Advances in Ranking

Workshop at NIPS: December 11, 2009

Ranking problems are increasingly recognized as a new class of statistical learning problems that are distinct from the classical learning problems of classification and regression. Such problems arise in a wide variety of domains, such as information retrieval, natural language processing, collaborative filtering, and computational biology. Consequently, there has been increasing interest in ranking in recent years, with a variety of methods being developed and a whole host of new applications being discovered.

This workshop brings together researchers interested in the area to share their perspectives, identify persisting challenges as well as opportunities for meaningful dialogue and collaboration, and to discuss possible directions for further advances and applications in the future.

The workshop web site is:

http://web.mit.edu/shivani/www/Ranking-NIPS-09

Accepted papers will appear there in an electronic version of these proceedings. The papers in these proceedings are listed alphabetically, by first author.

We especially wish to thank the Program Committee (listed on page 3) for their support of this workshop; their efforts were essential in arriving at the strong program we have. Special thanks also to the invited speakers and authors for their work and help in creating a stimulating and productive workshop. We also wish to thank Microsoft Research for supporting the workshop.

Shivani Agarwal, Chris Burges, Koby Crammer
Organizing Committee

Schedule

Morning Session: 7:30 - 10:30 am

7:30 am	**Opening Remarks**
7:35 am	**Keynote Lecture:** ***Classical Methods for Ranked Data: A Review*** *Persi Diaconis*
8:20 am	***A Decision-Theoretic Model of Rank Aggregation*** *Tyler Lu, Craig Boutilier*
8:40 am	***Conditions for Recovery of Rankings from Partial Information*** *Srikanth Jagabathula, Devavrat Shah*
9:00 am	**Coffee Break**
9:15 am	**Invited Talk**: ***Statistical Ranking and Combinatorial Hodge Theory*** *Lek-Heng Lim*
9:50 am	***SampleRank: Learning Preferences from Atomic Gradients*** *Michael Wick, Khashayar Rohanimanesh, Aron Culotta, Andrew McCallum*
10:10 am	***Half Transductive Ranking*** *Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Corinna Cortes, Mehryar Mohri*

Poster Preview 10:30 am - 3:30 pm

Image Ranking with Eye Movements *Kitsuchart Pasupa, Sandor Szedmak, David R. Hardoon*
Spectral Analysis for Phylogenetic Trees *Paige Rinker, Daniel N. Rockmore*
Simple Risk Bounds for Position-Sensitive Max-Margin Ranking Algorithms *Stefan Riezler, Fabio de Bona*
Globally Optimal Partial Ranking for a Category of Data Mining Problems *Junshui Ma, Vladimir Svetnik*
Canberra Distance on Ranked Lists *Giuseppe Jurman, Samantha Riccadonna, Roberto Visintainer, Cesare Furlanello*
Large Scale Learning to Rank *D. Sculley*
Semi-Supervised Ranking via Ranking Regularization *Martin Szummer, Emine Yilmaz*
Filter, Rank, and Transfer the Knowledge: Learning to Chat *Sina Jafarpour, Chris Burges, Alan Ritter*

Afternoon Session 3:30 -6:30 pm

3:30 pm	**Invited Talk**: ***Matchbox: General Purpose, Large Scale, Online Bayesian Recommendations*** *Thore Graepel, Ralf Herbrich*
4:05 pm	***Scalable Online Learning to Rank from Clicks*** *Filip Radlinski, Aleksandrs Slivkins, Sreenivas Gollapudi*
4:25 pm	**Poster Session and Coffee Break**
5:20 pm	***Learning to Rank Through the Wisdom of Crowds*** *Jennifer Wortman Vaughan*
5:45 pm	**Panel Discussion** *Chris Burges, Olivier Chapelle, Marina Meila, Mehryar Mohri, Yoram Singer*
6:25 pm	**Closing Remarks**

Organization

Workshop Chairs

- **Shivani Agarwal**, Massachusetts Institute of Technology
- **Chris Burges**, Microsoft Research
- **Koby Crammer**, The Technion

Program Committee

- **Nir Ailon**, Google Research
- **Chiranjib Bhattacharyya**, Indian Institute of Science
- **Olivier Chapelle**, Yahoo! Research
- **Stephan Clemencon**, Telecom Paristech
- **Michael Collins**, Massachusetts Institute of Technology
- **Corinna Cortes**, Google Research
- **Thore Graepel**, Microsoft Research Cambridge
- **Ralf Herbrich**, Microsoft Research Cambridge
- **Alex Klementiev**, University of Illinois, Urbana-Champaign
- **Ping Li**, Cornell University
- **Lek-Heng Lim**, University of California, Berkeley
- **Tie-Yan Liu**, Microsoft Research Asia
- **Mehryar Mohri**, New York University
- **Tao Qin**, Microsoft Research Asia
- **Filip Radlinski**, Microsoft Research Cambridge
- **Stephen Robertson**, Microsoft Research Cambridge
- **Dan Roth**, University of Illinois, Urbana-Champaign
- **Cynthia Rudin**, Massachusetts Institute of Technology
- **Robert Schapire**, Princeton University
- **Devavrat Shah**, Massachusetts Institute of Technology
- **Yoram Singer**, Google Research
- **Kevin Small**, Tufts University
- **Krysta Svore**, Microsoft Research Redmond
- **Michael Taylor**, Microsoft Research Cambridge
- **Nicolas Vayatis**, Ecole Normale Supérieure de Cachan
- **Jason Weston**, Google Research
- **Emine Yilmaz**, Microsoft Research Cambridge
- **Tong Zhang**, Rutgers University

Contents

Half Transductive Ranking

Bing Bai[1] **Jason Weston**[2] **David Grangier**[1]
Ronan Collobert[1] **Corinna Cortes**[2] **Mehryar Mohri**[2][3]

[1]NEC Labs America, Princeton, NJ
{bbai, dgrangier, collober}@nec-labs.com
[2] Google Research, New York, NY
{jweston, corinna, mohri}@google.com
[3] NYU Courant Institute, New York, NY
mohri@cs.nyu.edu

Abstract

We study the standard retrieval task of ranking a fixed set of documents given a previously unseen query and pose it as the *half-transductive* ranking problem. The task is partly *transductive* as the document set is fixed. Existing transductive approaches are natural non-linear methods for this set, but have no direct out-of-sample extension. *Functional* approaches, on the other hand, can be applied to the unseen queries, but fail to exploit the availability of the document set in its full extent. This work introduces a *half-transductive* approach to benefit from the advantages of both *transductive* and *functional* approaches and show its empirical advantage in supervised ranking setups.

1 Introduction

The task of ranking a set of documents given a query is the core task of Information Retrieval (IR). In most setups, the set of documents to rank is fixed (or slowly growing) while the set of submitted queries is unknown. This environment gives rise to an interesting learning problem, *half-transductive ranking*, in which all the documents to rank are available at training time, while the test queries are only revealed once the model is trained.

Although frequent in Information Retrieval, previous literature mainly ignores this specific aspect and considers a setup in which the ranking model is learned to generalize to both new documents and new queries. Most ranking models learn a scoring function which assigns a real valued score to a feature vector describing a query/document pair. Given a query, the function is applied to each document and documents are ranked by decreasing scores. Models relying on this paradigm include the ranking perceptron [1], ranking SVM [2] or rankNet [3] among others. Hence, these approaches can be referred to as *functional*.

In contrast, *non-functional* approaches have been proposed in the literature for fully transductive setups, where the learnt similarity measure can only be assessed between objects available at training time. These approaches, such as as Isomap [4], Locally Linear Embedding [5] or Laplacian Eigenmaps [6], perform nonlinear dimensionality by learning a parameter vector for each training object. The comparison of the parameter vectors associated with a document and a query (e.g. through the Euclidean distance) allows for scoring the document against the query. Compared to functional approaches, non-functional strategies provide highly non-linear embeddings, while relying on simple optimization. Moreover, they do not tie the object embedding to a specific feature representation but rather focus on the inter-object relationships.

Despite these advantages, non-functional approaches are hard to apply in our setup, which is only half-transductive: all the documents are available for training, but the test queries are not. Of course,

one could advocate for using out-of-sample extensions that allow embedding a new object into the learned space [7]. Such a strategy is however not desirable in a retrieval context since: (i) it requires solving a (potentially expensive) optimization problem after the submission of a new query, which is the most time critical period for a retrieval system; and (ii) it neglects that available training queries could be used to identify a good strategy for embedding test queries.

This work proposes a direct solution to the *half-transductive ranking* problem by combining a functional model for embedding queries and a non-functional model for embedding objects. This approach hence benefits from the advantages of the non-functional approaches mentioned above, while retaining the efficiency and generalization ability of functional approaches for coping with new queries. In the following, we first briefly describe existing ranking models before introducing the proposed approach. Then, we empirically compare this solution to its alternatives. Finally, we draw some conclusions and delineate future research directions.

2 Functional and Non-Functional Models

Our ranking problem is classical. We are first given a fixed set of m objects, $\{y_i\}_{i=1}^m$, and we consider the task of ranking these objects given a new query x, unknown at training time. In the following, we classify models addressing this problem as either *functional* or *non-functional*.

2.1 Functional Models

These models represent query x and object y using a joint feature representation $\Phi(x, y)$ and scores the pair (x, y) using a function $x, y \rightarrow f_w(\Phi(x, y))$ parameterized by w. Instances of such functions include: functions linear in the feature space [1, 2], neural network functions [3] or Gaussian Processes [8]. These models can be learned with training objectives linked to the pairwise ranking loss, Normalized Discounted Cumulative Gain (NDCG) or Mean Averaged Precision (MAP).

Functional models also include Latent Semantic Indexing (LSI) [9], which corresponds to a linear function with a specific choice of features and parameter regularization, i.e. one can notice that LSI scoring, $f_W(x, y) = \phi(x)^{\mathrm{T}} W^{\mathrm{T}} W \phi(y), \quad \text{where} \quad \phi(x), \phi(y) \in \mathbb{R}^d, \quad W \in \mathbb{R}^{d \times n}, \quad n < d,$ is equivalent to a linear model, when one defines $w = W^{\mathrm{T}} W$ and $\Phi(x, y) = \phi(x)\phi(y)^{\mathrm{T}}$. In this case, the parameters are learned with a mean-squared reconstruction objective, by applying Singular Value Decomposition to the document-term matrix [9]. The same parameterization is also used by Supervised Semantic Indexing (SSI [10]), which learns the parameters from a supervised signal, to minimize the pairwise ranking loss.

2.2 Non-Functional Models

These models are *transductive* approaches which assign a parameter vector $v_i \in \mathbb{R}^d$ to each object y_i available at training time. Contrary to functional approaches, these strategies do not tie the object representation v_i to a feature representation $\Phi(\cdot)$ through a function. Instead, they consider a function f comparing examples in the embedding space $\mathbb{R}^d$, e.g. the Euclidean distance $f(y_i, y_j) = ||v_i - v_j||_2$ or the dot-product $f(y_i, y_j) = v_i \cdot v_j$, and learn $\{v_i\}_{i=1}^m$ by considering desired relationships between objects.

Several transductive embedding approaches have been proposed in the recent years. Most of them are rooted either in factor analysis (e.g. Principal Component Analysis) or Multi-Dimensional Scaling, including kernel PCA [11], Isomap [4], Locally Linear Embedding [5] and Laplacian Eigenmaps [6]. For example the latter embeds points by minimizing $\sum_{ij} L(v_i, v_j, A_{ij}) = \sum_{ij} A_{ij} ||v_i - v_j||_2^2$ where A is a similarity ("affinity") matrix, under constraints that ensure a trivial solution is not reached. An overall review of this family of methods can be found in [12].

Non-functional, transductive models hence offer a flexible framework to learn highly non-linear models and benefit from available information on relationships between objects. However, extensions to new objects – such as queries in our framework – are computationally costly and depends on the availability of information on the relationship between the new object and the already available objects [7].

3 Half-Transductive Ranking

In this work, we propose *Half-Transductive Ranking* to benefit from the advantages of both functional and non-functional approaches. Like transductive approaches, we assign a vector v_i to each object y_i in the fixed set to be ranked. However, we then consider a query x can be any object, not necessarily available during training. In that case, a function is applied to map x into the embedding space. Specifically, our scoring function is

$$f(x, y_i) = \langle W\phi(x), v_i \rangle = \phi(x)^{\mathrm{T}} W^{\mathrm{T}} v_i, \quad \text{where} \quad W \in \mathbb{R}^{d \times n}, \quad \forall i = 1 \dots m,\ v_i \in \mathbb{R}^d,$$

i.e. the dot product is used to compare embedded examples and a linear projection is used for mapping queries (nonlinear embeddings are implemented via $\phi(\cdot)$ being a fixed nonlinear map).

The parameters of our model, i.e. W and $\{v_i\}_{i=1}^m$ are learned to optimize the pairwise ranking loss. From a set of preference relations $\mathcal{R}$, where $\forall (x, y_+, y_-) \in \mathcal{R}$ expresses that document y_+ is preferred to document y_- for query x, we would like to learn f such that $f(x, y_+) > f(x, y_-)$. Like the margin ranking perceptron [1] or the ranking SVM [2], we minimize the hinge loss over such pairwise constraints,

$$L_{\mathrm{HTR}}(\mathcal{R}) = \sum_{(x, y_+, y_-) \in \mathcal{R}} \max(0, 1 - \phi(x)^{\mathrm{T}} W^{\mathrm{T}} v_+ + \phi(x)^{\mathrm{T}} W^{\mathrm{T}} v_-).$$

In our experiments, we noticed that a better linear mapping W could be obtained by learning to embed documents *functionally* as well and we introduce an hyperparameter γ to weight this second learning objective, i.e.

$$L(\mathcal{R}; \gamma) = L_{\mathrm{HTR}}(\mathcal{R}) + \gamma \sum_{(x, y_+, y_-) \in \mathcal{R}} \max(0, 1 - \phi(x)^{\mathrm{T}} W^{\mathrm{T}} W \phi(y_+) + \phi(x)^{\mathrm{T}} W^{\mathrm{T}} W \phi(y_-)).$$

This second objective is only used to provide further data for learning the functional mapping W. At test time, the functional mapping W is used solely for projecting the queries while documents are represented with their transductive vectors $\{v_i\}_{i=1}^m$.

We propose to train this model using stochastic gradient descent as this optimization strategy allows to achieve great scalability while avoiding poor local optima. The learning algorithm is hence simple: initially, the entries of W and $\{v_i\}_{i=1}^m$ are drawn randomly using from a normal distribution $\mathcal{N}(0, 1)$. Then, one iteratively picks a random triplet $(x, y_+, y_-) \in \mathcal{R}$ and updates the parameters according to the gradient of $(W, v_+, v_-) \rightarrow L(\{(x, y_+, y_-)\}; \gamma)$. One can note that this approach is computationally interesting since each iteration only updates W and two of the vectors $\{v_i\}_{i=1}^m$. Moreover, one can further exploit the potential sparsity of $\phi(x)$ for further gains. In our experiments, regularization is achieved through early stopping, i.e. one stops training when the performance over held-out validation data stops improving.

4 Experiments and Results

Our ranking experiments are carried out on Wikipedia and use the link structure of this dataset to build a large scale ranking task. Wikipedia provides a large document set with a meaningful, closed link structure. Compared to benchmark datasets, we can work with bigger query sets compared to TREC[1] and we can extract document-level features as opposed to LETOR[2].

The dataset we consider consists of 1,828,645 English Wikipedia documents and 24,667,286 links[3] which is randomly split into two portions, 70% for training (and validation) and 30% for testing. The following task is then considered: given a query document x, rank the other documents such that if x links to y then y should be highly ranked. The links are hence used as surrogates for relevance assessments and are not considered as features. Documents are represented with bag-of-word vectors, with a normalized TFIDF weighting [10].

Our results are compared to four alternative models: simple cosine similarity (TFIDF), LSI [9], margin ranking perceptron [1] and SSI [10]. We report results in two settings, (i) using only the top

[1] TREC datasets offer ~ 50 to ~ 500 queries, `http://trec.nist.gov`

[2] LETOR provides features for query/doc. pairs only, `http://learningtorank.spaces.live.com/`

[3] We removed links to calendar years as they provide little information while being very frequent.

Table 1: Document-document ranking on Wikipedia

Algorithm	30k words			2.5M words		
	Rank-Err (%)	MAP	P@10	Rank-Err (%)	MAP	P@10
TFIDF	1.62	0.33	0.16	0.84	0.43	0.19
LSI	1.28	0.35	0.17	0.72	0.43	0.19
Perceptron	0.41	0.48	0.21	0.35	0.49	0.22
SSI	0.30	0.52	0.23	0.16	0.55	0.24
Half-Transductive	**0.20**	**0.56**	**0.24**	**0.11**	**0.61**	**0.26**

30,000 most frequent words; (ii) using all 2.5 million words in Wikipedia. In the first setting, we rely on a margin perceptron with $\Phi(x, y) = \phi(x)\phi(y)^{\mathrm{T}}$. In the second setting, we rely on perceptron using Hash Kernels [13] as the previous feature choice does not allow the model to fit in memory. The various hyperparameters of the models have been selected to maximize the ranking loss over a held-out validation subset of the training data: this procedure selected an embedding dimension $n = 100$, a regularizer value $\gamma = 0.1$ and a hash size of $h = 6\mathrm{M}$ for the perceptron with hash kernel.

Table 1 reports the performance using the pairwise ranking error, Mean Averaged Precision, MAP, and precision at top 10, P@10. It shows that supervised models (Perceptron, SSI, Half-Transductive) outperform unsupervised ones (TFIDF, LSI). It also shows that embedding the data in a lower dimensional space is an effective capacity control mechanism (SSI versus Perceptron). More importantly, it shows that the additional non-linearity added by introducing the non-functional parameter vectors $\{v_i\}_{i=1}^m$ greatly enhances performance (SSI versus Half-Transductive). In fact, the half-transductive model outperforms all alternatives. A last remark on γ, this hyper-parameter is important: it pushes $\{v_i\}_{i=1}^m$ toward the solution provided by SSI which results in significant gain ($\gamma = 0$ yields a ranking error of 0.38% over the 30k-word test data compared to 0.20%).

