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: `pennkey3`; Email: `email3@xxx.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 $\mathcal{X}^n \to \widehat{\mathcal{Y}}$, one typically learns a function $h_S : \mathcal{X}^n \to \mathcal{C}$ 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 $\texttt{pred} : \mathcal{C} \to \widehat{\mathcal{Y}}$ to map instances to permutations, $f_S = \texttt{pred} \circ h_S$.

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, $h_S \in \text{argmin}_{h \in \mathcal{H}_m} \sum_{i=1}^{m} \psi(h(\mathbf{x}_i), y_i)$. This gives consistency w.r.t. ψ. Given a distribution D over $\mathcal{X} \times \mathcal{Y}$, define ψ-risk as follows:

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

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

Ramaswamy and Agarwal [**?**] 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 [**?**]. 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. The work [**?**] on constructing convex calibrated surrogates for multiclass losses that 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, gives many applications to subset ranking.

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 empirical 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 opportunity for further work in this area. The surrogates for some of 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 [**?**].

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

In this section we consider permutations as true labels, i.e., $\mathcal{Y} = S_n$. A natural loss in this setting is the 0-1 loss $\ell_{0\text{-}1} : S_n \times S_n \to \mathbb{R}_+$ given by,

$$\ell_{0\text{-}1}(\sigma, y) = \mathbf{1}(\sigma \neq y)$$

The results from [**?**] and [**?**] imply that any convex surrogate that is $\ell_{0\text{-}1}$-consistent must have surrogate space of dimension at least $n!$. Xia et. al. 2008 [**?**] showed that we can get convex, consistent, score based surrogates (surrogate space is $\mathbb{R}^n$) for the 0-1 loss if we make some reasonable assumptions on the underlying probability distribution D. For every $\mathbf{x} \in \mathcal{X}$, let $\mathbf{p}(\mathbf{x})$ denote the

probability distribution over $\mathcal{Y}$ given by $p_y(\mathbf{x}) = \Pr(Y = y \mid X = \mathbf{x})$. As before, for any $y \in \mathcal{Y}$, $y(i)$ denotes the position of i in the ranking.

A distribution D over $\mathcal{X} \times \mathcal{Y}$ is said to be order preserving with respect to a pair of positions (i, j) at $\mathbf{x} \in \mathcal{X}$ if for all $y \in \mathcal{Y}$ such that $y(i) < y(j)$, we have $p_y(\mathbf{x}) > p_{\text{flip}(i,j,y)}(\mathbf{x})$ where $\text{flip}(i, j, y)$ denotes the permutation derived from y by flipping positions of i and j. The technical condition assumed to prove consistency in [**?**] is that for every $\mathbf{x} \in \mathcal{X}$, there is an ordering of indices $j_1, ..., j_n$ such that D is order preserving with respect to $(j_1, j_2), (j_2, j_3)$,..., and (j_{n-1}, j_n) at $\mathbf{x}$. With this restriction we have many consistent surrogates:

- The likelihood loss is defined in terms of a probabilistic model with Markov-like assumptions. $\psi_{\text{like}} : \mathbb{R}^n \times S_n \to \mathbb{R}_+$ is given by,

$$\psi_{\text{like}}(\mathbf{u}, y) = -\log P(y \mid \mathbf{u})$$

$$P(y \mid \mathbf{u}) = \prod_{i=1}^{n} \frac{\exp(u_{y(i)})}{\sum_{j=i}^{n} \exp(u_{y(j)})}$$

- The cosine loss is parameterized by a strictly decreasing score function $g : [n] \to \mathbb{R}_+$. Given a permutation $y \in \mathcal{Y}$, let $\mathbf{g}(y)$ denote the vector $(g(y(1)), ..., g(y(n)))^T$. The loss $\psi_{\cos} : \mathbb{R}^n \times S_n \to \mathbb{R}_+$ is given by,

$$\psi_{\cos}(\mathbf{u}, y) = 1 - \frac{\mathbf{g}(y)^T \mathbf{u}}{\|\mathbf{g}(y)\|_2 \|\mathbf{u}\|_2}$$

