decrease BIG loss (from c) by smaller depth

We also notice that minimizing eq(1) is equivalent to minimize the following quantity:

$$\text{argmin}_m l((\mathbf{y}, \mathbf{c}), m) = \text{argmin}_m \sum_{i=1}^{r}\sum_{j=1}^{i-1}(y_{ij} - y_{ji})1_{\{m(i)>m(j)\}} + \sum_{i=1}^{r}\sum_{j=1}^{i-1} y_{ij}1_{\{m(i)=m(j)\}} + \sum_{i=1}^{r} c_i 1_{\{(\max_j m(j))\geq i\}}$$

Before we present first algorithm, we start from some notations. For any $m \in \mathcal{M}$, by fact1, we know $m^{-1}(i) \neq \emptyset$ for $i \leq |m|$.

- Let $\mathcal{C}^m = \{\{m^{-1}(i)\}_{i=1}^{|m|}\}$ be set of clusters of m
  - $\mathcal{C}^m[i] = \{m^{-1}(i)\}$ is set of items in cluster i under m
  - Note that σ^* has r clusters, and each has one and only one item.
- Let cost of merging cluster i and $i+1$ under m be

$$cost^m_{i,i+1} = \sum_{n_1 \in \mathcal{C}^m[i]} \sum_{n_2 \in \mathcal{C}^m[i+1]} \left(-(y^{\mathbf{p}}_{n_1 n_2} - y^{\mathbf{p}}_{n_2 n_1})1_{n_1>n_2} 1_{\{m(n_1)>m(n_2)\}} + y^{\mathbf{p}}_{n_1 n_2} 1_{n_1>n_2} + y^{\mathbf{p}}_{n_2 n_1} 1_{n_2>n_1} \right) \tag{3}$$

  , where each summand is non-negative.

We first derive a consistent permutation σ^* under low.noise condition(or related conditions) [3], [**?**], and then plug σ^* into below algorithm. If there is a unique total ranking and it satisfies low.noise assumptions, we will find a m achieving lowest target loss.

Lemma 4. *Given m, define cost $= min_{i\in[|m|-1]} cost^m_{i,i+1}$. cost is non-decreasing after each clustering.*

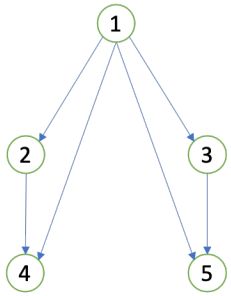
Proof. Consider we have m, and then we cluster $i, i+1$ to have m'. We need to show $\min_{l\in[|m|-1]} cost^m_{l,l+1} \leq \min_{l'\in[|m'|-1]} cost^{m'}_{l',l'+1}$. Notice that $cost^m_{l,l+1} = cost^{m'}_{l',l'+1}$ for $l = l' + 1$ and $l \neq i, i+1$. Also $cost^m_{i-1,i} \leq cost^{m'}_{i-1,i}$ because now there are more items in i-th cluster under m' and each summand of eq(3) is non-negative. Same reason for $cost^m_{i,i+1} \leq cost^{m'}_{i,i+1}$. □

Thanks to the fact that c_i is strictly increasing, this lemma implies once $c_{cur} - cost$ is negative, any kind of clustering in the future will increase target loss. This justifies the stopping condition in the algorithm.

Algorithm 1 Naive algorithm

procedure CLUSTERING RANKING $(\mathcal{G}, \mathbf{p}, \mathbf{c}, \sigma)$
  Initialize $cur = r$
  for $i \in [r-1]$ **do**
    Let $cost = \min_{i \in [|m|-1]} cost^m_{i,i+1}$
    Let $cl = \text{argmin}_{i \in [|m|-1]} cost^m_{i,i+1}$
    if $cost \leq c_{cur}$ **then**
      Merge clusters cl and $cl+1$. That is, set $m^{-1}(cl) \leftarrow m^{-1}(cl) \cup m^{-1}(cl+1)$; set $m^{-1}(cl+i) \leftarrow m^{-1}(cl+i+1)$ for $i \geq 1$; and set $|m| \leftarrow |m| - 1$.
      $cur \leftarrow cur - 1$
    else break;
  return m