5 Conclusions

This work studies the task of ranking a fixed set of documents given a previously unseen query. This classical IR setting yields us to introduce *half-transductive ranking*. Like transductive, non-functional embeddings, we learn a parameter vector to represent each of the known documents. At the same time, we learn a functional mapping to project test queries onto the document embedding space. This approach is different from test-time out-of-sample extensions for nonlinear dimensionality reduction techniques since we learn this function from training queries at training time. This approach is also different from classical functional ranking approaches which focus on models generalizing to both new documents and new queries while our strategy explicitly considers the availability of the documents at both train and test time. Our experiments over Wikipedia data demonstrate the advantages of our approach over functional ranking alternatives.

Acknowledgments Part of this work has been conducted while J. Weston was with NEC Labs America.

References

[1] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL)*, pages 263–270. Association for Computational Linguistics Morristown, NJ, USA, 2001.

[2] T. Joachims. Optimizing search engines using clickthrough data. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 133–142, 2002.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning (ICML)*, pages 89–96, New York, NY, USA, 2005. ACM Press.

[4] J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[5] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[6] M. Belkin and P. Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15(6):1373–1396, 2003.

[7] Y. Bengio, J.F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, isomap, MDS, eigenmaps, and spectral clustering. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[8] J. Guiver and E. Snelson. Learning to rank with softrank and gaussian processes. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 259–266, 2008.

[9] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[10] B. Bai, J. Weston, R. Collobert, and D. Grangier. Supervised semantic indexing. In *European Conference on Information Retrieval (ECIR)*, 2009.

[11] B. Schölkopf, A. Smola, and K. Müller. Kernel principal component analysis. *Advances in kernel methods: support vector learning*, pages 327–352, 1999.

[12] J. A. Lee and M Verleysen. *Nonlinear Dimensionality Reduction*. Springer, New York, 2007.

[13] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, A. Strehl, and V. Vishwanathan. Hash kernels. In *Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

Filter, Rank, and Transfer the Knowledge: Learning to Chat

Sina Jafarpour
Department of Computer Science
Princeton University
Princeton, NJ 08540
sina@cs.princeton.edu

Chris Burges
Microsoft Research
One Microsoft Way
Redmond, WA 98052
cburges@microsoft.com

Alan Ritter
Computer Science and Engineering
University of Washington
Seattle, WA 98195
aritter@cs.washington.edu

Abstract

Learning to chat is a fascinating machine learning task with many applications from user-modeling to artificial intelligence. However, most of the work to date relies on designing large hard-wired sets of rules. On the other hand, the growth of social networks on the web provides large quantities of conversational data, suggesting that the time is ripe to train chatbots in a more data driven way. A first step is to learn to chat by ranking the response repository to provide responses that are consistent with the user's expectations. Here we use a three phase ranking approach for predicting suitable responses to a query in a conversation. Sentences are first filtered, then efficiently ranked, and then more precisely re-ranked in order to select the most suitable response. The filtering is done using part-of-speech tagging, hierarchical clustering, and entropy analysis methods. The first phase ranking is performed by generating a large set of high-level grammatical and conceptual features, exploiting dictionaries and similarity measurement resources such as wikipedia similarity graphs, and by ranking using a boosted regression tree (MART) classifier. The more precise (conceptual) ranking is performed by designing more conceptual features obtained from similarity measurement resources such as query refinement and suggestion systems, sentence paraphrasing techniques, LDA topic modeling and structural clustering, and entropy analysis over wikipedia similarity graphs. The sentences are then ranked according to the confidence of a Transfer AdaBoost classifier, trained using transfer-learning methods in which a classification over a large corpus of noisy twitter and live-journal data is considered as the source domain, and the collaborative ranking of actively collected conversations, which are labeled in an online framework using user feedback, is considered as the destination domain. We give results on the performance of each step, and on the accuracy of our three phase ranking framework.

1 Introduction

The notion of designing automated chatbots is at least as old as Artificial Intelligence, since the Turing test requires that machines be endowed with conversational skills. However, most chatbots designed to date (such as ELIZA and A.L.I.C.E), are rule-based agents; a set of rules which drives the system's responses is hard-wired into the system. Although rule-based chatbots are suprisingly successful at engaging users for a while, they are brittle, with one false move starkly revealing their limitations, and they are rigid, with no way of displaying the kind of understanding and creativity that a truly engaging conversation requires.

Even just a decade ago, collecting large amounts of conversational data was impractical. Today however there is a flood of accessible conversational data, arising from social networks such as Twitter, from Live-Journal blogs, and from free semantic web applications. Moreover, new machine learning and ranking algorithms have been designed to manipulate large data sets in practical applications such as information retrieval, natural language processing, and web search ranking [1, 2]. In this paper, we investigate a framework for exploiting these large conversational data sets, using machine learning techniques with the objective of *learning* to chat.

Learning to chat can be regarded as a collaborative filtering problem, in which the suitability of a response is evaluated according to the extent that user likes or dislikes it, and our aim is to come up with an automatic ranking framework consistent with user evaluation. Having provided the system with a large corpus of millions of conversations (in our case twitter and live-journal data), we use an algorithm with three phases in order to rank the whole corpus and to provide a suitable response to the user query[1]. The first phase, which we call "filtering", filters the whole database of twitter responses, shrinking the set of candidate responses from two million to approximately two thousand.

We discuss what characteristics define a good filtering function, and we compare three filtering approaches based on part-of-speech tagging, hierarchical clustering, and entropy analysis. After identifying the filter set, following standard information retrieval approaches, we use a first-phase ranking in order to select more relevant candidate responses from the filter set. We designed a set of 300 features, including high-level features, grammatical features, and concept-based features, for the first phase ranker. The ranking is done using boosted regression trees (the MART classifier)[1].

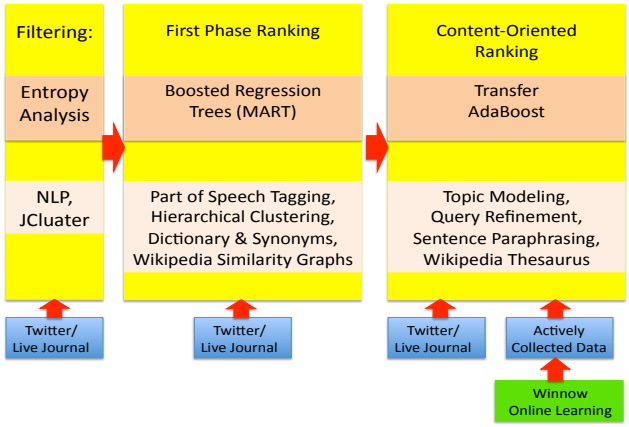


Figure 1: Three ranking phases for generating a response.

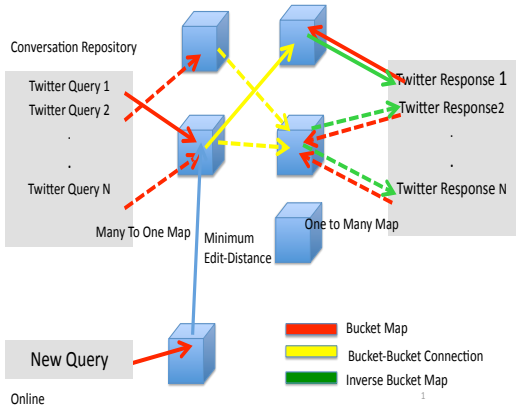


Figure 2: Mapping sentences into buckets for filtering.

Approaching the task of learning to chat by finding the best response in a large database of existing sentences may thus be viewed as a ranking problem. The next step (which we do not address in this paper) would be to alter the top ranked response to generate a better response, for example, by leveraging a model of the user's interests, or of the interests of users who had similar conversations. Here, we attempt to improve the selection of responses using a collaborative filtering step. Thus first, the unlabeled (and very noisy) twitter and live-journal datasets are used in the related classification task of distinguishing the *exact* response following a query, from a random response. A smaller data set is then actively collected and labeled by users, for which we designed an online-rating web-application for collecting data. We then use a transfer learning method (here, Transfer AdaBoost) [3, 4], in order to learn the destination (ranking) domain, exploiting the large number of instances available in the source (classification) domain. Having incorporated all two million source instances, we show that by then in addition exploiting the noise-free destination data, the accuracy of the transfer learning ranker increases on the task.

2 Filtering and First-Phase Ranking

Given a query and a corpus of millions of candidate sentences, a mechanism is needed to efficiently remove irrelevant sentences to reduce the set of candidate responses. A filtering is a many-to-one mapping from sentences to some

[1]We use the term "query" to denote the current sentence in a conversation, for which a response is desired, in analogy with the ranking task in IR.

string patterns known as *buckets*. Intuitively, one measure for selecting appropriate filtering methods is the extent to which either semantically related queries are mapped to the same bucket or their following responses are mapped to the same bucket. This intuitive argument can be made more precise using McDiarmid's inequality and tight concentration arguments:

Theorem 1 *For each query q^*, let $S_{q^*} = \{(q_1, r_1), \cdots, (q_n, r_n)\}$ be a set of appropriate responses r_i to q^*, coupled with the true preceding queries q_i in the repository, and let $\hat{r}_{q^*}$ be the most relevant response in S_{q^*}. Suppose M queries $\langle q_1^*, \cdots q_M^* \rangle$ are sampled iid from a distribution $\mathcal{D}$ for filtering, and for each q_i^*, ℓ samples $\langle r_1^i, \cdots, r_\ell^i \rangle$ are drawn iid from a distribution $\mathcal{D}_{q_i^*}$. Finally let $\mathcal{B}$ denote the bucket map, and suppose there exists a positive θ such that for all query q_i^*,*

$$\max \left\{ \Pr_{r_j^i} \left[\mathcal{B}(r_j^i) = \mathcal{B}(\hat{r}_{q_i^*}) \right], \Pr_{{r_j}^i} \left[\mathcal{B}(q_j^i) = \mathcal{B}(q_i^*) \right] \right\} \geq \theta.$$

Then with overwhelming probability, on average, each query has $\left(\theta - \sqrt{\frac{\log M}{\ell M}} \right) \ell$ suitable responses captured by mapping .

We investigated three heuristics for selecting the filtering function. In the first approach we used Part of Speech Tagging and Natural Language Processing methods [5]. Each sentence is first mapped to its chunked POS tag, and is then encoded to a binary string using Huffman coding. We use Huffman coding because it provides a hierarchical representation of the tags, i.e. tags with the same frequency (as a heuristic for similarity) have similar binary encoding. The second heuristic uses JCluster [6]. JCluster is a recursive algorithm for generating hierarchical clustering of the words. Let ABC denote trigrams in the corpus and b denotes the cluster of B. At each iteration JCluster splits each cluster to two sub-clusters so that $P(C|b)$ has minimal entropy. For instance, the words "you" and "u" are considered similar by JCluster. The 1-JCLuster bucket-map of each sentence can then be computed efficiently as follows: for each word of the sentence, the corresponding JCLuster index is extracted, and truncated to be limited to its most significant bits. The reason that 1-JCluster bucket-mapping is used is first to obtain a sufficiently large number of candidate sentences and second, we observe empirically that there exist a large number of similar queries that have the same 1-JCLuster representation but whose 2-JCLuster representation differs, which decreases the quality of the candidate responses.

The last heuristic combines an entropy analysis with hierarchical clustering. The major difference between this approach and the previous approaches is that in this approach a bucket map is many-to-many mapping, i.e. a set of binary strings, while in the previous approaches a bucket-map was always a single binary string. In this approach we first remove the common, low entropy words from each sentence. These words are then mapped to their JClusters indices which are truncated to their seven most-significant bits. Experimental results suggest that the third approach almost always provides more suitable responses comparing to the first two heuristics. Having performed the filtering to remove a large fraction of irrelevant sentences, a boosted regression tree classifier (MART) is used for finding suitable responses. A set of informative features were designed to train the MART classifier so that the ranking of the candidate filtered responses for a new query can be done using the confidence of the classifier. Our feature set incorporated the grammatical properties of the (query/response) pair, such as part of speech tagging, the parsing tree, the hierarchical clustering, and high-level features (such as whether the query is a question and the response is a statement, etc), and content-based features exploiting public resources such as dictionaries and synonyms, and also wikipedia similarity graphs.

The final problem is to provide labeling for training examples. We use the following method for labeling instances: for each query, its true response (the sentence following that in the twitter conversation) is labeled positive, and a random response is selected and labeled as negative. The labeled set is then divided uniformly into training, cross validation and test set. Having run the MART algorithm with 200 rounds, we observed that the test error continuously drops from 50% to 32% (which is 36% relative improvement); furthermore, no sign of over-fitting is observed. However, having tried the same learning task with 10 random negative responses, rather than just one, for each query, the test error dropped from 9% of random guess to 7.5% (which is only 16% relative improvement). Hence, although the test error still drops, which is a good news for the first-phase ranking, the improvement over the random guess decays as the number of random negative instances increases. The main reasons for this behavior are (1) the twitter data is very noisy; there are lots of generic responses, outliers, inconsistencies and errors in this labeling approach, and (2) chatting is a different task from distinguishing a random twitter response from the true response (although the two tasks are related). We tried to solve the first difficulty by designing an active data-collecting web application, and the second issue by using transfer learning methods.

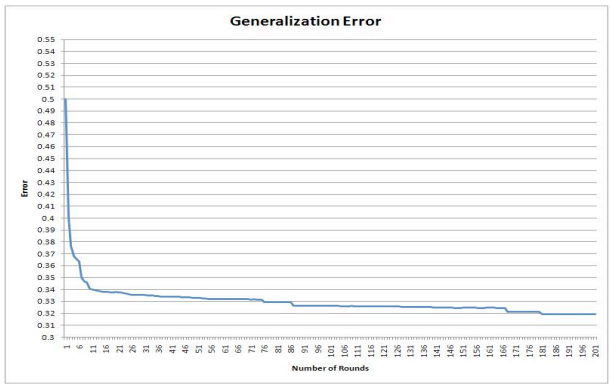


Figure 3: Test error vs. number of trees for MART(first phase ranker)

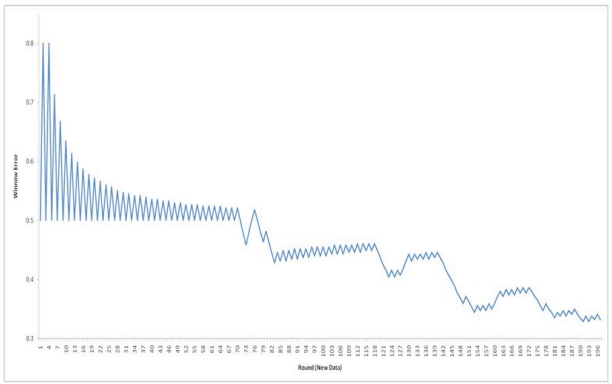


Figure 4: Winnow (online learning) generalization error vs. round

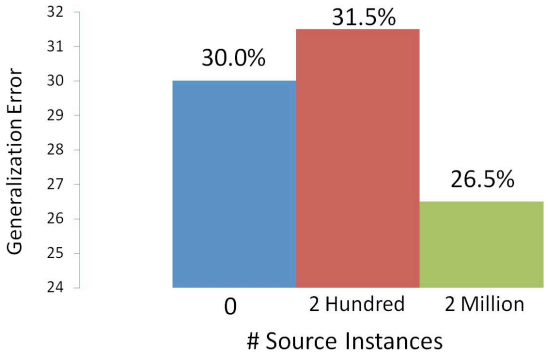


Figure 5: TrAdaBoost(transfer learning)

Figure 6: Generalization error of different phases of the ranking process

3 Collaborative Filtering, Online Learning, and Transfer Learning

To overcome the difficulties with Twitter and Live-Journal data in terms of noise and outlier effects in labeling, we designed a web-application game in order to actively collect less noisy training data. In this game, a query (with some conversation history), and a candidate response is provided to the user. Note that the main goal of this game is just collecting more accurate conversational data for the content-oriented learning phase. The web application provides a query and a candidate response at real-time. As a result, even though using MART might provide better responses, due to the necessity of real-time ranking and online learning requires fast and *online* learning methods, the *winnow* algorithm [7], an online multiplicative update algorithm which tends to be both fast and robust, is used for online candidate generation. The user then presses the like or dislike button. If dislike is pressed, the online classifier is updated and another response is generated, and if the user dislikes three such responses, then the user is asked to type a new, suitable response. If on the other hand the user liked one of the three responses, then the previous query is added to the chat history, and the new response becomes the next query. The online learning classifier is always updated after the user's action. Since the data is collected during an interactive game with labelers, collecting a large number of labeled examples requires a server managing the winnow algorithm and serving all human labelers simultaneously. This is where the term *collaborative learning* comes from. The Winnow is updated from all human labelers at the same time, and managing parallel sessions not only helps in speeding up the data collection rate, but it also provides advantages in training the winnow classifier, making the game more fun for the users. We enlisted the help of our local colleagues to play the game and as a result collected one thousand labeled samples.

Table 1: Some Examples of Automatically Generated Responses

Query	Suggested Response
I play football a lot	how long have you been playing ? i 've been on 2 years
I like Oranges	unless you eat like 10 acre of them at one time ... no
How is the weather in Seattle?	perfect for a day off ...

The question naturally follows: is it possible to exploit the large number of twitter instances, together with the data from the web game, in order to train a more accurate classifier? We propose a knowledge transfer solution for this question using the Transfer AdaBoost algorithm. Transfer learning tries to exploit the available information provided in one domain, known as the *source* domain, in order to improve or facilitate the task of learning another domain, called *destination domain*. In our chat-bot example, the source domain is the set of twitter and live-journal data, and the destination domain is the set of actively collected conversations.

Having identified the source domain, and the destination domain, we design a set of content-oriented features in order to capture as much information as possible between a query and the possible responses. The features include the first-phase ranking features as well as features learned from the topic modeling of a fresh twitter corpus using HMM and LDA Analyses [8], and extracted from query refinement tables and sentence paraphrasing techniques [9]. To train the TrAdaboost, we use 400 positive and 400 negative actively sampled conversations as the destination domain, and 2 million twitter conversations as the source domain. The destination set is then randomly divided to training and test sets with the same size. It turns out that with 2000 random twitter data the test error of the TrAdaBoost algorithm increases. However, whenever the whole twitter corpus is used, the error over the test set drops to 26.4%, comparing to 30.2% of not using knowledge transfer on a set of 200 positive and 200 negative human-labeled instances. Having trained TrAdaBoost, given a new query, the response is generated by first filtering the twitter corpus, then picking 100 sentences for which the trained MART algorithm has the highest confidence, and among them finding the sentence for which the TrAdaBoost has the highest confidence.