- The cross entropy loss is simply the KL divergence between the probablity distributions over $\mathcal{Y}$: $P(\cdot \mid \mathbf{g}(y))$ and $P(\cdot \mid \mathbf{u})$ where P is the same as in the likelihood loss and $\mathbf{g}$ is the same as in the cosine loss. More precisely $\psi_{\text{KL}} : \mathbb{R}^n \times S_n \to \mathbb{R}_+$ is defined by,

$$\psi_{\text{KL}}(\mathbf{u}, y) = -\sum_{y' \in \mathcal{Y}} P(y' \mid \mathbf{g}(y)) \log\Big(\frac{P(y' \mid \mathbf{u})}{P(y' \mid \mathbf{g}(y))}\Big)$$

All three losses share a common prediction map $\text{pred}_{\text{0-1}} : \mathbb{R}^n \to S_n$ which on input a score vector $\mathbf{u}$ outputs a ranking according to decreasing order of the scores, breaking ties arbitrarily. These surrogate losses had been used in practical applications even before any consistency results were known for them. The cosine loss is the loss used in RankCosine whereas the cross entropy loss is used in ListNet. The cosine loss is non-convex while the other two surrogates are convex.

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. Hence the quadratic surrogate loss for low rank losses defined in [**?**] 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 $\ell_{P@q} : S_n \times \{0,1\}^n \to \mathbb{R}_+$ 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 $\psi_{P@q} : \mathbb{R}^n \times \{0,1\}^n \to \mathbb{R}_+$ and its corresponding prediction mapping $\mathsf{pred}_{P@q} : \mathbb{R}^n \to S_n$ are given by:

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

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

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 $\ell_{\mathrm{MAP}} : S_n \times \{0,1\}^n \to \mathbb{R}_+$ is given by,

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

where $\|\mathbf{y}\|_1 = \sum_{i=1}^n y_i$ is the number relevant documents according to $\mathbf{y}$. It is known that there is no n-dimensional convex calibrated surrogate for this loss. [**?**] gives an $O(n^2)$ dimensional surrogate for this loss. The consistent (convex) surrogate $\psi_{\mathrm{MAP}} : \mathbb{R}^{n(n+1)/2} \times \{0,1\}^n \to \mathbb{R}_+$ and the corresponding pred mapping are as follows:

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

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

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 pred function. Therefore, one has to make low 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

For relevance scores, the discounted cumulative gain at K is

$$\text{DCG@}K(\sigma, (y_j)_{j=1}^n) = \sum_{r=1}^{K} \frac{G(y_{\sigma^{-1}(r)})}{D(r)} = \sum_{i:\sigma(i)\leq K} \frac{G(y_i)}{D(\sigma(i))}.$$

(The first sum runs over positions and the second sum runs over object indices.) The gain function G boosts relevance scores, and is normally taken to be $G(y) = 2^y - 1$. The discount function D discounts the importance of the term based on the position of the item, and is normally taken to be $D(\sigma(i)) = \log(1 + \sigma(i))$. As before, DCG is the non-truncated version of this, and NDCG normalizes DCG to obtain a number in $[0, 1]$, defined as

$$\text{NDCG}(\sigma, (y_j)_{j=1}^n) = \frac{1}{N((y_j)_{j=1}^n)} DCG(\sigma, (y_j)_{j=1}^n),$$

where

$$N(y) = N((y_j)_{j=1}^n) = \frac{1}{\max_{\sigma'} \text{DCG}(\sigma', (y_j)_{j=1}^n)} \text{DCG}(\sigma, (y_j)_{j=1}^n)$$

is the normalization factor. In many applications, it is more important for the output to rank well for items that are near the top compared to ones below, and to rank well for highly relevant items compared to those with low scores; NDCG factors these in using the discount and gain functions respectively.

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 function output and labels; when they are equal, the NCDG is also one. In [**?**], it was established that regression works, in the sense that minimizing the 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. Moreover, such a surrogate is pointwise, in the sense that training is done on individual objects (x_j, y_j), and does not exploit potential structure in the problem.