The last thing we should worry is that when there are more than 1 permutation achieve lowest pd loss. We should note that we may only derive one permutation from algorithm in [3], so whether other unoutputed permutation will achieve lower target loss? The answer is YES. Let's look at an example. The total ranking is not unique. $\sigma_1 = (1,2,3,4,5)$ or $\sigma_2 = (1,2,4,3,5)$ both minimize pd



loss, but will have different solution in our case. Whether σ_1 or σ_2 got outputed depends on $(\mathcal{G}, \mathbf{p})$. If we utilize surrogates from Theorem 6 from [3],

$$
\begin{aligned}
u_2 &= -y^{\mathbf{p}}_{12} + y^{\mathbf{p}}_{24} \\
u_3 &= -y^{\mathbf{p}}_{13} + y^{\mathbf{p}}_{35} \\
u_4 &= -y^{\mathbf{p}}_{24} - y^{\mathbf{p}}_{14} \\
u_5 &= -y^{\mathbf{p}}_{35} - y^{\mathbf{p}}_{15}
\end{aligned}
$$

where according to the graph, $y^{\mathbf{p}}_{21}, y^{\mathbf{p}}_{42}, y^{\mathbf{p}}_{41}, y^{\mathbf{p}}_{31}, y^{\mathbf{p}}_{51}, y^{\mathbf{p}}_{53} = 0$. It is easily to see that

$$
\begin{aligned}
\{y_{12} = 100, y^{\mathbf{p}}_{24} = 100, y^{\mathbf{p}}_{13} = 100, y^{\mathbf{p}}_{35} = 99, y^{\mathbf{p}}_{14} = 200, y^{\mathbf{p}}_{15} = 300\} &\Rightarrow \sigma_1 \\
\{y_{12} = 100, y^{\mathbf{p}}_{24} = 100, y^{\mathbf{p}}_{13} = 400, y^{\mathbf{p}}_{35} = 99, y^{\mathbf{p}}_{14} = 200, y^{\mathbf{p}}_{15} = 499\} &\Rightarrow \sigma_2
\end{aligned}
$$

Although σ_1, σ_2 are equivalent w.r.t pdloss, it is not for our case. Let $c_1 = 0, c_2 = c_3 = c_4 = c_5 = 1$. Then if we start from σ_1, the best $m_1 = (1, \{2,3\}, \{4,5\})$ and from σ_2, the best $m_2 = (1, 2, \{3,4\}, 5)$. However m_1 have smaller target loss.

To solve this issue, we must go through all possible total ordering. That is, if we have all topological sorting of difference graph, we can run Algorithm 1 on each of them to find out the lowest target loss. Thanks to Theorem 7 of [3], we will have a calibrated surrogates which will also give us all possible total ordering.

Algorithm 2 Improved algorithm

procedure XXX $(\mathcal{G}, \mathbf{p}, \mathbf{c})$
    Get $\mathcal{M}$ from Algorithm 1 from [3]
    Let $m^* = NULL$, $loss = \infty$
    for $\sigma \in \mathcal{M}$ **do**
        Let m =Clustering ranking $(\mathcal{G}, \mathbf{p}, \mathbf{c}, \sigma)$, with l be the corresponding loss.
        if $l \leq loss$ **then** Set $m^* = m$
    return m^*

Warning: this algorithm only works under the assumption that $\mathcal{M}$ is small. If $\mathcal{M}$ is huge, then it would be hard to solve the problem quickly. Consider a DAG over r items(r is even), with edges $\{(i, i+1)\}_{i=1}^{r/2} \cup \{(i, i+1)\}_{r/2+1}^{r}$. Then $|\mathcal{M}| = (\frac{r}{2} + 1)!$.

References

[1] Hang Li. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

[2] Harish G Ramaswamy and Shivani Agarwal. Convex calibration dimension for multiclass loss matrices. *The Journal of Machine Learning Research*, 17(1):397–441, 2016.

[3] Harish G Ramaswamy, Shivani Agarwal, and Ambuj Tewari. Convex calibrated surrogates for low-rank loss matrices with applications to subset ranking losses. In *Advances in Neural Information Processing Systems*, pages 1475–1483, 2013.

[4] Ambuj Tewari and Peter L Bartlett. On the consistency of multiclass classification methods. *Journal of Machine Learning Research*, 8(May):1007–1025, 2007.