4 Conclusion

In this paper, we modeled the learning-to-chat problem as a collaborative ranking task, and we designed a triphase ranking framework consisting of filtering, first-phase ranking using MART, and conceptual ranking using Transfer AdaBoost. At each step, designing informative but not restrictive features play a key role, and to accomplish this a large variety of different similarity resources have been used. Like any other machine learning task, availability of a large corpus of conversational instances will lead to better ranking results and possibly more suitable responses. The Twitter data is extremely noisy and our work leads us to believe that training chat-bots with less noisy, massive training sets should further improve performance. A complementary task to collaborative filtering is user modeling, which is likely to provide valuable information regarding the response to be chosen. How to build such user models in response to the user's interaction with the chatbot is an interesting direction for future research.

References

[1] J. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics, Vol. 29, PP. 1189-1232*, 1999.

[2] C. Burges, T. Shaked, E. Renshaw, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML, pp. 89-96*, 2005.

[3] S. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE)*, 2009.

[4] W. Dai, Q. Yang, G. Xue, and Y. Yu, "Boosting for Transfer Learning," *ICML'07, PP. 193 - 200*, 2007.

[5] A. Bies, M. Ferguson, K. Katz, R. Macintyre, M. Contributors, V. Tredinnick, G. Kim, M. Marcinkiewicz, and B. Schasberger, "Bracketing Guidelines for Treebank II Style Penn Treebank Project," 1995.

[6] J. Goodman, "JCLUSTER, a fast simple clustering program that produces hierarchical, binary branching, tree structured clusters.," *http://research.microsoft.com/en-us/um/people/joshuago/*, 2009.

[7] N. Littlestone, "Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm," *Machine Learning PP. 285-318*, 1998.

[8] D. Blei, A. Ng, and M M. Jordan, "Latent Dirichlet Allocation," *JMLR, Vol. 3, PP. 993-1022*, 2003.

[9] S. Cucerzan and R. White, "Query suggestion based on user landing pages," *ACM SIGIR*, 2007.

[10] S. Young, M. Gasic, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu, "The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management," *Computer Speech and Language. http://mi.eng.cam.ac.uk/ farm2/papers/ygkm09.pdf*, 2009.

[11] K. Church, "A stochastic parts program and noun phrase parser for un- restricted texts," *Proceedings of the Second Conference on Applied Natural Language Processing*, 1988.

[12] S. DeRose, "Grammatical category disambiguation by statistical optimization," *Computational Linguistics, Vol. 14*, 1988.

[13] T. Hufmann, "Unsupervised Learning by Probabilistic Latent Semantic Analysis," *Machine Learning Journal, Vol. 42, PP. 177-196.*, 2001.

[14] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2003.

[15] J. Weizenbaum, "ELIZA a computer program for the study of natural language communication between man and machine," *Communications of the ACM, Vol.9 PP. 36-45*, January 1966.

[16] R. Wallace, "From Eliza to A.L.I.C.E," *http://www.alicebot.org/articles/wallace/eliza.html*, 2004.

[17] Y. Freund and R. Schapire, "Game theory, On-line prediction and Boosting," *In Proceedings of the Ninth Annual Conference on Computational Learning Theory, PP. 325-332*, 1996.

[18] Q. Wu, C. Burges, K. Svore, and J. Gao, "Ranking, Boosting, and Model Adaptation," in *Tecnical Report, MSR-TR-2008-109*, October 2008.

[19] C. Burges, "A tutorial on Support Vector Machines for pattern recognition," *Data Mining and Knowledge Discovery, Vol 2, PP. 955-974*, 1998.

[20] R. Caruana, S. Baluja, and T. Mitchell, "Using the Future to Sort Out the Present: Rankprop and Multitask Learning for Medical Risk Evaluation," *Advances in Neural Information Processing Systems (NIPS '95)*, 1995.

[21] G.Cavallant, N. Cesa-Bianchi, and C. Gentile, "Linear classification and selective sampling under low noise conditions," *Advances in Neural Information Processing Systems*, 2009.

[22] N. Craswell and M. Szummer, "Random walks on the click graph," *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval), pp. 239-246*, 2007.

[23] W. Dolan and C. Brockett, "Automatically Constructing a Corpus of Sentential Paraphrases," *Third International Workshop on Paraphrasing (IWP2005), Asia Federation of Natural Language Processing*, 2005.

Conditions for Recovery of Rankings from Partial Information *

Srikanth Jagabathula **Devavrat Shah** †

Abstract

We consider the problem of *exact* recovery from partial information of a non-negative function defined on the space of permutations of n elements. As the main result, we derive sufficient conditions, not only in terms of structural properties, but also in terms of a bound on the sparsity, for functions that can be recovered. We also propose an algorithm for recovery and prove that it finds the function with the sparsest support. In addition, we show that ℓ_1 optimization, which is usually used as a surrogate to find the sparsest solution, fails to recover the function.

1 Introduction

Functions over permutations serve as rich tools for modeling uncertainty in several important practical applications; however, they are limited because their size has a factorial blow-up. In order to overcome this difficulty, they are often approximated using a small subset of their Fourier coefficients. Such an approximation raises the natural question: which functions can be "well approximated" by a partial set of Fourier coefficients? This setup has several applications; as an example consider the *Identity Management Problem*, where the goal is to track the identities of n objects from noisy measurements of identities and positions. This problem is typically modeled using a distribution over permutations, which is often approximated using a partial set of Fourier coefficients. Recent work by [1, 2] deals with updating the distribution with new observations in the Fourier domain. Since only a partial set of Fourier coefficients are maintained, the final distribution must be recovered from only partial information.

To fix ideas, let S_n denote the set of permutations of n elements and $f : S_n \to \mathbb{R}_+$ denote a non-negative function. Our goal is to determine f when only partial information is available. Variants of this problem were considered before. One approach [3] was to obtain lower bounds on the energy contained in subsets of Fourier coefficients to obtain an approximation of a function with the "minimum" energy. This approach, however, does not naturally extend to the case of exact recovery. The problem of exact recovery, on the other hand, has been widely studied in the context of discrete-time functions. In particular, this problem received much attention in the area of *compressive sensing*, which has recently gained immense popularity. In the compressive sensing literature (see [4]), the basic question pertains to the *design* of an $m \times n$ "sensing" matrix A so that based on $y = Ax$ (or its noisy version), one can recover x. Our setup is different because in our case, the corresponding matrix A is *given* rather than a *design* choice. In addition, the corresponding matrix A does not satisfy the Restricted Isometry Property (RIP) [5], which is crucial for the approaches in the compressive sensing literature to work.

*A complete version of this paper is available at http://arxiv.org/abs/0910.0895

†Both authors are supported by NSF CAREER project CNS 0546590 and AFOSR Complex Networks program. They are with the Laboratory of Information and Decision Systems and Department of EECS, MIT. Emails: {jskanth, devavrat } @ mit.edu

2 Notation and Problem Statement

In this section, we describe the setup and the precise problem statement. As mentioned before, our interest is in learning non-negative real valued functions $f : S_n \to \mathbb{R}_+$, where $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$. The support of f is defined as $\text{supp}(f) = \{\sigma \in S_n : f(\sigma) \neq 0\}$, and its cardinality, $|\text{supp}(f)|$, is called the *sparsity* of f and denoted by K. We also call the sparsity of f its ℓ_0 norm and denote it by $|f|_0$.

In this paper, our goal is to learn f from partial information. In order to define the partial information we consider, we introduce the following notation. Consider a partition of n, i.e. an ordered tuple $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$, such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_r \geq 1$ and $n = \lambda_1 + \lambda_2 + \ldots + \lambda_r$; for example, $\lambda = (n-1, 1)$ is a partition of n. Now consider a partition of the n elements, $\{1, \ldots, n\}$, as per the λ partition, i.e. divide n elements into r bins with the ith bin having λ_i elements. It is easy to see that n elements can be divided as per the λ partition in $D_\lambda = n!/(\lambda_1!\lambda_2! \ldots \lambda_r!)$ distinct ways. Let the distinct partitions be denoted by $t_i, 1 \leq i \leq D_\lambda$[1]. For example, for $\lambda = (n-1, 1)$ there are $D_\lambda = n!/(n-1)! = n$ distinct partitions given by $t_i \equiv \{1, \ldots, i-1, i+1, \ldots, n\}\{i\},\ 1 \leq i \leq n$.

Now, given a permutation $\sigma \in S_n$, its action on t_i is defined through its action on the n elements of t_i; specifically, σ acts on t_i to yield the λ partition with the n elements permuted according to σ. In the above example with $\lambda = (n-1, 1)$, σ acts on t_i to give $t_{\sigma(i)}$, i.e. $\sigma : t_i \equiv \{1, \ldots, i-1, i+1, \ldots, n\}\{i\} \to t_{\sigma(i)} \equiv \{1, \ldots, \sigma(i)-1, \sigma(i)+1, \ldots, n\}\{\sigma(i)\}$. Now, for a given partition λ and a permutation $\sigma \in S_n$, define a 0/1 valued $D_\lambda \times D_\lambda$ matrix $M^\lambda(\sigma)$ as $M^\lambda_{ij}(\sigma) = 1$ if and only if $\sigma(t_j) = t_i$ for all $1 \leq i, j \leq D_\lambda$. The matrix $M^\lambda(\sigma)$ corresponds to a degree D_λ representation of the permutation group.

The partial information we consider in this paper is the Fourier transform of f at the representation M^λ for each λ; we call it the λ-partial information, and is defined as follows.

Definition 1 (λ-Partial Information)**.** *Given a function $f : S_n \to \mathbb{R}_+$ and partition λ, the Fourier Transform at representation M^λ, which we call the λ-partial information, is denoted by $\hat{f}(\lambda)$ and is defined as $\hat{f}(\lambda) = \sum_{\sigma \in S_n} f(\sigma) M^\lambda(\sigma)$.*

Recall the example of $\lambda = (n-1, 1)$ with f as a probability distribution on S_n. Then, $\hat{f}(\lambda)$ is an $n \times n$ matrix with the (i, j)th entry being the probability of element j mapped to element i under f. That is, $\hat{f}(\lambda)$ corresponds to the *first order* marginal of f in this case. The partial information we consider is related to the Fourier series coefficients of the function, which correspond to its Fourier transforms at irreducible representations. The motivation for considering this partial information is two fold: first, as we shall see, Fourier transforms at representations M^λ have natural interpretations, which makes it easy to collect in practical applications; second, each representation M^λ can be decomposed into Fourier transforms at irreducible representations.

It is helpful to think about the partial information we have in terms of a weighted bipartite graph. In particular, given λ-partial information, we think of each permutation σ as a complete matching in a $D_\lambda \times D_\lambda$ bipartite graph with the left nodes labeled from t_1 to t_{D_λ}, the right nodes labeled from t_1 to t_{D_λ}, and with an edge between t_i on left and t_j on right if and only if $\sigma(t_i) = t_j$. We refer to this representation as the λ-bipartite graph of σ. We can now think of the λ-partial information as a weighted bipartite graph obtained by the superposition of individual λ-bipartite graphs of permutations in the support of the function, weighted by their respective function values; we call this the λ-weighted bipartite graph. For brevity, we also use the terms bipartite graphs and weighted bipartite graphs when the partition λ they correspond to is clear from the context .

Our goal is to recover f from the given partial information. Of course, since only partial information is available, such recovery is not possible in general. Therefore, our goal is to: (a) characterize the class of functions that can be recovered exactly from only partial information, and (b) design a procedure to perform recovery.

[1]To keep notation simple, we use t_i instead of t_i^λ that takes explicit dependence on λ into account.

3 Main Results

In this section, we provide our main results. In order to understand what kind of functions can be recovered, we first take an example.

Example 1. *For any $n \geq 4$, consider the four permutations $\sigma_1 = (1,2)$, $\sigma_2 = (3,4)$, $\sigma_3 = (1,2)(3,4)$ and $\sigma_4 = \mathrm{id}$, where* id *is the identity permutation. Let $f\colon S_n \to \mathbb{R}_+$ be a function such that $f(\sigma_1) = p_1, f(\sigma_2) = p_2, f(\sigma_3) = p_3$ and $f(\sigma) = 0$ for $\sigma \neq \sigma_1, \sigma_2, \sigma_3$. When $\lambda = (n-1,1)$, it is easy to see that $M^\lambda(\sigma_1) + M^\lambda(\sigma_2) = M^\lambda(\sigma_3) + M^\lambda(\sigma_4)$. Without loss of generality, let $p_1 \leq p_2$. Then, $\hat{f}(\lambda) = \sum_{i=1}^{3} p_i M^\lambda(\sigma_i) = (p_2 - p_1)M^\lambda(\sigma_2) + (p_3 + p_1)M^\lambda(\sigma_3) + p_1 M^\lambda(\sigma_4)$. Thus, function g with $g(\sigma_2) = p_2 - p_1$, $g(\sigma_3) = p_3 + p_1$, $g(\sigma_4) = p_1$ and $g(\sigma) = 0$ for all other $\sigma \in S_n$ is such that $\hat{g}(\lambda) = \hat{f}(\lambda)$; moreover, $\|g\|_0 = 3 = \|f\|_0$.*

It is easy to see from Example 1 that, without any additional information, it is impossible to recover the underlying function uniquely from the given information. In order to understand this further, it is convenient to imagine recovering the function from its λ-partial information by decomposing the λ-weighted bipartite graph into its corresponding λ-bipartite graphs and weights. Note that the bipartite graph of σ_1 is completely contained in the superimposition of the bipartite graphs of σ_2 and σ_3. Therefore, we can decompose the weighted bipartite graph in two distinct ways: (a) by "peeling-off" σ_1, followed by σ_2 and σ_3, yielding function f, and (b) by "absorbing" σ_1 into σ_2 and σ_3, resulting in permutations $\sigma_2, \sigma_3, \sigma_4$, and thereby the function g. The confusion here is regarding the inclusion of the permutation σ_1. This confusion has arisen because σ_1 does not have a "signature" or a "witness" in the data, which could help us ascertain its presence; the "signature" or "witness" of σ_1 is swamped by the permutations σ_2 and σ_3. Therefore, in order to be able to recover f we require that each permutation possess a unique "signature" or "witness" in the data that can be used to identify it. In particular, we require that each permutation in the support of the underlying function have at least one edge in its bipartite graph that is not contained in the bipartite graph of any other permutation in the support; this edge serves as the "witness" or the "signature" of this permutation. This condition ensures that the bipartite graph of no permutation is completely contained in the superposition of the bipartite graphs of other permutations.

It turns out that the "witness" condition we imposed above is not sufficient for exact recovery. First, note that for exact recovery to be possible, the function values must all be distinct (for a simple counter example, consider the function with only σ_1, σ_2, as defined in Example 1, in its support and equal function values). However, as will be shown shortly, it is not sufficient for the function values to be all distinct. In fact, we want something stronger: the sum of the function values corresponding to any one subset of permutations should be distinct from the sum of function values over any other subset. The necessity of this condition is illustrated through the following example. Consider *any* four permutations $\sigma_1, \sigma_2, \sigma_3$ and σ_4, possibly satisfying the witness condition. We construct four new permutations by appending two new elements $n+1, n+2$ as follows: $\sigma_1' = \sigma(n+1)(n+2), \sigma_2' = \sigma_2(n+1)(n+2), \sigma_3' = \sigma_3(n+1, n+2), \sigma_4' = \sigma_4(n+1, n+2)$, where we assume that all permutations are represented using the cycle notation. Now suppose we form a function f by assigning the four permutations weights p_1, p_2, p_3 and p_4 such that $p_1 + p_2 = p_3 + p_4$ respectively. It is easy to see that given the λ-partial information corresponding to $\lambda = (n-1,1)$, we can also decompose it into the function g with support $\sigma_1'' = \sigma(n+1, n+2), \sigma_2'' = \sigma_2(n+1, n+2), \sigma_3'' = \sigma_3(n+1)(n+2), \sigma_4'' = \sigma_4(n+1)(n+2)$. As we show shortly, the above two conditions are, in fact, sufficient for exact recovery. Formally, the conditions we impose on f are as follows.

Condition 1 (Sufficiency Conditions)**.** Let f satisfy the following:

- *Unique Witness*: for any $\sigma \in \mathrm{supp}\,(f)$, there exists $1 \leq i_\sigma, j_\sigma \leq D_\lambda$ such that $M^\lambda_{i_\sigma j_\sigma}(\sigma) = 1$, but $M^\lambda_{i_\sigma j_\sigma}(\sigma') = 0$, for all $\sigma' (\neq \sigma) \in \mathrm{supp}\,(f)$.

- *Linear Independence*: for any collection of integers $c_1, \ldots, c_K$ taking values in $\{-K, \ldots, K\}$, $\sum_{k=1}^{K} c_k p_k \neq 0$, unless $c_1 = \ldots = c_K = 0$.

Note that, as shown through the above examples, if either of these conditions is not satisfied, then, in general, it is not possible to recover the function even if we impose a bound on the sparsity of the function. Thus, in that sense, these conditions are necessary.

We prove that Condition 1 is sufficient for exact recovery by proposing an algorithm called the sparsest-fit algorithm (the rationale for the name will become apparent shortly) and proving that the algorithm recovers f from $\hat{f}(\lambda)$ whenever f satisfies Condition 1. The description of the algorithm is given in [6]. For reasons that will become apparent soon, we discuss the complexity of the algorithm towards the end of the section. We now have the following theorem.