In what is termed the pairwise approach, the surrogate loss $\phi(\vec{s}, y)$ decomposes over pairs of objects rather than single objects as above:

$$\phi(\vec{s}, y) \stackrel{\Delta}{=} \sum_{j,k:i\neq j} \phi'(s_j - s_k, y_j - y_k))$$

for some ϕ'. (This can work for preference graphs by replacing $y_j - y_k$ with the weight $w_{j,k}$.) The pairwise approach attempts to reduce the problem of ranking to the problem of classifying a pair as ± 1 (i.e., when an object should be preferred over another). Some choices of ϕ' are the hinge loss

of Ranking SVM [**?**] ($\phi'(s,y) = \mathbf{1}_{y<0}(1-s)_+$), exponential loss of RankBoost [**?**], and the logistic loss of RankNet [**?**]. It is instructive to review how the logistic loss of RankNet is derived. First, formulate a probability model for the probability of a pair (x_j, x_k) receiving the label +1, based on the function's output. Then think of $\text{sign}(y_j - y_k)$ for this pair as coming from the underlying distribution that we are learning. We want the distribution induced by the function's output to be similar to the underlying distribution, so it makes sense to take ϕ' as the KL-divergence

$$\phi'(s_j - s_k, y_j - y_k) = \text{KL}(\Pr[(x_j, x_k) \text{ has label } 1; s_j - s_k], \Pr[(x_j, x_k) \text{ has label } 1; \text{sign}(y_j - y_k)]).$$

RankNet uses the probability models

$$\Pr[(x_j, x_k) \text{ has label } +1; s_j - s_k] = \frac{1}{1 + e^{-(s_j - s_k)}}$$

$$\Pr[(x_j, x_k) \text{ has label } +1; \text{sign}(y_j - y_k)] = \begin{cases} 1 & \text{sign}(y_j - y_k) = 1 \\ \frac{1}{2} & \text{sign}(y_j - y_k) = 0 \\ 0 & \text{sign}(y_j - y_k) = -1 \end{cases}$$

(The latter can be changed to a proportion over the training set.) Computing the KL divergence gives $\phi'(s,y) = -\overline{p}s + \log(1 + e^s)$, where $\overline{p} = \Pr[(x_j, x_k) \text{ has label } +1; \text{sign}(y_j - y_k)]$.

An important observation of the pairwise approach is that these surrogates penalize misordered pairs x_j, x_k the same regardless of where they reside in the true sorted list according to y. Considering that NDCG places more weight on doing well near the top of the list compared to below, Burges et. al. adapted the gradient in the gradient descent used by RankNet to place more weight on improving performance for items near the top. This method is called LambdaRank, and has inspired a number of ranking algorithms that are now state-of-the-art [**?**].

How these pairwise approaches relate to a ranking measure like NDCG in general is less clear than the pointwise case. In [**?**], Chen et. al. resolved this by defining a notion of an essential loss $\ell(f; \vec{x}, y)$, which is computed as a weighted sum of n 0-1 misclassification errors. Each of these errors arise from the classification task of having f output the right object at position j according to y, after the objects at positions $1, \ldots, j-1$ have been removed. They showed that this essential loss is a proxy bounded by the surrogate loss and the ranking loss $1 - \text{NDCG}(\vec{s}, \vec{y})$. Using this idea, they showed that Ranking SVM, RankBoost, and RankNet all minimize $1 - \text{NDCG}(\vec{s}, y)$.

MOVE TO PD SECTION: In terms of consistency, Duchi et. al. [**?**] studied the consistency of pairwise surrogates, and proved that a surrogate is consistent w.r.t. a loss $\ell(\sigma, G)$ that is sensitive only to changes in rank (e.g. PD loss and NDCG) iff it is ℓ-calibrated (called edge-consistent in that paper). Using calibration, they showed that Ranking SVM and RankBoost are inconsistent w.r.t. PD loss, even under a low-noise assumption. However, empirical success of these pairwise surrogates led to another study that generalized edge-consistency/calibration to a condition they called rank-consistency. Roughly, a surrogate is rank-consistent if all its minimizers disagree with every non-minimizer of ℓ on some pair. Under a condition on distributions that they call rank-differentiability, similar to the low-noise condition, weighted pairwise surrogates are consistent w.r.t. PD loss. The surrogates of Ranking SVM, RankBoost, and RankNet, are all instances, and are hence consistent w.r.t. PD loss. END MOVE.