Theorem 1. *When f satisfies Condition 1, the sparsest-fit algorithm recovers f exactly from $\hat{f}(\lambda)$. Thus, Condition 1 is sufficient for exact recovery of f from $\hat{f}(\lambda)$.*

The characterization we have provided above is in terms of the structural properties of the underlying function. The next natural issue that arises is how Condition 1 translates into the "complexity" of the functions that can be recovered. A natural measure of complexity is the sparsity of the function. Therefore, we try to understand how Condition 1 translates into a bound on the sparsity of the functions that can be recovered. In order to understand this question, it is helpful to think about the problem we are trying to solve as a game between Alice and Bob, in which Alice picks a function $f\colon S_n \to \mathbb{R}_+$, constructs $\hat{f}(\lambda)$ and gives it to Bob, while Bob tries to correctly guess the function f used by Alice. Clearly, if Alice is adversarial, even if Alice and Bob agree on a bound on the sparsity of f, Bob has no chance of winning (cf. Example 1). However, suppose Alice is not adversarial and chooses a function with a given sparsity bound randomly. Then, it is reasonable to wonder if Bob can guess the function correctly with a high probability. Put differently, we know that recovery, even with a sparsity bound, is not possible in the worst-case; however, is it possible in the average case? In order to answer this question, we first propose a natural random generative model for the function with a given sparsity. Then, for a given partition λ, we obtain a sparsity bound $K(\lambda)$ so that if $K < K(\lambda)$ then recovery of f generated according to the generative model from $\hat{f}(\lambda)$ is possible with high probability. Note that this characterization is similar to the characterization for functions in the compressive sensing literature. We discuss this connection in detail in the next section. We now describe the random model and then provide our results on the sparsity bound $K(\lambda)$ for all partitions λ.

Definition 2 (Random Model)**.** *Given $K \in \mathbb{Z}_+$ and an interval $\mathscr{C} = [a, b]$, $0 < a < b$, a random function f with sparsity K and values in $\mathscr{C}$ is generated as follows: choose K permutations from S_n independently and uniformly at random [2], say $\sigma_1, \ldots, \sigma_K$; select K values from $\mathscr{C}$ uniformly at random, say $p_1, \ldots, p_K$; then function f is defined as $f(\sigma) = p_i$ for $\sigma = \sigma_i$, $1 \le i \le K$ and $f(\sigma) = 0$ otherwise. We will denote this model as $R(K, \mathscr{C})$.*

In what follows, we spell out the result starting with a few specific cases so as to better explain the dependency of $K(\lambda)$ on D_λ.

Case 1: $\lambda = (n-1, 1)$. Here $D_\lambda = n$ and $\hat{f}(\lambda)$ provides the *first order* marginal information. As stated next, for this case, the achievable recoverability threshold $K(\lambda)$ scales[3] as $n \log n$.

Theorem 2. *A function f generated as per Definition 2 can be recovered from $\hat{f}(\lambda)$ by the sparsest-fit algorithm with probability $1 - o(1)$ as long as $K \le (1-\varepsilon) n \log n$ for any fixed $\varepsilon > 0$.*

Case 2: $\lambda = (n-m, m)$ with $1 < m = O(1)$. Here $D_\lambda = \Theta(n^m)$ and $\hat{f}(\lambda)$ provides the *mth order* marginal information. As stated next, for this case, we find that $K(\lambda)$ scales at least as $n^m \log n$.

Theorem 3. *A function f generated as per Definition 2 can be recovered from $\hat{f}(\lambda)$ by the sparsest-fit algorithm for $\lambda = (n-m, m), m = O(1)$, with probability $1 - o(1)$ as long as $K \le \frac{(1-\varepsilon)}{m!} n^m \log n$ for any fixed $\varepsilon > 0$.*

In general, for any λ with $\lambda_1 = n - m$ and $m = O(1)$, arguments of Theorem 3 can be adapted to show that $K(\lambda)$ scales as $n^m \log n$. Theorems 2 and 3 suggest that the

[2] Throughout, we will assume that the random selection is done *with* replacement.

[3] Throughout this paper, by log we mean the natural logarithm, i.e. $\log_e$, unless otherwise stated.

recoverability threshold scales $D_\lambda \log D_\lambda$ for $\lambda = (\lambda_1, \ldots, \lambda_r)$ with $\lambda_1 = n - m$ for $m = O(1)$. Next, we consider the case of more general λ.

Case 3: $\lambda = (\lambda_1, \ldots, \lambda_r)$ with $\lambda_1 = n - O\left(n^{2/9-\delta}\right)$ for any $\delta > 0$. As stated next, for this case, the recoverability threshold $K(\lambda)$ scales at least as $D_\lambda \log\log D_\lambda$.

Theorem 4. *A function f generated as per Definition 2 can be recovered from $\hat{f}(\lambda)$ by the sparsest-fit algorithm for $\lambda = (\lambda_1, \ldots, \lambda_r)$ with $\lambda_1 = n - n^{2/9-\delta}$ for any $\delta > 0$, with probability $1 - o(1)$ as long as $K \leq (1-\varepsilon) D_\lambda \log\log D_\lambda$ for any fixed $\varepsilon > 0$.*

Case 4: Any $\lambda = (\lambda_1, \ldots, \lambda_r)$. The results stated thus far suggest that the threshold is essentially D_λ, ignoring the logarithm term. For general λ, we establish a bound on $K(\lambda)$ as stated in Theorem 5 below. Before stating the result, we introduce some notation. For given λ, define $\alpha = (\alpha_1, \ldots, \alpha_r)$ with $\alpha_i = \lambda_i / n,\ 1 \leq i \leq r$. Let $H(\alpha) = -\sum_{i=1}^r \alpha_i \log \alpha_i$ and $H'(\alpha) = -\sum_{i=2}^r \alpha_i \log \alpha_i$.

Theorem 5. *Given $\lambda = (\lambda_1, \ldots, \lambda_r)$, a function f generated as per Definition 2 can be recovered from $\hat{f}(\lambda)$ by the sparsest-fit algorithm with probability $1 - o(1)$ as long as $K \leq C\, D_\lambda^{\gamma(\alpha)}$ where*

$$\gamma(\alpha) = \frac{M}{M+1}\left[1 - O(1)\frac{H(\alpha) - H'(\alpha)}{H(\alpha)}\right], \quad \textit{with} \quad M = \left\lfloor \frac{1}{1-\alpha_1} \right\rfloor$$

and $0 < C < \infty$ is a constant.

At a first glance, the above result seems very different from the crisp formulas of Theorems 2-4. Therefore, let us consider a few special cases. First, observe that as $\alpha_1 \uparrow 1$, $M/(M+1) \to 1$. Further, as stated in Lemma 1, $H'(\alpha)/H(\alpha) \to 1$. Thus, we find that the bound on sparsity essentially scales as D_λ. Note that cases 1, 2 and 3 fall squarely under this scenario since $\alpha_1 = \lambda_1/n = 1 - o(1)$. Thus, this general result contains the results of Theorems 2-4 (ignoring the logarithm terms). Next, consider the other extreme of $\alpha_1 \downarrow 0$. Then, $M \to 1$ and again by Lemma 1, $H'(\alpha)/H(\alpha) \to 1$. Therefore, the bound on sparsity scales as $\sqrt{D_\lambda}$. This ought to be the case because for $\lambda = (1, \ldots, 1)$ we have $\alpha_1 = 1/n \to 1$, $D_\lambda = n!$, and unique signature property holds only up to $o(\sqrt{D_\lambda}) = o(\sqrt{n!})$ due to the standard Birthday paradox. In summary, Theorem 5 appears reasonably tight for the general form of partial information λ. We now state the Lemma 1 used above.

Lemma 1. *Consider any $\alpha = (\alpha_1, \ldots, \alpha_r)$ with $1 \geq \alpha_1 \geq \cdots \geq \alpha_r \geq 0$ and $\sum_{i=1}^r \alpha_r = 1$. Then, $\lim_{\alpha_1 \uparrow 1} H'(\alpha)/H(\alpha) = 1$ and $\lim_{\alpha_1 \downarrow 0} H'(\alpha)/H(\alpha) = 1$.*

This finishes all the results on sparsity bounds. Coming back to the complexity of the sparsest-fit algorithm, for $K = O(D_\lambda \log D_\lambda)$, the algorithm has a running time complexity of $O\left(\exp(\log^2 D_\lambda)\right)$ with a high probability under the random model $R(K, \mathscr{C})$. Thus, the complexity is quasi-polynomial in the input size, which is "close" to $\text{poly}(D_\lambda)$, the best running-time complexity we can hope for since the input size is D_λ^2. The worst-case time complexity is, however, $O(2^K)$; note that this is exponential in sparsity and not input size.

4 The sparsest solution

In this section, we explore the connections of our problem setup with the setups considered in the area of compressive sensing. As mentioned before, the typical question in the area of compressive sensing pertains to the design of the sensing matrix A such that it is possible to recover x from partial measurements $y = Ax$. The typical result in this context corresponds to a bound on the sparsity of x for which exact recovery is possible, provided the matrix A satisfies some conditions (viz. RIP conditions). Our problem can be cast in this form by, with some abuse of notation, imagining the function f as an $n! \times 1$ vector and the data $\hat{f}(\lambda)$ as the $D_\lambda^2 \times 1$ vector. Then $\hat{f}(\lambda) = Af$, where A is an $D_\lambda^2 \times n!$ matrix and each column corresponds to the matrix $M^\lambda(\sigma)$ for a certain permutation σ. As mentioned before, however, the matrix A in our setup is given rather than a design choice.

The typical approach in compressive sensing is to recover x by finding the sparsest solution (through ℓ_0 optimization) consistent with the given information y. Since finding the sparsest solution is, in general, computationally hard, its convex relaxation, ℓ_1 optimization, is considered. The main results, then, correspond to conditions under which the two approaches yield the correct solution. Such conditions, and the sparsity bounds we obtained above, raise the natural question of whether the two approaches will work in our setup. We answer the first question in the affirmative and the second one in the negative. In particular, suppose the ℓ_0 optimization problem we solve is

$$\begin{aligned} &\text{minimize} && \|g\|_0 && \text{over} && g : S_n \to \mathbb{R}_+ \\ &\text{subject to} && \hat{g}(\lambda) = \hat{f}(\lambda). \end{aligned} \tag{1}$$

Then, we have the following theorem.

Theorem 6. *Consider a function f that satisfies Condition 1. Then, f is the unique solution to the ℓ_0 optimization problem* (1)*.*

It now follows from Theorem 1 that the sparsest-fit algorithm indeed identifies the sparsest solution. Note that Condition 1 is essentially necessary for f to be the unique sparsest solution. In particular, the examples given above Condition 1 indicate that when either of the conditions is not satisfied, then f may not be the unique sparsest solution.

Finally, the convex relaxation, ℓ_1 minimization ((1) with the objective replaced by $\|g\|_1$), fails to recover the function, as stated in the following theorem.

Theorem 7. *Consider a function f randomly generated as per Definition 2 with sparsity $K \geq 2$. Then, with probability $1 - o(1)$ the solution to ℓ_1 minimization is not unique.*

5 Conclusion

In summary, we considered the problem of *exactly* recovering a non-negative function, defined over the space of permutations, given partial information. We derived sufficient conditions, not only in terms of structural properties, but also in terms of sparsity bounds, for functions that can be recovered exactly from only partial information. If either of the conditions we imposed in not satisfied, then, in general, it is not possible to recover the underlying function even if we impose bounds on sparsity. We also proposed the sparsest-fit algorithm that recovers f whenever Condtion 1 is satisfied. We also showed that the sparsest-fit algorithm indeed finds the sparsest solution and that ℓ_1 optimization fails to recover the function. Natural next steps include extending our approach to other forms of partial information and to the scenario where the underlying function is not exactly, but only approximately, sparse.

References

[1] J. Huang, C. Guestrin, and L. Guibas. Efficient inference for distributions on permutations. *Advances in Neural Information Processing Systems*, 20:697–704, 2008.

[2] R. Kondor, A. Howard, and T. Jebara. Multi-object tracking with representations of the symmetric group. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, 2007.

[3] K.L. Kueh, T. Olson, D. Rockmore, and K.S. Tan. Nonlinear approximation theory on compact groups. *Journal of Fourier Analysis and Applications*, 7(3):257–281, 2001.

[4] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2005.

[5] S. Jagabathula and D. Shah. Inferring rankings under constrained sensing. In *NIPS*, pages 7–1, 2008.

[6] S. Jagabathula and D. Shah. Inferring rankings using constrained sensing. *http://arxiv.org/abs/0910.0063*, 2009.

Canberra distance on ranked lists

Giuseppe Jurman, Samantha Riccadonna, Roberto Visintainer and Cesare Furlanello*
Fondazione Bruno Kessler
I-38123 Povo (Trento), Italy
{jurman, riccadonna, visintainer, furlan}@fbk.eu

Abstract

The Canberra distance is the sum of absolute values of the differences between ranks divided by their sum, thus it is a weighted version of the $L1$ distance. As a metric on permutation groups, the Canberra distance is a measure of disarray for ranked lists, where rank differences in top positions need to pay higher penalties than movements in the bottom part of the lists. Here we describe the distance by assessing its main statistical properties and we show extensions to partial ranked lists. We conclude providing two examples of use in functional genomics.

1 Introduction

The Canberra distance, introduced by Lance and Williams in the late sixties [1] as a software metric, is a weighted version of the classic $L1$ or Spearman's footrule which naturally extends to a metric on symmetric groups. This metric is particularly useful when comparing ranked lists in functional genomics. In fact, in the case of panels of biomarkers, we may require to differently penalize rank differences in the higher portion of the lists rather than those in the bottom section. The theory of metric methods on ranked data is relatively recent: an exhaustive reference is [2]. Pioneer work in this fiel has been carried on by Hoeffding [3] mixing statistics and permutation group theory and by Diaconis on distances for symmetric groups [4] *i.e.*, right-invariant distances. A few metrics on permutation groups are known, and for some of them a description in terms of their key properties (expected value, variance, distribution, extremal values) has been stated and proved [5]. We previously introduced the use of the Canberra distance in machine learning for computational biology and define an indicator of stability for ranked lists of biomarkers [6]. Although electively oriented towards computational biology, the potentiality of the Canberra distance can be of interest for the broader community dealing with rank comparison on general problems where the difference between relevant and negliglible objects (*i.e.*, with high and low rank) plays an important role.

Here we explicitly fin the approximated and exact (wherever feasible) expressions for the expectation value, the variance and the maximum value and argument, together with assessing its normality. All the aforementioned are expressed as functions of the harmonic numbers, the partial sums of the harmonic series summing the reciprocals of all natural numbers. Although a piece of classical number theory [7], computing with harmonic number is not trivial. Only recently papers have appeared providing closed[1] or at least recursive forms for more complex expressions involving sums and products of harmonic numbers [8].

After describing the key elements, here we discuss extensions of the Canberra distance to partial lists. In the last part, we show two applications for functional genomics. A molecular profilin and a study of variability of Canberra distances for differentially expressed biomarker lists are presented. All the algorithms described are implemented as functions of the `mlpy` Open Source Python library for machine learning, freely available at `https://mlpy.fbk.eu`.

*Corresponding author. Lab website: `http://mpba.fbk.eu`

[1] A closed form is an expression of a variable n that can be computed by applying a number of basic operations not depending on n; $\frac{n(n+1)}{2}$ is a closed form for the sum of the fi st n naturals, while $\sum_{i=1}^{n} i$ is not.

2 Properties of the Canberra distance

Preliminaries Given two real-valued vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, their Canberra distance and its natural extension to a distance on the permutation group on p objects S_p are define as follows:

$$\mathrm{Ca}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \frac{|\mathbf{x}_i - \mathbf{y}_i|}{|\mathbf{x}_i| + |\mathbf{y}_i|} \quad \text{and} \quad \mathrm{Ca}(\tau, \sigma) = \sum_{i=1}^{p} \frac{|\tau(i) - \sigma(i)|}{\tau(i) + \sigma(i)}, \text{ for } \tau, \sigma \in S_p.$$

Because of right-invariance, definin $\mathrm{Ca}_I(\sigma) = \mathrm{Ca}(\sigma, \mathrm{Id}_{S_p})$, the identity $\mathrm{Ca}(\sigma, \tau) = \mathrm{Ca}_I(\sigma\tau^{-1})$ holds and it will be repeatedly used hereafter.

For a natural $n \in \mathbb{N}$ and a positive exponent $a \in \mathbb{R}$, the generalized harmonic number $H_n^{(a)}$ is define as $H_n^{(a)} = \sum_{i=1}^{n} \frac{1}{i^a}$; when $a = 1$ the bracketed exponent is omitted and one speaks of harmonic number in short. Relations involving $H_n^{(a)}$ are dealt with by the identities proved in [8] expressing $H_n^{(a)}$ as functions of H_n. By Euler's formula, H_n can be expanded as

$$H_n = \log(n) + \gamma - \sum_{b=1}^{\infty} \frac{B_b}{n^b},$$

where $\gamma \approx 0.57721$ is the Euler-Mascheroni constant and B_b denotes the b-th Bernoulli number; truncating the formula at $b = 1$ the approximation reads as follows:

$$H_n = \log(n) + \gamma + \frac{1}{2n} + o(n^{-1}) \,. \tag{1}$$

In most of the proofs we follow the notations, the strategy and the results of Hoeffding's paper [3]: accordingly, we use the shorthands

$$c_p(i,j) = \frac{|i-j|}{i+j}, \quad f_p(i) = \frac{1}{p}\sum_{j=1}^{p} c_p(i,j) \quad \text{and}$$

$$d_p(i,j) = c_p(i,j) - \frac{1}{p}\sum_{g=1}^{p} c_p(i,g) - \frac{1}{p}\sum_{h=1}^{p} c_p(h,j) + \frac{1}{p^2}\sum_{h,g=1}^{p} c_p(h,g) \,.$$

Proofs are only sketched due to the length of the required computations: a few details are provided only for the case of the expectation value as an example of the following proofs.

The following analysis is to be considered as a combinatorial problem: as in [2, 5], the truly random permutations case is assumed. All permutations are supposed to be chosen independently and uniformly in S_p and the same applies to all the permutation pairs.