Without heuristics like the virtual gradient of RankNet, the pairwise approach does not incorporate bias towards the top. Hence in [**?**], Cao et. al. proposed the use of surrogates that is defined over

the whole list of items and hence takes the whole structure into account – this is termed the listwise approach. In that work, like in RankNet, we use probability models, but this time of

$$\Pr[(x_1, \ldots, x_n) \text{ is ordered according to } \sigma; \vec{s}]$$
$$\Pr[(x_1, \ldots, x_n) \text{ is ordered according to } \sigma; y].$$

This is thus a generalization of RankNet from pairwise to listwise. To make computation tractable, the two models are replaced with just a model of x_j being ranked first, but the loss is still the KL divergence of these two. This method is called ListNet, and also ranks among the state-of-the-art in the cross-benchmark study. Similar ideas come up in other work, e.g. ListMLE uses just a model for

In [**?**], it was established that the normalization factor in NDCG is crucial for consistency; the main result is that a surrogate is consistent iff for all $\mathcal{X}$-conditional distributions on relevance scores, the minimizer achieving the conditional surrogate risk respects the order induced by the expected value of $\frac{G(y)}{N(y)}$. It was found that the least-squares loss and ListNet mentioned above are both not consistent, because they do not use the right normalization; corrections to make them consistent were then proposed.

Recent work this year addressed the implicit loss in LambdaRank [**?**]. Generalizing the probability models above, they define a probability model for $\Pr[y \mid \vec{s}]$ as a mixture model, i.e.

$$\Pr[y \mid \vec{s}] = \sum_{\sigma \in P_n} \Pr(y; \vec{s}, \sigma) \Pr(\sigma; \vec{s}).$$

LambdaRank can be seen as an instance of this model, giving it a theoretical explanation. However, it is still unclear whether this surrogate is consistent; this is an avenue for future research.

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. [**?**] 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

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

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

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

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'
  - $C_{cl}(m) = \left(\sum_{i \in C_{1,u}, j \in C_{1,u}} y_{ij} + \sum_{i \in C_{2,d}, j \in C_{2,d}} y_{ij} \right) + \left(\sum_{i \in C_{1,u}, j \in C_{1,d}} (y_{ij} + y_{ji}) \right) + \left(\sum_{i \in C_{2,u}, j \in C_{2,d}} (y_{ij} + y_{ij}) \right)$
  - $C_{cl}(m') = \left(\sum_{i \in C_{1,u}, j \in C_{1,u}} y_{ij} + \sum_{i \in C_{2,d}, j \in C_{2,d}} y_{ij} \right) + \left(\sum_{i \in C_{1,u}, j \in C_{2,u}} (y_{ij} + y_{ij}) \right) + \left(\sum_{i \in C_{1,d}, j \in C_{2,d}} (y_{ij} + y_{ij}) \right)$

  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) [**?**], [**?**], 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.*

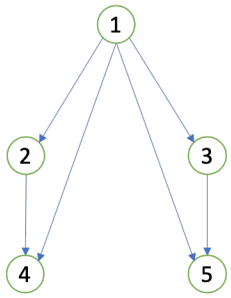
Proof. Consider we have m, and then we cluster $i, i+1$ to have m'. We need to show $\min_{l \in [|m|-1]} cost^m_{l,l+1} \leq \min_{l' \in [|m'|-1]} cost^{m'}_{l',l'+1}$. Notice that $cost^m_{l,l+1} = cost^{m'}_{l',l'+1}$ for $l = l' + 1$ and $l \neq i, i+1$. Also $cost^m_{i-1,i} \leq cost^{m'}_{i-1,i}$ 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 [**?**], 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 [**?**],

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

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 [**?**], 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 [**?**]
  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 $|\mathcal{M}| = (\frac{r}{2} + 1)!$.