Results

R1. The expected value of the Canberra distance is

$$\begin{aligned} E\{\mathrm{Ca}(S_p)\} &= \frac{1}{|S_p|}\sum_{\sigma\in S_p} \mathrm{Ca}_I(\sigma) = \frac{1}{p!}(p-1)!\sum_{i,j=1}^{p} c_p(i,j) = \sum_{i=1}^{p} f_p(i) \\ &= \sum_{i=1}^{p} \frac{2i}{p}(2H_{2i} - H_i - H_{p+i} - 1) + 1 \\ &= \frac{2}{p}\left(\sum_{i=1}^{p} 2iH_{2i} - \sum_{i=1}^{p} iH_{n+i} - \sum_{i=1}^{p} iH_i - \sum_{i=1}^{p} i\right) + \sum_{i=1}^{p} 1 \\ &= \left(2p + 2 + \frac{1}{2p}\right) H_{2p} - \left(2p + 2 + \frac{1}{4p}\right) H_p - \left(p + \frac{3}{2}\right) \,. \end{aligned} \tag{2}$$

The proof is straightforward: the key step is the identity $f_p(i) = \frac{2i}{p}(2H_{2i} - H_i - H_{p+i} - 1) + 1$ together with the $\sum_{j=1}^{p} jH_{j+l} = \frac{(p-l)(p+l+1)}{2}H_{p+l+1} + \frac{l(l+1)}{2}H_{l+1} + \frac{p(2l-p-1)}{4}$.

R2. The approximation of the expected value by Euler's formula (1) **is**

$$\begin{aligned} E\{\mathrm{Ca}(S_p)\} &= \left(2p+2+\frac{1}{2p}\right)H_{2p} - \left(2p+2+\frac{1}{4p}\right)H_p - \left(p+\frac{3}{2}\right) \\ &= \left(2p+2+\frac{1}{2p}\right)\left(\log(2p)+\gamma+\frac{1}{4p}+o(p^{-1})\right) \\ &\quad - \left(2p+2+\frac{1}{4p}\right)\left(\log(p)+\gamma+\frac{1}{2p}+o(p^{-1})\right) - \left(p+\frac{3}{2}\right) \\ &= \frac{\log(p)}{4p} + \frac{\gamma}{4p} + \log(2)\left(2p+2+\frac{1}{2p}\right) - \left(\frac{1}{2}+\frac{1}{2p}\right) - \left(p+\frac{3}{2}\right) + o(1) \\ &= \log(2)(2p+2) - \frac{1}{2} - p - \frac{3}{2} + o(1) \\ &= \log(4)(p+1) - (p+2) + o(1)\ . \end{aligned} \tag{3}$$

The $o(1)$ term indicating the difference between the exact and the approximated values is about $4 \cdot 10^{-2}$ for $p = 20$, $2 \cdot 10^{-3}$ for $p = 10^3$ and it is less than 10^{-5} for $p > 10^5$.

R3. The variance can be computed starting from the identity

$$V\{\mathrm{Ca}(S_p)\} = \frac{1}{p-1}\sum_{i,j=1}^{p} d_p^2(i,j) = \frac{1}{p-1}\sum_{i,j=1}^{p} c_p^2(i,j) + E^2\{\mathrm{Ca}(S_p)\} - 2c_p(i,j)f_p(j)\ .$$

The expanded form involves tens of terms in H_p, H_p^2, H_{2p} and H_{2p}^2. In particular, for the two occurring sums $\sum_{i=1}^{p} i^2 H_{p+i}H_{2i}$ and $\sum_{i=1}^{p} i^2 H_{p+i}H_i$ no closed form is known.

R4. By integral approximation, the variance can be asymptotically estimated as

$$V\{\mathrm{Ca}(S_p)\} = \left(\frac{22}{3} + \frac{7}{3}\log^2(4) - \frac{4\pi^2}{9} - \frac{16}{3}\log(4)\right)p + o(p) \simeq 0.0375p + o(p)\ .$$

R5. Canberra distance is asymptotically normal. Since $d_p^2(i,j)$ reaches its maximum in $(1,1)$ and $d_p^2(1,1) = \frac{1}{p}\left[E\{\mathrm{Ca}(S_p)\} - 2(p+2-2H_{p+1})\right]$, the limit

$$\lim_p \frac{d_p^2(1,1)}{\frac{p-1}{p}V\{\mathrm{Ca}(S_p)\}} = 0$$

holds: then asymptotic normality follows from [3, Th. 1-4].

R6. Canberra distance has a maximal permutation. The two solutions $\rho_M,\ \rho_M^{-1}$ to the equation $\rho = \arg\max_{S_p}(\mathrm{Ca}_I)$ depend on the parity of p ($\rho_M = \rho_M^{-1}$ for even p):

$$\rho_M = \begin{pmatrix} 1 & 2 & \dots & \frac{p}{2} & \frac{p}{2}+1 & \frac{p}{2}+2 & \dots & p \\ \frac{p}{2}+1 & \frac{p}{2}+2 & \dots & p & 1 & 2 & \dots & \frac{p}{2} \end{pmatrix} \text{ or } \begin{pmatrix} 1 & 2 & \dots & \frac{p-1}{2} & \frac{p-1}{2}+1 & \frac{p-1}{2}+2 & \dots & p \\ \frac{p-1}{2}+1 & \frac{p-1}{2}+2 & \dots & p-1 & p & 1 & \dots & \frac{p-1}{2} \end{pmatrix}^{\pm 1}$$

respectively for even and odd p. The proof passes through restricting to smaller sets of permutations according to the following conditions:

1. if $\sigma \in S_p$ fi es an index $z \in \Omega_p = \{1,\dots,p\}$, then $\exists \tau \in S_n$ such that $\mathrm{Ca}_I(\tau) > \mathrm{Ca}_I(\sigma)$;
2. if $\exists\{i,\sigma(i)\} \subset \Omega_{\left[\frac{p}{2}\right]}$ then $\exists \tau \in S_p$ such that $\mathrm{Ca}_I(\tau) > \mathrm{Ca}_I(\sigma)$ and
   if $\exists\{i,\sigma(i)\} \subset \Omega_p \setminus \Omega_{\left[\frac{p}{2}\right]+1}$ then $\exists \tau \in S_p$ such that $\mathrm{Ca}_I(\tau) > \mathrm{Ca}_I(\sigma)$;
3. if $i \leq \left[\frac{p}{2}\right] - 1$ and $\sigma(i) > \sigma(i+1)$ then $\exists \tau \in S_p$ such that $\mathrm{Ca}_I(\tau) > \mathrm{Ca}_I(\sigma)$.

The maximum value is then $\mathrm{Ca}_I(\rho_M) =$

$$\begin{cases} 2r(H_{3r} - H_r) & \text{if } p = 4r \\ (2r+1)H_{6r} + rH_{3r+1} - (r+\frac{1}{2})H_{3r} - (2r+1)H_{2r+1} + \frac{1}{2}H_r & \text{if } p = 4r+1 \\ (2r+1)(2H_{6r+3} - H_{3r+1} - 2H_{2r+1} + H_r) & \text{if } p = 4r+2 \\ (2r+1)H_{6r+5} + \frac{1}{2}H_{3r+2} - (2r+1)H_{2r+1} - (r+1)H_{r+1} + (r+\frac{1}{2})H_r & \text{if } p = 4r+3\ , \end{cases}$$

which can be approximated by Euler's formula for any p by $\max_{S_p}(\mathrm{Ca}_I) = \frac{\log(3)}{2}p - \frac{2}{3} + o(1)$. The $o(1)$ term is $5.9 \cdot 10^{-4}$ for $p = 10^3$, $5.9 \cdot 10^{-5}$ for $p = 10^4$ and $6.0 \cdot 10^{-6}$ for $p = 10^5$.

Extensions to partial lists The Canberra distance can be naturally extended to partial lists in a few natural ways. If comparison among top-k lists is investigated for k fi ed, the Hausdorff version of the Canberra metric can be define [2]; the results in [9] show its equivalence with the Canberra distance with location parameter $l = k + 1$: $\mathrm{Ca}^{(k+1)}(\tau, \sigma) = \sum_{i=1}^{p} \frac{|\min\{\tau(i),k+1\} - \min\{\sigma(i),k+1\}|}{\min\{\tau(i),k+1\} + \min\{\sigma(i),k+1\}}$. The expected value of the Canberra distance with location parameter is

$$\begin{aligned} E\{\mathrm{Ca}^{(k+1)}(S_p)\} &= \frac{k}{p}\left(\left(2k+2+\frac{1}{2k}\right)H_{2k} - \left(2k+2+\frac{1}{4k}\right)H_k - \left(k+\frac{3}{2}\right)\right) \\ &\quad + \frac{2(p-k)}{p}(2(k+1)(H_{2k+1} - H_{k+1}) - k) \\ &= \frac{(k+1)(2p-k)}{p}\log(4) - \frac{2kp+3p-k-k^2}{p} + o(1)\ . \end{aligned}$$

Details of the top-k extension are described in [6], with application for profil ng in synthetic and oncological datasets and enrichment techniques.

If ranked lists of different lenghts $\sigma \in S_{l_1}$, $\tau \in S_{l_2}$ are to be compared by a distance d, the possible approach is to consider quotient groups of a larger group S_p, for $l_1, l_2 \leq p$: for f a suitable function, $d(\sigma, \tau) = f\left(\{d(\alpha, \beta) \colon \alpha \in S_{\tau_1}, \beta \in S_{\tau_2}\}\right)$, for $S_\xi = \{\rho \in S_p | \rho_{|\mathrm{supp}(\xi)} = \xi\}$. In [10] the case of d the Canberra distance and f the mean function is considered.

3 Applications on molecular signatures

The role of the Canberra distance as a stability indicator for ranked lists of molecular biomarkers was fi st described in [6], *e.g.*, as a measure of disarray among ranked lists produced by different classifie s or laboratories. On large scale projects such as the US-FDA led initiative MAQC-II [11], the stability indicator was applied coupled with accuracy metrics to evaluate potential sources of variability for microarray analysis, as the impact of batch preprocessing or normalization methods. Stability analysis was also applied in other ranking problems such as the analysis of gene enriched lists and the comparison of filte ing methods [6, 10]. In the f rst application in this section we present how to use the Canberra distance for model selection. In a second data intensive application we show how to quantify similarity among sets of ranked lists of differentially expressed probes under several conditions, available from the Broad Institute CMAP[2] repository of signatures.

List distances for model selection In a molecular profilin task, the degree of difference among the ranked lists produced during the feature selection process is as relevant as the classifie performance when selecting a model. We propose to couple the mean of the mutual Canberra distances among feature lists (as the disarray measure) with the classifie Area Under the ROC Curve (AUC) for model selection. As an example, we can choose a model with the optimal number of features in a binary classificatio task. The task aims at identifying positive TMPRSS2-ERG gene fusion cases from negative ones given two different cohorts of prostate cancer patients (Swedish WW, 103 positive and 352 negative and US PHS Cohort, 41 + 60). The Setlur dataset[3] consists of 6144 gene expression values from a custom Illumina DASL Assay originally described in [12]. We apply 10 times a 5-fold cross validation on the two cohorts separately, using Spectral Regression Discriminant Analysis [13] (SRDA) with $\alpha = 10^3$ as the classifie and the feature weighting algorithm, and Entropy based Recursive Feature Elimination (ERFE) as the ranking procedure [14]. In the diagnostic plot of Fig.1 we show the points corresponding to the different feature subsets for each model on an AUC versus normalized Canberra Distance space. The plot can be used as an effective tool for model selection purposes: although models have similar AUC, they show widely different levels of similarity of the corresponding ranked lists. For instance, on both cohorts, the model with 15 features has a better list similarity measure than the models with 50 and 100 features which reach the best AUC, without paying too much in terms of performance. From this perspective, the best possible compromise between similarity and performance is represented by the model with 25 features. Finally, note that the top-5 lists have small Canberra distance, but the selected features show relatively worse predictivity. On the other hand, due to the large number of irrelevant features

[2] Connectivity Map 02: `http://www.broadinstitute.org/cmap/`

[3] Available on GEO `http://www.ncbi.nih.gov/geo`, accession number GSE8402.

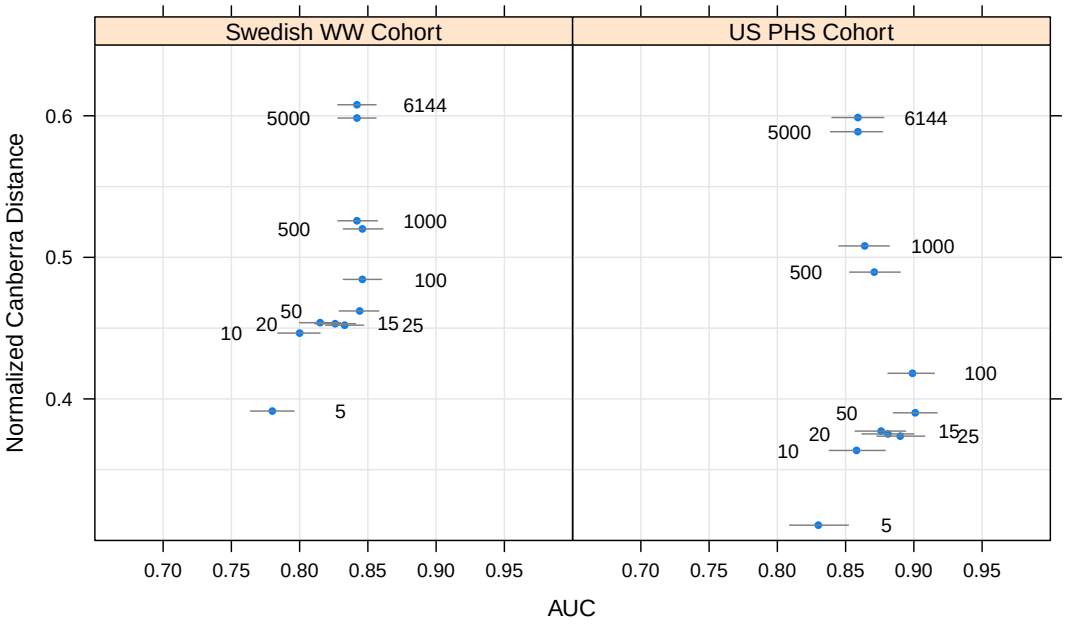


Figure 1: Accuracy-Similarity plots for the Setlur prostate cancer dataset, separately for the Swedish and US cohorts. Each point corresponds to a SRDA/ERFE model with different number of features, reported aside. Coordinates of each point are given by the AUC (bars: 95% bootstrap confidenc intervals) and the normalized Canberra distance, averaged on the $10 \times 5-$CV resampling.

(which are consistently low ranked with almost random weights), the models with more variables are characterized by an higher level of dissimilarity, which is however not heavily reflectin on the corresponding AUC values.

Similarity among CMAP signatures We consider the CMAP project that collects 6,100 lists of 22,283 gene expression probes from the HG-U133A platform and its variants [15]. The probes in the CMAP lists are decreasingly ranked according to their differential expressions between a treated and an untreated cell line by a specifi compound at different concentrations. In total, 1,309 compounds are tested at concentrations from 10^{-2} M down to 10^{-8} M on f ve different human tumoral cell lines in different batches. We computed the 18,601,950 values of the Canberra distances between all distinct pairs of lists by using the `mlpy` library on a high performance facility. Given the four compounds Haloperidol, LY-294002, Tanespimycin and Trichostatin A, the corresponding subset of z lists is extracted. Given the subset, potential sources of variability are different concentration, batch, platform and cell line. The distribution of the $z(z-1)/2$ Canberra distances normalized by the expected value (2) $E\{\mathrm{Ca}(S_{22283})\}$ is plotted in Fig.2(a). Distributions were computed using the R package `stats`. The mean distance is also explicitly drawn in the plots: a smaller mean (bottom panels) indicates an higher similarity among lists produced by the corresponding compounds, while mean values close to one (Haloperidol) indicate that the set of lists is not far from being randomly extracted within S_{22283}. The four compounds exhibit different histogram shapes and means, suggesting differences in variability. The Trichostatin A lists are the most uniform through the different experimental conditions. Trichostatin A and Tanespimycin have mean close to 0.8, while most of the distances of the other two compounds lie above this threshold. Moreover, the similarity among the histograms of Fig. 2(b) for the three cell lines HL60, MCF7 and PC3, accounting for the 99.5% of the data, shows that, on the CMAP data, cell line does not impact on variability.

Acknowledgments

The authors acknowledge funding by the European Union FP7 Project HiperDART. They also thank Andrea Gobbi for his help with the mathematical analysis, Davide Albanese for his precious work in developing and maintaining `mlpy`, and Silvano Paoli for his support with the HPC facility.

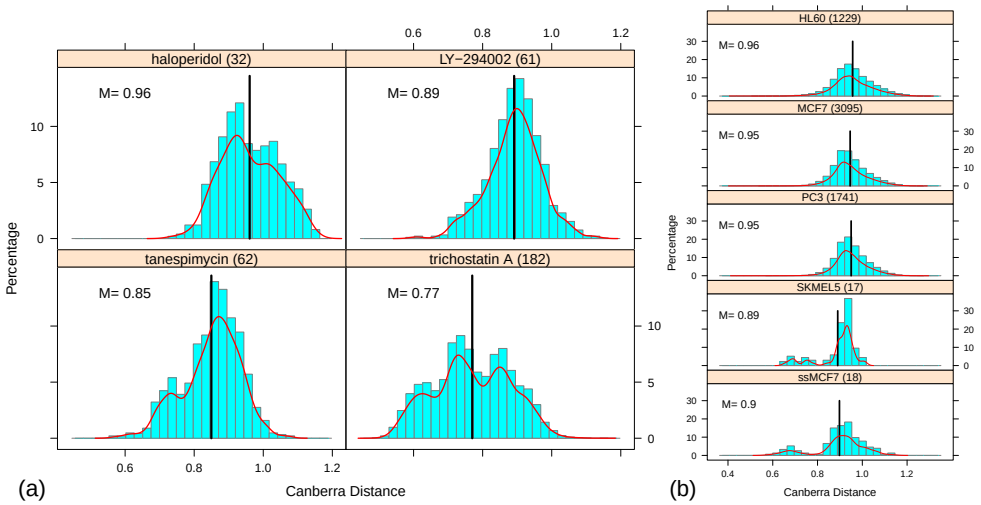


Figure 2: (a) Histogram and distribution density of the normalized Canberra distance among CMAP lists for the compounds Haloperidol, LY-294002, Tanespimycin and Trichostatin A (in parentheses: number of lists for the compound). The black vertical bar is the mean of all values for a compound. (b) Same for the f ve cell lines HL60, MCF7, PC3, SKMEL5 and ssMCF7.

References

[1] G.N. Lance and W.T. Williams. Mixed-Data Classificato y Programs I - Agglomerative Systems. *Aust. Comput. J.*, 1(1):15–20, 1967.

[2] D.E. Critchlow. *Metric methods for analyzing partially ranked data*. LNS 34. Springer, 1985.

[3] W. Hoeffding. A Combinatorial Central Limit Theorem. *Ann. Math. Stat.*, 22(4):558–566, 1951.

[4] P. Diaconis. *Group representations in probability and statistics*. Institute of Mathematical Statistics Lecture Notes – Monograph Series 11. IMS, 1988.

[5] P. Diaconis and R.L. Graham. Spearman's Footrule as a Measure of Disarray. *J. R. Stat. Soc. B*, 39:262–268, 1977.

[6] G. Jurman, S. Merler, A. Barla, S. Paoli, A. Galea, and C. Furlanello. Algebraic stability indicators for ranked lists in molecular profiling *Bioinformatics*, 24(2):258–264, 2008.

[7] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison Wesley, Reading, Mass., 1989.

[8] J. Spieß. Some Identities Involving Harmonic Numbers. *Math. Comput.*, 55(192):839–863, 1990.

[9] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top-k lists. *SIAM J. Disc. Math.*, 17:134–160, 2003.

[10] G. Jurman, S. Riccadonna, R. Visintainer, and C. Furlanello. Algebraic Comparison of Partial Lists. Submitted, 2009.

[11] The MicroArray Quality Control (MAQC) Consortium. The MAQC-II Project: A comprehensive study of common practices for the development and validation of microarray-based predictive models. Submitted, 2009.

[12] S.R. Setlur, K.D. Mertz, Y. Hoshida, F. Demichelis, M. Lupien, S. Perner, A. Sboner, Y. Pawitan, O. Andrn, L.A. Johnson, J. Tang, H.O. Adami, S. Calza, A.M. Chinnaiyan, D. Rhodes, S. Tomlins, K. Fall, L.A. Mucci, P.W. Kantoff, M.J. Stampfer, S.O. Andersson, E. Varenhorst, J.E. Johansson, M. Brown, T.R. Golub, and M.A. Rubin. Estrogen-dependent signaling in a molecularly distinct subclass of aggressive prostate cancer. *J. Natl. Cancer Inst.*, 100(11):815–825, 2008.

[13] D. Cai, H. Xiaofei, and J. Han. Srda: An efficien algorithm for large-scale discriminant analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):1–12, 2008.

[14] C. Furlanello, M. Serafini S. Merler, and G. Jurman. Entropy-Based Gene Ranking without Selection Bias for the Predictive Classificatio of Microarray Data. *BMC Bioinformatics*, 4(1):54, 2003.

[15] J. Lamb, E.D. Crawford, D. Peck, J.W. Modell, I.C. Blat, M.J. Wrobel, J. Lerner, J.-P. Brunet, A. Subramanian, K.N. Ross, M. Reich, H. Hieronymus, G. Wei, S.A. Armstrong, S.J. Haggarty, P.A. Clemons, R. Wei, S.A. Carr, E.S. Lander, and T.R. Golub. The Connectivity Map: Using Gene-Expression Signatures to Connect Small Molecules, Genes, and Disease. *Science*, 313(5795):1929–1935, 2006.

A Decision-theoretic Model of Rank Aggregation

Tyler Lu and **Craig Boutilier**
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
{tl, cebly}@cs.toronto.edu

Abstract

Modern social choice theory has spurred considerable recent work in computational *rank aggregation*, the problem of aggregating rankings into a consensus ranking. However, in the social choice context where rankings represent preferences of agents, or "voters," over outcomes, or "candidates," there is scant justification in the literature for producing rankings as an *output* of the aggregation process. We introduce a novel model which offers one possible decision-theoretic motivation for constructing a consensus ranking rather than an outcome. Roughly speaking, we view rankings as a compact, easily communicable decision policy in the face of uncertainty about candidate availability. This gives rise to a principled approach, based on minimizing expected dissatisfaction with the candidate choice, for optimization functions on rankings. We provide exact and approximation algorithms for computing optimal rankings in our model that exploit the properties of our objective function. We also describe interesting connections to popular voting protocols such as the plurality rule and the Kemeny consensus.

1 Introduction

Social choice theory is concerned with the problem of aggregating preferences of individual agents over some decision or outcome space to determine a suitable consensus outcome. Arrow [1] famously considered the problem of aggregating rankings of outcomes (or candidates) by agents (or voters) not into a choice, but into a consensus ranking. Since this time, the problem of rank aggregation has attracted considerable attention in social choice. Allowing voters to express *rankings* rather than single votes has clear value: the relative preference of a voter for any candidate—not just it's top-ranked—can sensibly play a role in determining the winner, and does in common rules like Borda. By contrast, much work in social choice assumes that the output of an aggregation procedure should itself be a ranking of all candidates. A variety of models take just this approach, e.g., the Kemeny consensus, which produces a ranking that minimizes the sum of pairwise candidate "disagreements" between the votes (input rankings) and the result. However, within the context of social choice problems such as voting the need to produce a consensus *ranking* rather than a consensus outcome is often left unspecified and unjustified.

Consider the popular Kemeny consensus: if the goal is to produce a single winner, why should one produce a ranking? And if this ranking is used to produce a winner, why should pairwise disagreements across the entire ranking of each voter be minimized? Of course, there are a variety of rank aggregation settings where the decision space explicitly requires a ranking. For example, if each "voter" is expressing a noisy assessment of some underlying objective ranking (e.g. quality of sports teams), under certain assumptions, the Kemeny consensus provides a maximum likelihood estimate of the underlying ranking [10]. In web search, one might want to rank results to minimize average effort to find the relevant results [2, 4, 5]; but it is unclear why social choice/voting models are appropriate. Instead, a model that explicitly models search costs and probability of relevance would be more appropriate [9].

More generally, we argue that the decision criterion for which aggregation is being implemented should directly influence the process by which one aggregates rankings. To this end, we propose a new model that justifies the use of rankings in preference aggregation. Our model supposes that any candidate may be rendered unavailable with a certain probability, and the output ranking determines a decision (or "winner") by selecting the best available candidate. In our model, the optimal aggregate ranking minimizes the expected voter "dissatisfaction" with the selected candidate. To illustrate, suppose an organization considers a number of candidates for an open position. Members of the hiring committee submit their preference orderings; however, there is a chance any candidate may take another job, so the committee must be prepared to select a candidate from *any* subset of available candidates. A ranking—where the top-ranked available candidate is chosen—provides us with a compact, easily interpretable policy for selection.

In this short abstract, we briefly develop this model formally and sketch some theoretical and computational results, including the relation of this new aggregation model with the Kemeny consensus. We also discuss some directions for future research, including how this model can be used to support personalization in a learning setting, and suggesting alternative, decision-theoretically motivated approaches to rank aggregation.

Considerable work on computational social choice has focused on the Kemeny aggregation rule [7]. It has been shown to be NP-hard to compute [6, 4], but good approximation heuristics have been given in the context of web meta-search [4] as well as a polynomial time approximation scheme (PTAS) [8]. Practical approaches for exact computation have also been explored [3]. While the Kemeny rule is quite popular and natural, it does not address why one needs an aggregate ranking. Our model gives intuitions as to why a Kemeny consensus can be useful from a decision-making perspective.

Rank or preference aggregation also has some interesting connections to the literature on rank learning. For example, [2] focuses on learning a preference function over all pairs of items while only given preference comparisons between some of the items. A final ranking of all items that is most consistent with the learned preference function is computed. This can be seen as preference aggregation where the preference comparisons come from different users and the learned ranking is an aggregation of these preferences. In settings where we have little information on each user's preferences over items, as in recommender systems, the preferences of similar users can be aggregated under the assumption that similar users have similar preference functions. Thus we can leverage the more abundant aggregated preferences to facilitate better learning. This is roughly the idea behind, for example, the "label ranking paradigm" of [5] where one is interested in building personalized ranking for each user while only having limited preference data.

2 Model and Technical Results

We have voters $N = \{1, 2, \ldots, n\}$ with corresponding total preference orderings $V = \{v_1, ..., v_n\}$ (i.e. *votes* or *rankings*) over the set of candidates $C = \{c_1, \ldots, c_m\}$. Denote by $v_\ell(c_i)$ the rank of c_i in vote v_ℓ so that voter ℓ prefers c_i over c_j if $v_\ell(c_i) < v_\ell(c_j)$. From a decision-theoretic perspective, we are interested in aggregating voters' orderings to select one candidate (i.e. decision). However, in some real life scenarios we cannot expect any particular candidate to be available. Some decisions turn out to be unavailable and backups must be planned in advance. One solution to this problem is to provide a complete ranking of all the candidates, and upon knowing which candidates are available, the top available candidate of the ranking is selected. Thus, ranking here is useful in the sense that it acts as a policy of what to do under uncertain outcomes. The next question is how do we rank the candidates so as to maximize the voters' "happiness?" But first we give a formalization of the above discussion.

Definition 1. A *decision policy* is any function $W : 2^C \to C \cup \{\emptyset\}$. That is, given available candidates and a set of votes, W outputs a candidate to recommend. We further restrict that $W(\emptyset) = \emptyset$, that is, the decision policy does not make a recommendation when no candidates are available.

Let $\mathcal{W}$ be a class of decision policies and P a probability distribution over 2^C giving the probability that exactly a certain subset $S \subseteq C$ is available. For a ranking r and available candidates $S \subseteq C$, let $\mathrm{top}(r, S)$ be the highest ranked candidate in r that also occurs in S. Our goal, given input V and P,

is to output a decision policy

$$W^* = \operatorname*{argmin}_{W \in \mathcal{W}} \mathop{\mathbf{E}}_{S \sim P} \left[\sum_{\ell=1}^{n} \mathbf{1}[W(S) \neq \mathrm{top}(v_\ell, S)] \right], \tag{1}$$

where $\mathbf{1}$ is the indicator function. We can view this as expected dissatisfaction, where a voter is dissatisfied iff its top-ranked available candidate is not chosen. If W^* is a modification of the plurality voting rule—selecting the candidate with most top votes after removing unavailable candidates from consideration—it is the optimal policy. However, this function does not give an explicit sense of which candidates are likely to be chosen over others, and does not admit a compact representation (in general, requiring exponential description length, i.e. not the algorithm, but describing the mapping from available candidates to recommendations).

In this work we study the class of "ranking policies" $\mathcal{W} = \{\mathrm{top}(r, \cdot) : r \text{ a ranking}\}$. This is the set of decision policies that outputs the top available candidate of a ranking r. We focus on a simple probability model P of available candidates $S \subseteq C$ being defined by $P(S) = p^{m-|S|}(1-p)^{|S|}$. That is, candidates are unavailable with probability p, independently of other candidates. When $p = 1$, the optimization problem is trivial. For any two rankings r, v let $t(c, r, v) = |\{c' : r(c') < r(c)\} \cap \{c' : v(c') < v(c)\}|$ be the number of candidates ranked above c in both r and v. With ranking policies and the above distribution P, it can be shown that the objective function in Equation (1) is

$$\mathcal{M}_p(r, V) := \sum_{\ell=1}^{n} \sum_{i=1}^{m} (1-p) p^{r(c_i)-1} \left(1 - p^{v_\ell(c_i) - t(c_i, r, v_\ell) - 1}\right) .$$

We denote $\mathcal{M}_p$ the expected number of mistakes with p representing the probability a candidate is unavailable. Suppose $r = r_1 r_2 \cdots r_m$, let $\mathcal{M}_p^{(i,j)}(r, V) := \sum_{\ell=1}^{n} \sum_{u=i}^{j} (1-p) p^{u-1} \left(1 - p^{v_\ell(r_u) - t(r_u, r, v_\ell) - 1}\right)$. We conjecture that optimizing $\mathcal{M}_p(r, V)$ is NP-hard for values of p that are not too small. In fact, we will see later that, for values of p close to 1, an optimal ranking is also Kemeny optimal, implying NP-hardness. One nice property of the objective function is that, roughly, it penalizes exponentially less for misorderings near the bottom of the ranking.

Algorithmically, one can compute the optimal ranking using an integer programming formulation. We have also developed a very natural greedy heuristic to approximate the optimal solution. The idea is that we first find the top-ranked candidate r_1 by greedily choosing the candidate with the least expected number of mistakes if it were placed in the top position (assuming that if it is unavailable, no decision is made). For the second position, we consider the remaining candidates and compute an updated expected number of mistakes that takes into account the fact that r_1 is in the first position. This is repeated for all m positions. The heuristic can be improved as follows: we detect if there's a *dominant candidate* or Condorcet winner at each stage: a candidate having more than half of all top votes (w.r.t. the remaining candidates) and choose it first (it can be shown this is optimal). Although we have only very loose bounds on the approximation ratio, our experiments suggest that this heuristic often finds the optimal solution, and is very close to optimal otherwise. We also have a PTAS guaranteeing arbitrarily good approximations, which can be implemented by modifying our integer programming formulation of exact computation. The basic idea is to find the myopically top k candidates, then arbitrarily order the remaining candidates. This offers a good approximation since the objective function penalizes "mistakes" much less near the bottom.

Theorem 1. *The algorithm that myopically selects the top k candidates, i.e. optimizes $\mathcal{M}_p^{(1,k)}(r, V)$, (and also considers dominant candidates) has an approximation ratio of $2p^k/(1-p)^2$.*

There are several interesting connections between our model, Kemeny aggregation, and plurality voting. First it can be seen that when $p = 0$ the top candidate in the optimal ranking is also the one plurality voting would elect. This makes sense as our model would only focus the mistakes at the top of rankings. When p is very close to 1, our model tells us that the candidates ranked at the bottom matters a lot too. Thus it becomes important to get nearly all of the ranking correct and not just the top candidates. In fact, we can show that our model really becomes a Kemeny aggregation problem (making the optimization NP-hard).

Theorem 2. *Let $\epsilon = 2/(nm(m-1)+2)$ and $p > (1-\epsilon)^{\frac{1}{m-1}}$ then the following holds (1) for any set of candidates C, votes V, any minimizer r^* of $\mathcal{M}_p(\cdot, V)$ is also a Kemeny optimal ranking, (2) there exists candidates C', votes V', and a Kemeny optimal ranking K^* which is* not *a minimizer of*

$\mathcal{M}_p(\cdot, V')$. *In fact this is true for all* $p \in (0,1)$*, and (3) for any set of candidates* C*, votes* V*, any Kemeny optimal ranking* K^* *we have* $\mathcal{M}_p(K^*, V)/\min_r \mathcal{M}_p(r, V) \leq 1/(1-\epsilon)$.

A related question is how well the Kemeny optimal ranking compares in our mistake model. It turns out when p is small it is twice as bad.

Theorem 3. *For any set of votes* V*, let* K^* *be the Kemeny optimal ranking, then* $\mathcal{M}_p(K^*, V)/\min_r \mathcal{M}_p(r, V) \leq 2/(1-p)^2$*. Furthermore, for small values of* p *there exists set of votes such that this ratio matches the upper bound very closely.*

When p is larger the bound above is loose (e.g. if p is close to 1 we know Kemeny is a very good approximation of our optimization model). The basic idea is that Kemeny will always choose a candidate at the top if it has more than half of all top votes (coinciding with optimal ranking for small p) then the remaining ordering is not as important.

3 Conclusion and Future Work

We have introduced a novel model that provides one possible rationale for computing a consensus ranking of voter preferences rather than a decision. Our model induces a principled objective function for rank aggregation, that differs from classical rank aggregation rules, but bears some strong connection to methods such as plurality and Kemeny voting rules. We have provided exact algorithms as well as approximation algorithms that can exploit the fact that items higher in the ranking are more important.

Future work includes looking at other interesting and realistic distributions over available candidates, as well as dealing with aggregating partial preferences from voters. We are also exploring other decision models that would naturally induce rankings, such as web search and or other consensus recommendations that provide a range of options for a variety of users. One of our primary aims is to incorporate such decision models into techniques for rank learning where limited preference data from users must be aggregated to facilitate learning. In this setting, the tradeoff between making fully personalized decisions (with limited data) and pure consensus decisions (with increased degree of dissatisfaction) gives rise to natural criteria for clustering/aggregating certain subsets of user and not others. From a more technical perspective the complexity of our optimization problem remains open, as does proving tighter approximation bounds for the greedy algorithm.

References

[1] Kenneth J. Arrow. *Social Choice and Individual Values, Second edition (Cowles Foundation Monographs Series)*. Yale University Press, September 1970.

[2] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1998.

[3] Vincent Conitzer, Andrew J. Davenport, and Jayant Kalagnanam. Improved bounds for computing kemeny rankings. In *AAAI*, 2006.

[4] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *World Wide Web*, pages 613–622, New York, NY, USA, 2001. ACM.

[5] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16-17):1897 – 1916, 2008.

[6] John Bartholdi III, Craig Tovey, and Michael Trick. Voting schemes for which it can be difficult to tell who won the election. In *Social Choice and Welfare*, volume 6, pages 157–165, 1989.

[7] John G. Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

[8] Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *ACM STOC*, pages 95–103, New York, NY, USA, 2007. ACM.

[9] Filip Radlinski and Thorsten Joachims. Active exploration for learning rankings from clickthrough data. In *KDD*, pages 570–579, 2007.

[10] Peyton Young. Optimal voting rules. In *Journal of Economic Perspectives*, volume 9, pages 51–64, 1995.

Globally Optimal Partial Ranking for A Category of Data Mining Problems

Junshui Ma*, and Vladimir Svetnik

Merck Research Laboratories, Merck & Co, Inc.
126 E. Lincoln Ave., RY33-300
Rahway, NJ 07065
junshui_ma@merck.com and *vladimir_svetnik@merck.com*

Abstract

This paper first introduces a new category of data mining problems, which focus on discovering information embedded in the feature-feature interactions of the underlying dataset. The solution to this category of problems is then formulated as finding the top-m partial ranking of all possible feature subsets. Based on a well-designed intuitive evaluation function, a computationally efficient procedure is proposed to obtain the top-m and bottom-m partial ranking of all possible feature subsets, and the result is guaranteed to be globally optimal.