7 Online Learning with Hierarchical Experts

In this section we look at an online decision making problem with structured/dependent experts. In online decision making, we have a finite number of experts N and a finite number of rounds T. At each round we have to play an expert after which a loss value for each expert is revealed. The goal is to minimize the total loss incurred. There are many algorithms for this problem but the update time at each round depends linearly on N. In cases where the experts are structured (such as paths in a graph), under some structural assumptions on the losses, we can get much faster algorithms such as the ones presented in [**?**] and [**?**].

Motivated by hierachical classification studied in [**?**], we look at experts arranged in a hierachical structure. For example, suppose every day one has to send one person from the Science school to attend a quiz. The exact topic of the quiz is unknown except for the fact that it will be a sub-field of Science. The experts are people with expertice in different branches of Science. Some have a broad knowledge of general topics such as Chemistry or Physics while some might have knowledge about very specific topics such as String Theory or Electrochemistry. In such cases the experts are naturally arranged in a tree. We formalize this as an online decision making problem with structure and provide a solution based on [**?**].

7.1 The Problem

We have a complete rooted binary tree $\texttt{Tr}$ of height n with each node representing an expert. Given two nodes y_1 and y_2, let $d_{\texttt{Tr}}(y_1, y_2)$ denote the tree distance between the two nodes, i.e., the length of the unique path from y_1 to y_2 in $\texttt{Tr}$. The learning proceeds in rounds of interaction between the learner $\mathcal{L}$ and the environment as follows:

Learner $\mathcal{L}$: For $t = 1, ..., T$,

- Play expert $\widehat{y}_t \in V(\texttt{Tr})$
- Receive best expert $y_t \in V(\texttt{Tr})$
- Incur loss $d_{\texttt{Tr}}(\widehat{y}_t, y_t)$
- Update model

We allow the learner to randomize the choice of $\widehat{y}_t$ but the environment's choice of y_t only depends on the probability distribution of $\widehat{y}_t$ and not on the actual expert played. If we look at the quiz example, we can assume we have one expert for every topic and the true label gives the topic of the quiz on the particular day. Note that the tree need not be binary, but we assume binary trees for the sake of simplicity.

Since there are only $N = 2^{n+1} - 1$ experts, we can associate a unique number to every expert in the range $[2^{n+1} - 1]$ and hence each expert can be represented using only $O(n)$ bits. A more natural representation of each expert is using the location with respect to the root. The vertices of the binary tree $\texttt{Tr}$ is taken to be $\{w \in \{0,1\}^* \mid |w| \leq n\}$ which is the set of binary sequences of length atmost n and the parent of a node w is obtained by removing the last bit from w. The root is repesented by the empty string ϵ, 0 is a child of ϵ, 01 is a child of 0 and so on. The two representations are polynomial time inter-convertible as long as the numbering in the former representation is done in some natural way. We'll use the string/word based representation throughout the rest of this discussion.

The goal of a learning algorithm is twofold: (a) Play step and Update step runs in time polynomial in n and does not depend on N. (b) For all sequences of true experts $y_1, ..., y_T$, the regret is sub-linear in T, where the regret of a learner $\mathcal{L}$ is the total loss incurred minus the loss of the best expert in hindsight:

$$R(\mathcal{L}) = \sum_{t=1}^{T} d_{\texttt{Tr}}(\widehat{y}_t, y_t) - \min_{y \in V(\texttt{Tr})} \sum_{t=1}^{T} d_{\texttt{Tr}}(y, y_t)$$

Non-linearity of the loss: Note that the loss $d_{\texttt{Tr}}$ is not linear in its first component with the string based representation of nodes. Nonetheless, can we represent each node y as a vector $\mathbf{u}_y$ in $\{0,1\}^d$ and assign a loss vector $\mathbf{l}_y \in \mathbb{R}^d_+$ to every node such that $d_{\texttt{Tr}}(y_1, y_2) = \mathbf{u}_{y_1}^T \mathbf{l}_{y_2}$? If we could, for small d, then we can apply the Component Hedge algorithm from [**?**]. But note that d has to be atleast the rank of the N-by-N matrix $M_{\texttt{Tr}}$ given by $M_{\texttt{Tr}}[y_1, y_2] = d_{\texttt{Tr}}(y_1, y_2)$. We expect the rank of $M_{\texttt{Tr}}$ to be high, although we are not aware of any known bounds.