1 Introduction

Some data mining tasks treat rankings as the subjects to work on, i.e. visualization [1], comparison [2], and aggregation [3], while some other data mining tasks find their solutions by ranking the underlying dataset [4]. This study focuses on the latter.

A surprisingly large number of real-world problems converge to one category of data mining tasks. Here are some examples in this category: (1) finding the set of expressed genes that appear more frequently among people with a disease than those without; (2) finding the set of words that are used more frequently in one category of articles than the others; (3) finding the chemical compound substructures that appear more frequently in compounds with a known property than those without [5]; (4) finding the set of drugs that patients take simultaneously and then frequently report a particular adverse event (AE) than other AEs [6]. The shared goal of these tasks is that, instead of studying features individually, they focus on the information embedded in feature groups, feature subsets. That is, they emphasize the feature-feature interactions.

In this paper, the solution to this category of data mining problems is first formulated as finding a top-m partial ranking. Then, a computationally efficient procedure to obtain the partial ranking is proposed.

2 Data mining solutions formulated as partial ranking

Assume the problem dataset, *Tset*, has N instances, and M features involved in all the instances. For example, in the second example regarding article categorization, N is the number of articles analyzed, and M is the size of the vocabulary used in all the articles; in the third example regarding chemical compound substructures, N is the totally number of chemical compounds studied, M is the number of all possible substructures that appear among the compounds. The dataset, *Tset*, is composed of two parts:

(i) a label vector, $\mathbf{L} = [a_1 \quad a_2 \quad \cdots \quad a_N]^T$, where $a_i \in \{0,1\}$ labels the i^{th} instance with 1 for the active class, and 0 for the inactive class, and

(ii) a feature matrix, $\mathbf{D} = \left[\mathbf{C}_1^T \quad \mathbf{C}_2^T \quad \cdots \quad \mathbf{C}_N^T\right]^T$, where each row, $\mathbf{C}_i = [e_{i1} \quad e_{i2} \quad \cdots \quad e_{iM}]$, represents an instances using the M feature variables, $e_{ij} \in \{0,1\}, j = 1 \cdots M$. $e_{ij} = 1$ indicates that the j^{th} feature exists in the i^{th} instance, while $e_{ij} = 0$ indicates non-existence. Assuming $\mathbf{D}$ a binary matrix does not dramatically damage its generality, because continuous or g-level categorical feature variables can always be converted to binary by either thresholding, or by expanding the column of a g-level variable to g columns of binary variables.

Generally speaking, the feature matrix, $\mathbf{D}$, is very sparse. An instance $\mathbf{C}_i$ can thus also be efficiently represented as the indices of the features with non-zero feature variables. That is, $\mathbf{C}_i = \{m_1^{(i)} \quad m_2^{(i)} \quad \cdots \quad m_{n_i}^{(i)}\}$, when the m_k th feature exists in $\mathbf{C}_i$, and $\mathbf{C}_i$ has totally n_i features. Similarly, a subset of features can be denoted as $s = \{m_1, m_2 \cdots, m_{|s|}\}$, $|s|$ represents the number of features in s. All of the subsets of the M features formed a subset space, Ω. A subset evaluation function, $v(s): \Omega \rightarrow R$, can be introduced to align with the goal of the data mining task. A top-m partial ranking of all of the subsets can be obtained according to the values of $v(s)$. The top m subsets then form a pool of candidate solutions to the data mining tasks.

Given the evaluation function $v(s)$, ranking can be obtained by sorting the values from $v(\Omega)$. However, the subset space Ω is usually a huge discrete space. For example, a fairly modest real-world chemical compound dataset is composed of 73 basic structure features [7], i.e. M=73. Ranking its subset space Ω requires evaluating $9.44 \text{x} 10^{21}$ subsets, and then sorting them. Even if we reduce our attempt from a full ranking to a top-m partial ranking, it only reduces the computation from sorting to searching, and the task can still take a normal PC over one million years to finish. Although some stochastic search methods like random forest [8] and the combination of decision trees and simulated annealing [9] can be effective, they do not guarantee the selected subsets are top ranked in a global sense. Therefore, a computationally efficient solution to this partial ranking problem becomes essential.

3. A computationally efficient partial ranking procedure

First, we introduce a subset evaluation function, $v(s)$. We propose an evaluation function as, between the active and inactive classes, the difference in probability that an instance has the feature subset s. That is,

$$v(s) = \Pr(c \in C(s) \mid c \in C_A) - \Pr(c \in C(s) \mid c \in C_N) \tag{1}$$

where c denotes an instance; s denotes the investigated feature subset; C_A is the set of instances in the active class; C_N is the set of instances in the inactive class; and $C(s)$ is the set of instances (active or inactive) that have the subset s.

Equation (1) can be derived as

$$v(s) = \Pr(c \in C(s))\{\frac{\Pr(c \in C_A \mid c \in C(s))}{\Pr(c \in C_A)} - \frac{\Pr(c \in C_N \mid c \in C(s))}{1 - \Pr(c \in C_A)}\}. \tag{2}$$

This reveals that the evaluation function $v(s)$ has a flavor of a posteriori probability. For any given dataset, $\Pr(c \in C_A)$ is fixed. Therefore, the second part of (2) in the brace can be reduced to something similar to the difference in likelihood probability, while the first part, i.e. $\Pr(c \in C(s))$, is something like a prior probability. For any given dataset, maximizing the second part alone is equivalent to maximizing $\Pr(c \in [C_A \cap C(s)]) / \Pr(c \in C(s))$, which favors the subset s that has smaller $\Pr(c \in C(s))$ when $\Pr(c \in [C_A \cap C(s)])$ is the same. Thus, without the first part of $\Pr(c \in C(s))$, the evaluation function $v(s)$ would unjustifiably give those rare feature subsets, i.e.

the subsets supported by very small number of instances, higher values. Fortunately, the first part of (2), $\Pr(c \in C(s))$, counters this effect, and forces the evaluation function to bias to the subsets supported by a large number of instances. It is generally favorable if a large percentage of data support a data mining discovery. In other words, $\Pr(c \in C(s))$ provides the regularization effect to our evaluation function $\nu(s)$ in a similar way that the prior probability does to a posteriori probably.

The basic idea behind the proposed fast ranking algorithm is to first shrink the subset space Ω to a manageable set of candidate subsets using any available frequent-set search algorithms [10]. Then, a full ranking is done within the candidate set. We prove that, as long as a specified condition is met, the top m subsets in the candidate set is also the top m subsets in the whole space Ω. This proof ensures that the proposed algorithm can generate a global optimal partial ranking.

The algorithmic procedure includes four steps:

1. Split the instances in *Tset* into two class sets: $C_A = \bigcup_{a_i=1} \mathbf{C}_i$, and $C_N = \bigcup_{a_i=0} \mathbf{C}_i$.
2. Within the active set C_A, find all of the candidate subsets s that meet the following condition using a fast frequent-set search algorithm, e.g. the Apriori algorithm [10].
   $S_{set} = \{s \,|\, |C(s,A)| > \alpha |C_A|\}$, (3)
   where $0 < \alpha < 1$ is called the *minimal frequency* of S_{set}, $|\bullet|$ denotes the number of elements in a set, and $C(s,A)$ represents the set of compounds within C_A that have s.
3. For each s in S_{set}, calculate $\nu(s)$ according to (1).
4. Sort the subsets in S_{set} according to their $\nu(s)$ in decreasing order to get a full ranking of the candidate set, $S_{array} = (s^{(1)} \quad s^{(2)} \quad \cdots \quad s^{(|S_{set}|)})$.

The first *m* subsets, $(s^{(1)} \quad s^{(2)} \quad \cdots \quad s^{(m)})$, in S_{array} are also the top-m ranking of the whole subset space Ω, as long as

$$\nu(s^{(m)}) \geq \alpha . \quad (4)$$

Proof: Suppose a list of subsets, $\{s^{*(1)}, s^{*(2)}, \cdots, s^{*(m)}\}$, are ranked as the top-m subsets globally, i.e. $\nu(s^{*(1)}) \geq \nu(s^{*(2)}) \geq \cdots \geq \nu(s^{*(m)}) \geq \alpha$.
According to (1), we get $\Pr(c \in C(s^{*(m)}, A) \,|\, c \in C_A) - \Pr(c \in C(s^{*(m)}, N) \,|\, c \in C_N) \geq \alpha.$
Because $\Pr(c \in C(s^{*(m)}, N) \,|\, c \in C_N) \geq 0$, we get $\Pr(c \in C(s^{*(m)}, A) \,|\, c \in C_A) \geq \alpha$, which can be estimated as $|C(s^{*(m)}, A) \cap C_A| / |C_A|$.

Since $C(s^{*(m)}, A) \subseteq C_A$, it becomes obvious that $s^{*(m)}$ should be in the candidate set, S_{set}. Thus, the top-m subsets in S_{set} should also be the top-m subsets globally. ■

The minimal frequency α is the only algorithmic parameter. Its selection depends on the nature of the dataset. From an algorithmic perspective, α is like a parameter to control the level of trustable signal-to-noise ratio. When a large α is used, the method can only capture subsets with strong signals (i.e. the subsets having much higher probability in C_A than they do in C_N). In contrast, a very small α has to be used to capture subsets having weak signals (i.e. the subsets having similar probability in C_A and C_N). From the computational perspective, α is a parameter to control the amount of computation required. A very small α will produce a large candidate set S_{set}, which increases the computation at Steps 3 and 4. According to our experience, we start with setting α as 0.15, and then adjust it according to the initial results, if necessary.

If the dataset have very small number of active instances, i.e. $|C_A|$ is very small, the algorithm may not be able to find the top m subsets with a tractable amount of computation due to the condition of $|C(s,A)|>\alpha|C_A|$ for forming the candidate set in Step 2. However, this is not a critical issue in real-world applications, since it is reasonable for a data mining method to refuse generating hypotheses regarding the active class when no enough data from that class are provided.

This algorithm can be easily adjusted to find the bottom-m partial ranking by replacing the active class to inactive class in Step 2 of the algorithmic procedure.

4. Experiment

Due to space limit, we only briefly present the results from one public chemical compound dataset, BZR [7]. This dataset has 163 chemical compounds, among which 73 belong to the active class. The structure of each compound is represented using features called *drugbits* [11]. One or several drugbits features form a compound substructure, which is considered as a candidate structure-activity-relation (SAR) rule [5]. Each compound in this dataset is represented by 73 drugbits features. In this study, we want to know what substructures, i.e. candidate SAR rules, are more frequently seen in the active class than in the inactive one.

Three methods, i.e. the Random-Forest-variable-importance method [11], the Tanalyze method [12], and the proposed method, were applied to this dataset. The first two methods can only identify substructures composed of a single drugbits, while the proposed method is inherently capable of finding substructures with all possible drugbits combinations.

Using the proposed method, the top-m ranking of all possible substructures was generated within a minute using a Pentium M 1.6GHz computer. The top five are listed in Table 1. Two of them are SAR rules with a single drugbits, and they agree with the single-feature rules discovered by existing SAR-rule discovery methods, i.e. the Random-Forest-variable-importance method and the Tanalyze method. More importantly, the remaining three are SAR rules with multiple drugbits. It is this capability of generating feature-interaction SAR rules that helps the proposed method stand out. Although some tree-based methods [8] can also find multiple-drugbits SAR rules in a heuristic way, no other methods can guarantee that the generated SAR rules are globally optimal, as the proposed method does. These multiple-drugbits, i.e. feature-interaction, rules provide a new perspective to chemists for future compound structure refinement.

Table 1: Top substructures discovered from BZR dataset

Rank	Feature Subset, s	# Cmpd. in a Set		% in a Set		$v(s)$
		Active	Inactive	Active	Inactive	
1	{***aromatic_heterocycle***}	*53*	*25*	*72.60*	*27.78*	***0.4482***
2	{***aliphatic_heterocycle***, ***aromatic_heterocycle***}	*47*	*18*	*64.38*	*20.00*	***0.4438***
3	{***aliphatic_heterocycle***, ***aromatic_Cl***, ***aromatic_heterocycle***}	*36*	*12*	*49.32*	*13.33*	***0.3598***
4	{***aromatic_Cl***, ***aromatic_heterocycle***}	*39*	*18*	*53.42*	*20.00*	***0.3342***
5	{***aromatic_ester2***}	*28*	*7*	*38.36*	*7.78*	***0.3058***

5. Conclusion and discussion

A category of data mining problems focus on discovering feature interactions embedded in the dataset. The solution to this category of tasks is formulated as obtaining the top-m partial ranking of all possible combinations of features, i.e. feature subsets. A subset evaluation function $v(s)$ is introduced, and a practical fast partial ranking algorithm is subsequently provided to guarantee that a globally optimal top-m (or bottom-m) ranking can be achieved.

Although the method is presented as a fast algorithm for top-m partial ranking, it can also be used as an optimizer in a discrete space. We want to emphasize that the proposed method indeed provides a new solution to a class of extremely difficult optimization problems, i.e. optimization in a discrete space. Exploring in this direction can potentially lead to many more new algorithms. In additions, this method can be considered as a special feature selection algorithm, which makes full use of the sparse property of the feature matrix, and is guaranteed to select the best subset of features in the sense of the evaluation function. Finally, its connection with linear discriminant analysis (LDA) is under investigation.

References

[1] A. Ukkonen, "Visualizing Sets of Partial Rankings", The 7th International Symposium On Intelligent Data Analysis, 2007, Ljuljana.

[2] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing partial rankings", SIAM Journal on Discrete Mathematics, Volume 20, Number 3, p.628-648, 2006.

[3] N. Ailon, "Aggregation of partial rankings, p-rakings and top-m lists", Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, New Orleans, Louisiana, 2007.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web", Technical Report, Stanford InfoLab, 1999.

[5] W. J. Dunn III, "Quantitative Structure-Activity Relationships," *Chemometrics and Intelligent Laboratory Systems*, 1989, 6, 181-190.

[6] U. S. Food and Drug Administration, *Adverse Event Reporting System (AERS) website homepage*, http://www.fda.gov/cder/aers/default.htm.

[7] J. J. Sutherland, L. A. O'Brien, and D. F. Weaver, "A Comparison of Methods for Modeling Quantitative Structure-Activity Relationships", Journal of Medicinal Chemistry, vol. 47, pp. 5541-5554, 2004.

[8] L. Breiman, "Random Forests", Machine Learning, 45, 5–32, 2001.

[9] P. Blower, M. Fligner, J. Verducci, and J. Bjoraker, "On Combining Recursive Partitioning and Simulated Annealing to Detect Groups of Biologically Active Compounds", Journal of Chem. Inf. Comp. Sci., vol. 42, pp. 392-404, 2002.

[10] B. Goethals, "Survey on frequent pattern mining," 2003, [Online] Available: http://citeseer.ist.psu.edu/goethals03survey.html.

[11] V. Svetnik, T. Wang, C. Tong, A. Liaw, R. P. Sheridan, and Q. Song, "Boosting: An Ensemble Learning Tool for Compound Classification and QSAR Modeling," Journal of Chemical Information and Modeling, vol. 45 (3), pp. 786 -799, 2005.

[12] R. P. Sheridan, P. Hunt, and J. Chris Culberson, "Molecular Transformations as a Way of Finding and Exploiting Consistent Local QSAR", Journal of Chemical Information and Modeling, vol. 46, pp.180-192, 2006.

Image Ranking with Eye Movements

Kitsuchart Pasupa[*]
School of Electronics & Computer Science
University of Southampton
kp2@ecs.soton.ac.uk

Sandor Szedmak[†]
School of Electronics & Computer Science
University of Southampton
ss03v@ecs.soton.ac.uk

David R. Hardoon[‡]
Data Mining Department
Institute for Infocomm Research (I^2R)
drhardoon@i2r.a-star.edu.sg

Abstract

In order to help users navigate an image search system, one could provide explicit rank information on a set of images. These rankings are learnt so to present a new set of relevant images. Although, requiring explicit information may not be feasible in some cases, we consider the setting where the user provides implicit feedback, eye movements, to assist in such a task. This paper explores the idea of implicitly incorporating eye movement features in an image ranking task. Previous work had demonstrated that combining eye movement and image features improved the retrieval accuracy. Despite promising results the proposed approach is unrealistic as no eye movements are given a-priori for new images. We propose a novel search approach which combines image together with eye movements features in a tensor Ranking Support Vector Machine, and show that by extracting the individual source-specifi weight vectors we are able to construct a new image-based semantic space which outperforms in retrieval accuracy.

1 Introduction

Recently, relevance feedback, which is explicitly provided by the user while performing a search query on the quality of the retrieved images, has shown to be able to improve on the performance of Content-Based Image Retrieval systems, as it is able to handle the large variability in semantic interpretation of images across users. Many systems rely on an explicit feedback mechanism, where the user explicitly indicates which images are relevant for their search query and which ones are not. However, providing explicit feedback is also a laborious process as it requires continues user response. Alternatively, it is possible to use implicit feedback (e.g. eye movements, mouse pointer movements) to infer relevance of images. In other words, user responses that are implicitly related to the task performed.

In this study we explore the use of eye movements as a particular source of implicit feedback to assist a user when performing such a task (i.e. image retrieval). Eye movements can be treated as an implicit relevance feedback when the user is not consciously aware of their eye movements being tracked. This work is an extended study from [4] where they demonstrated that ranking of images can be inferred from eye movements using Ranking Support Vector Machine (Ranking SVM). Their experiment shows that the performance of the search can be improved when simple images

[*] www.ecs.soton.ac.uk/∼kp2
[†] www.ecs.soton.ac.uk/∼ss03v
[‡] www.davidroihardoon.com

features namely histograms are fused with the eye movement features. Despite their encouraging results, their proposed approach is largely unrealistic as they combine image and eye features for both training and testing. Whereas in a real scenario no eye movements will be presented a-priori for new images. Therefore, we propose a novel search methodology which combines image features together with implicit feedback from users' eye movements during training, such that we are able to rank new images with only using image features. For this purpose, we propose using tensor kernels in the Ranking SVM framework. Tensors have been recently used in bioinformatics [2, 6]. In this study we use the tensor product to constructed a joined semantics space by combining eye movements and image features.