7.2 Background

Our algorithm reduces this problem to the problem studied in Cortes et. al. 2015 [**?**] in which the authors study a class of problems in the online prediction with expert advice setting. In this setting, there is a prediction space $\mathcal{Y}$ and a finite number of experts. There is loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. At each round t, a prediction/advice (in $\mathcal{Y}$) from each expert is revealed using which the learner predicts a $\widehat{y}_t \in \mathcal{Y}$. Then the "correct" prediction y_t is revealed and the learner incurs loss $\ell(\widehat{y}_t, y_t)$.

To state the exact problem studied in the paper, we need some definitions. Let Σ be a finite aphabet. Here, we set $\Sigma = \{0, 1\}$. A finite automaton A over Σ is a tuple (Σ, Q, I, F, E) where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $E \subseteq Q \times (\Sigma \cup \epsilon) \times Q$ is a finite set of transitions (labeled edges) from state to state. We take the prediction space $\mathcal{Y} = \Sigma^*$. Every path from a state in I to a state in F is considered as an expert. At each round t, new labels on the edges are revealed. The advice of an expert ξ at time t, denoted using $\xi(t)$, is the label of the path ξ at time t.

The loss function $\ell : \Sigma^* \times \Sigma^* \to \mathbb{R}$ is given by a weighted finite state transducer WFST $\mathcal{U}$ over Σ. A WFST $\mathcal{U}$ is a tuple (Σ, Q, I, F, E) where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $E \subseteq Q \times (\Sigma \cup \epsilon)^2 \times \mathbb{R}_+ \times Q$ is a finite set of edges labeled by an input symbol, an output symbol and a weight. For any two strings $y_1, y_2 \in \Sigma^*$, let $P_{\mathcal{U}}(y_1, y_2)$ denote the set of paths from a state in I to a state in F with the input and output labels of the path being y_1 and y_2 respectively. The value of an input-output pair $(y_1, y_2) \in \Sigma^* \times \Sigma^*$ denoted by $\mathcal{U}(y_1, y_2)$ is the sum over all paths in $P_{\mathcal{U}}(y_1, y_2)$ the product of edge weights on the path.

$$\mathcal{U}(y_1, y_2) = \sum_{\pi \in P_{\mathcal{U}}(y_1, y_2)} \prod_{e \in \pi} \text{weight}(e)$$

where $\text{weight}((q_1, a_i, a_o, r, q_2)) = r$ and a path π is viewed as a multiset of edges from E. $\mathcal{U}(y_1, y_2)$ is defined to be 0 if $P_{\mathcal{U}}(y_1, y_2) = \emptyset$. The corresponding loss function $\ell_{\mathcal{U}} : \Sigma^* \times \Sigma^* \to \mathbb{R}$ is given by,

$$\ell_{\mathcal{U}}(y_1, y_2) = -\log(\mathcal{U}(y_1, y_2))$$

Finally, the online learning setting is as follows: There is a WFST $\mathcal{U}$ and a finite automaton A whose (fixed) set of paths is the set of experts.

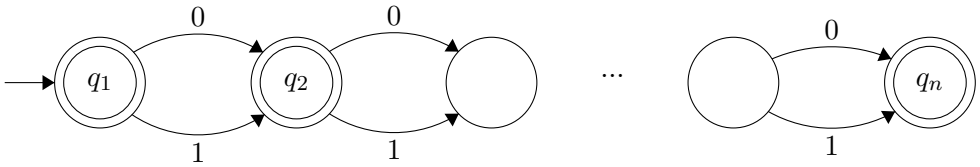
For rounds $t = 1, ..., T$,

- Receive new labels on edges of A
- Play $\widehat{y}_t \in \Sigma^*$ based on predictions $\xi(t)$ of each expert ξ
- Receive correct prediction $y_t \in \Sigma^*$
- Incur loss $\ell_{\mathcal{U}}(\widehat{y}_t, y_t)$
- Update model

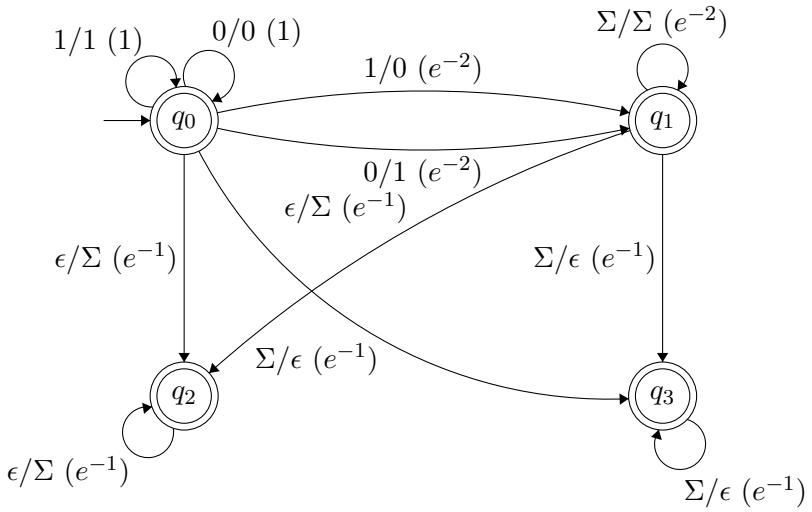
The paper provides a Follow The Perturbed Leader (FTPL) based algorithm for this problem that runs in time polynomial in the size of A, $\mathcal{U}$ and T if the underlying graphs are acyclic. The regret is also bounded by a constant.

7.3 Reduction

We can easily cast our hierarchical experts problem in the above language. Consider the following finite automaton A:



There are n linearly arranged states with two transitions from one state to the next (one for each label 0 and 1). q_1 is the only initial state and every state is a final state. Hence, every path uniquely represents a node in the tree Tr, specifically, the node represented by the label on the path. In our case, the advice of each expert/path remains the same for each round and is exacly the representation of the expert/node in Tr. Now we need to represent the loss d_{Tr} using a WFST $\mathcal{U}$. Such a $\mathcal{U}$ is given below. The input and output labels on the edges are separated by a '/' and the weight is given in paranthesis.



q_0 is the only initial state and every state is a final state. The underlying automaton of $\mathcal{U}$ automaton is determiistic and $P_{\mathcal{U}}(y_1, y_2)$ is a singleton for every pair of experts y_1 and y_2. The transducer ignores the longest common prefix and multiplies e^{-2} as long as both strings still have symbols to be read and then multiplies e^{-1} for the rest of the remaining string. It is easy to see that for any two experts/nodes in Tr, y_1 and y_2, $\mathcal{U}(y_1, y_2) = \exp(-d_{\text{Tr}}(y_1, y_2))$ giving us that $\ell_{\mathcal{U}}(y_1, y_2) = d_{\text{Tr}}(y_1, y_2)$. Note that $\mathcal{U}$ is not acyclic but can be made acyclic by making n copies and adding transitions from one copy to another.

Hence if we use the solution from [**?**] with the automaton A (same label for every time step) and loss given by $\mathcal{U}$, we get a solution running in time polynomial in $|A|$, $|\mathcal{U}|$ and T and hence polynomial in n and T. The regret bound we get is,

$$\mathbf{E}[R(\mathcal{L})] \leq 8n\sqrt{N(1 + 2\log(n))L_{min}} + 64n^2N(1 + 2\log(n))$$

The expectation is under the randomness of the algorithm and L_{min} is the cumulative loss of the best expert in hindsight.

7.4 Further Work

Although the runtime is polynomial in n and does not depend on N, the solution uses an FTPL style algorithm causing the memory used and time taken in round t to be linear in t which is not feasible in most cases. Also, the regret bound, inspite of being sublinear in T, depends on N which is large. One possible direction is to design a new online learning algorithm directly for the hierarchical experts case that does not have such limitations.

References