2 Ranking SVM

We are given a set of samples $\mathcal{S} = \{(\mathbf{x}_i, r_i)\}, i = 1, \ldots, M$, and a set of pairs of indices $\mathcal{P} = \{(i,j)|i < j\}$,$m = |\mathcal{P}|$. Let $\mathbf{x}_i$ denote some feature vector and r_i denote the ranking assigned to $\mathbf{x}_i$. If $r_1 \succ r_2$, it means that $\mathbf{x}_1$ is more relevance than $\mathbf{x}_2$. Consider a linear ranking function,

$$\mathbf{x}_i \succ \mathbf{x}_j \Longleftrightarrow \langle \mathbf{w}, \mathbf{x}_i \rangle - \langle \mathbf{w}, \mathbf{x}_j \rangle > 0,$$

where $\mathbf{w}$ is a weight vector and $\langle \cdot, \cdot \rangle$ denotes dot product between vectors. This can be placed in a binary SVM classificatio framework where $c_{(i,j)}$ is the new label indicating the quality of rank pair,

$$sgn(\langle \mathbf{w}, \mathbf{x}_i - \mathbf{x}_j \rangle) = \begin{cases} c_{(i,j)} = +1 & \text{if } r_i \succ r_j \\ c_{(i,j)} = -1 & \text{if } r_j \succ r_i \end{cases},$$

which can be solved by the following optimisation problem,

$$\min \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{(i,j)\in\mathcal{P}} \xi_{(i,j)} \tag{1}$$

subject to the following constrains:

$$\begin{aligned} \forall (i,j) \in \mathcal{P} \quad &: \quad c_{(i,j)}(\langle \mathbf{w}, \mathbf{x}_i - \mathbf{x}_j \rangle + b) \geq 1 - \xi_{(i,j)} \\ &: \quad \xi_{(i,j)} \geq 0 \end{aligned}$$

where C is a hyper-parameter which allows trade-off between margin size and training error, and $\xi_{(i,j)}$ is training error. Alternatively, we are represent the Ranking SVM as a vanilla SVM where we re-represent our samples as

$$\phi(\mathbf{x})_{(i,j)} = \mathbf{x}_i - \mathbf{x}_j$$

with label $c_{(i,j)}$. Finally, we quote from [3] the general dual SVM optimisation as

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{(i,j)\in\mathcal{P}} \alpha_{(i,j)} - \frac{1}{2} \sum_{(i,j),(k,l)\in\mathcal{P}} \alpha_{(i,j)}\alpha_{(k,l)}c_{(i,j)}c_{(k,l)}\kappa(\phi_{(i,j)}(x_i, x_j), \phi_{(k,l)}(x_k, x_l)) \tag{2}$$

$$\text{subject to } \textstyle\sum_{(i,j)\in\mathcal{P}} \alpha_{(i,j)}c_{(i,j)} = 0 \text{ and } \alpha_{(i,j)} \geq 0, \;\; (i,j) \in \mathcal{P},$$

where we again use $c_{(i,j)}$ to represent the label and $\kappa(\phi_{(i,j)}(x_i, x_j), \phi_{(k,l)}(x_k, x_l))$ to be the kernel function between $\phi_{(i,j)}$ and $\phi_{(k,l)}$.

3 Tensor Ranking SVM

Let us consider the learning situation when we have two sources of feature vectors, e.g. in our scenario we are given a feature vector to every image pairs $\phi(x)_{(i,j)} = \mathbf{x}_i - \mathbf{x}_j$ and we have feature vectors $\phi(y)_{(i,j)} = \mathbf{y}_i - \mathbf{y}_j$ derived from the eye movement, $\mathbf{y}_i$, to a subset of the images. The task is to compute a hyperplane to the Ranking SVM such that the normal vector $\mathbf{w}$ of this hyperplane also incorporates the information from the partially given eye movements and not only from the image features. We suppose that both sources of the information are available in the training procedure. To combine the two sources into a common feature vector the tensor product of the feature vectors $\phi(x)_{(i,j)} \circ \phi(y)_{(i,j)}$ of the sources is computed, and let the optimal normal vector of the common

space be denoted by W. Then we assume that the projection of W into the space of image features can express the effect of the eye movements. Thus, in the common space we have the following constraints

$$\forall(i,j) \in \mathcal{P} : c_{(i,j)}(\langle W, \phi(x)_{(i,j)} \circ \phi(y)_{(i,j)} \rangle + b) \geq 1 - \xi_{(i,j)}.$$

For the sake of simplicity, we change the index expression exploiting the fact that Ranking SVM is a conventional SVM with special feature vectors. The index i denotes the constraints in the next formulas.

In the following section we propose to construct a tensor kernel on the ranked image and eye movements features, i.e. following equation (2), to then to train an SVM. Therefore, let $\mathbf{X} \in \mathbb{R}^{n \times m}$ and $\mathbf{Y} \in \mathbb{R}^{\ell \times m}$ be the matrix of sample vectors, $\mathbf{x}$ and $\mathbf{y}$, for the image and eye movements respectively, where n is the number of image features and ℓ is the number of eye movement features and m are the total number of samples. We continue to defin K^x, K^y as the kernel matrices for the ranked images and eye movements respectively. In our experiments we use linear kernels, i.e. $K^x = X'X$ and $K^y = Y'Y$. The resulting kernel matrix of the tensor $T = X \circ Y$ can be expressed as pair-wise product (see [5] for details)

$$\bar{K}_{ij} = (T'T)_{ij} = K^x_{ij} K^y_{ij}.$$

We use $\bar{K}$ in conjunction with the vanilla SVM formulation as given in equation (2). Whereas the set up and training are straight forward the underlying problem is that for testing we do not have the eye movements. Therefore we propose to decompose the resulting weight matrix from its corresponding image and eye components such that each can be used independently.

The goal is to decompose the weight matrix W given by a dual representation

$$W = \sum_i^m \alpha_i c_i \phi_x(\mathbf{x}_i) \circ \phi_y(\mathbf{y}_i)$$

without accessing the feature space. Given the paired samples $\mathbf{x}, \mathbf{y}$ the decision function in equation is

$$\begin{aligned} f(\mathbf{x}, \mathbf{y}) &= W \circ \phi_x(\mathbf{x}) \phi_y(\mathbf{y})' \\ &= \sum_{i=1}^m \alpha_i c_i \kappa_x(\mathbf{x}_i, \mathbf{x}) \kappa_y(\mathbf{y}_i, \mathbf{y}). \end{aligned}$$

3.1 Decomposition

We want to decompose the weight matrix into a sum of tensor products of corresponding weight components for the images and eye movements

$$W \approx W^T = \sum_{t=1}^T \mathbf{w}_x^t \mathbf{w}_y^{t\prime},$$

so that $\mathbf{w}_x^t = \sum_{i=1}^m \beta_i^t \phi_x(\mathbf{x}_i)$ and $\mathbf{w}_y^t = \sum_{i=1}^m \gamma_i^t \phi_y(\mathbf{y}_i)$ where $\boldsymbol{\beta}^t, \boldsymbol{\gamma}^t$ are the dual variables of $\mathbf{w}_x^t, \mathbf{w}_y^t$.

We compute

$$WW' = \sum_{i,j}^m \alpha_i \alpha_j c_i c_j \kappa_y(\mathbf{y}_i, \mathbf{y}_j) \phi_x(\mathbf{x}_i) \phi_x(\mathbf{x}_j)' \tag{3}$$

and are able to express $K^y = (\kappa_y(\mathbf{y}_i, \mathbf{y}_j))_{i,j=1}^m = \sum_{k=1}^K \lambda_k \mathbf{u}^k \mathbf{u}^{k\prime} = U \Lambda U'$, where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_K)$ by performing an eigenvalue decomposition of the kernel matrix K^y with entries $K^y_{ij} = \kappa_y(\mathbf{y}_i, \mathbf{y}_j)$. Substituting back into equation (3) gives

$$WW' = \sum_k^K \lambda_k \sum_{i,j}^m \alpha_i \alpha_j c_i c_j \mathbf{u}_i^k \mathbf{u}_j^{k\prime} \phi_x(\mathbf{x}_i) \phi_x(\mathbf{x}_j)'.$$

Letting $\mathbf{h}_k = \sum_{i=1}^{m} \alpha_i c_i \mathbf{u}_i^k \phi_x(\mathbf{x}_i)$ we have $WW' = \sum_k^K \lambda_k \mathbf{h}_k \mathbf{h}_k' = HH'$ where $H = \left(\sqrt{\lambda_1}\mathbf{h}_1, \ldots, \sqrt{\lambda_K}\mathbf{h}_K\right)$. We would like to fin the singular value decomposition of $H = V\Upsilon Z'$. Consider for $A = diag(\boldsymbol{\alpha})$ and $C = diag(\mathbf{c})$ we have

$$\begin{aligned}[H'H]_{k\ell} &= \sqrt{\lambda_k \lambda_\ell} \sum_{ij} \alpha_i \alpha_j c_i c_j \mathbf{u}_i^k \mathbf{u}_j^\ell \kappa_x(\mathbf{x}_i, \mathbf{x}_j) \\ &= \left[\left(CAU\Lambda^{\frac{1}{2}} \right)' K^x \left(CAU\Lambda^{\frac{1}{2}} \right) \right]_{k\ell},\end{aligned}$$

which is computable without accessing the feature space. Performing an eigenvalue decomposition on $H'H$ we have

$$H'H = Z\Upsilon V' V \Upsilon Z' = Z\Upsilon^2 Z'$$

with Υ a matrix with υ_t on the diagonal truncated after the j^{th} eigenvalue, which gives the dual representation of $\mathbf{v}_t = \frac{1}{\upsilon_t} H\mathbf{z}_t$ for $t = 1, \ldots, T$, and since $H'H\mathbf{z}_t = \upsilon_t^2 \mathbf{z_t}$ we are able to verify that

$$WW'\mathbf{v}_t = HH'\mathbf{v}_t = \frac{1}{\upsilon_t} HH'H\mathbf{z}_t = \upsilon_t H\mathbf{z}_t = \upsilon_t^2 \mathbf{v}_t.$$

Restricting to the firs T singular vectors allows us to express $W \approx W^T = \sum_{t=1}^{T} \mathbf{v}_t \left(W'\mathbf{v}_t\right)'$, which in turn results in

$$\mathbf{w}_x^t = \mathbf{v}_t = \frac{1}{\upsilon_t} H\mathbf{z}_t = \sum_{i=1}^{m} \beta_i^t \phi_x(\mathbf{x}_i),$$

where $\beta_i^t = \frac{1}{\upsilon_t} \alpha_i c_i \sum_{k=1}^{T} \sqrt{\lambda_k} \mathbf{z}_k^t u_i^k$. We can now also express

$$\mathbf{w}_y^t = W'\mathbf{v}_t = \frac{1}{\upsilon_t} W'H\mathbf{z}_t = \sum_{i=1}^{m} \gamma_i^t \phi_y(\mathbf{y}_i),$$

where $\gamma_i^t = \sum_{j=1}^{m} \alpha_i c_i \beta_j^t \kappa_x(\mathbf{x}_i, \mathbf{x}_j)$ are the dual variables of $\mathbf{w}_y^t$. We are therefore now able to decompose W into W_x, W_y without accessing the feature space giving us the desired result.

We are now able to compute, for a given t, the ranking scores in the linear discriminant analysis form $s = \mathbf{w}_x^t \hat{X}$ for new test images $\hat{X}$. These are in turn sorted in order of magnitude (importance). Equally, we can project our data into the new define semantic space β where we train and test an SVM. i.e. we compute $\tilde{\phi}(\mathbf{x}) = K^x \beta$, for the training samples, and $\tilde{\phi}(\mathbf{x}_t) = K_t^x \beta$ for our test samples. We explore both these approaches in our experiments.

4 Experiments

We evaluate two different scenarios for learning the ranking of image based on image and eye features; 1) Predicting rankings on a page given only other data from a single specifi user. 2) A global model using data from other users to predict rankings for a new unseen user. We compute a 256-bin grey scale histogram on the whole image as the feature representation. These features are intentionally kept relatively simple. The 33 eye movement features are computed based only on the eye trajectory and locations of the images in the page. This type of features are general-purpose and are easily applicable to all application scenarios. We compare our proposed tensor Ranking SVM algorithm which combines both information from eye movements and image histogram features to a Ranking SVM using either of these features alone, and to a Ranking SVM using both eye movements and histogram features. We further emphasis that training and testing a model using only eye movements is *not realistic* as there are no eye movements presented a-priori for new images, i.e. one can not test. This comparison provides us with a baseline as to how much it may be possible to improve on the performance using eye movements. In the experiments we use a linear kernel function. Although, it is possible to use a non-linear kernel on the eye movement features as this would not effect the decomposition for the image weights. See [4] for more details on experiment setup and feature extraction.

4.1 Page Generalisation

In the following section we focus on predicting rankings on a page given only other data from a single specifi user. We employ a leave-page-out routine where at each iteration a page, from a given user, is withheld for testing and the remaining pages, from the same user, are used for training.

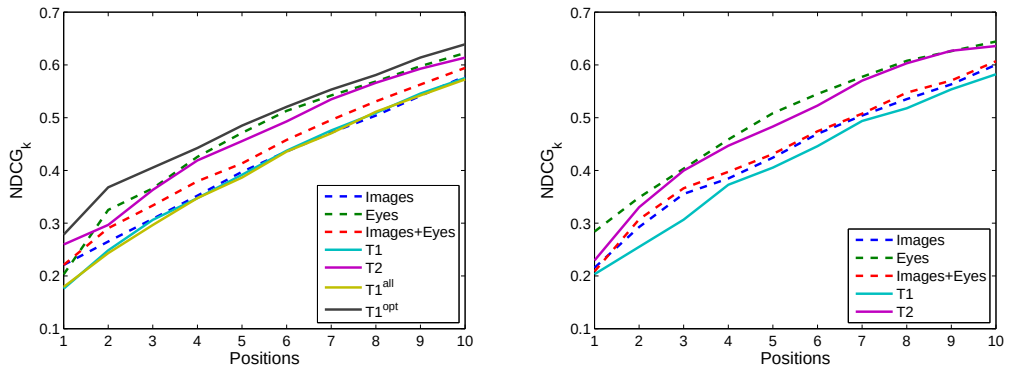
We evaluate the proposed approach with the following four setting: (1) $T1$: using the largest component of tensor decomposition in the form of a linear discriminator. We use the weight vector corresponding to the largest eigenvalue (as we have a t weights). (2) $T2$: we project the image features into the learnt semantic space (i.e. the decomposition on the image source) and train and test within the projected space a secondary Ranking SVM. (3) $T1^{all}$: similar to $T1$ although here we use all t weight vectors and take the mean value across as the fina score. (4) $T1^{opt}$: similar to $T1$ although here we use the n-largest components of the decomposition. i.e. we select n weight vectors to use and take the mean value across as the fina score.

We use a leave-one-out cross-validation for $T1^{opt}$ to obtain the optimal model for the later case which are selected based on maximum average Normalised Discount Cumulative Gain (NDCG) across 10 positions.

In figur 1(a) we plot the average performance across all users. The figur shows that $T1$ and $T1^{all}$ are slightly worse than using image histogram alone. However, when we carefully select the number of largest components in tensor decomposition, the performance of the classifie is greatly improved and clearly outperforms the Ranking SVM with eye movements. Using classifie $T2$, the performance is improved above the Ranking SVM with image features and it is competitive with Ranking SVM with eye movements features.

4.2 User Generalisation

In the following section we focus on learning a global model using data from other users to predict rankings for a new unseen user. Although, as the experiment is set up such that each user views the same pages as all other users we employ a leave-user-leave-page-out routine. We evaluate the proposed approach with the following two setting: (1) $T1$ (2) $T2$. We plot in figur 1(b) the average NDCG performance on the leave-user-leave-page-out routine, demonstrating that on average we improve on the ranking of new images for new users.



(a) Page generalisation (b) User generalisation

Figure 1: Sub-figure 1(a) shows average NDCG performance across all users for predicting rankings on a page given only other data from a single specifi user. Sub-figure 1(b) shows the average NDCG performance where we are able to observe that $T2$ outperforms the ranking of only using image features. The 'Eyes' and 'Images+Eyes' plots in all the figure demonstrate how the ranking would perform if eye-features were indeed available a-priori for new images.

5 Discussion

Improving search and content based retrieval systems with implicit feedback is an attractive possibility given that a user is not required to explicitly provide information to then improve, and personalise, their search strategy. This, in turn, can render such a system more user-friendly and simple to use (at least from the users' perspective). Although, achieving such a goal is non-trivial as one needs to be able to combine the implicit feedback information into the search system in a manner that does not then require the implicit information for testing. In our study we focus on implicit feedback in the form of eye movements, as these are easily available and can be measured in a non-intrusive manner. Previous studies [1] have shown the feasibility of such systems using eye moments for a textual search task. Demonstrating that it is indeed possible to 'enrich' a textual search with eye features. Although their proposed approach is computationally complex since it requires the construction of a regression function on eye measurements on each word. This was not realistic in our setting. Furthermore, [4] had extend the underlying methodology of using eye movement as implicit feedback to an image retrieval system, combining eye movements with image features to improve the ranking of retrieved images. Although, still, the proposed approach required eye features for the test images which would not be practical in a real system. We propose a novel search strategy for combining eye movements and image features with a tensor product kernel used in a Ranking SVM framework.

Acknowledgments

The authors would like to acknowledge financia support from the European Community's Seventh Framework Programme (FP7/2007–2013) under *grant agreement* n° 216529, Personal Information Navigator Adapting Through Viewing (PinView) project (http://www.pinview.eu) and the IST Programme of the European Community, PASCAL2 Network of Excellence (http://www.pascal-network.org), IST-2007-216886. This publication only reflect the authors' views. The authors would also like to thank Craig J. Saunders for data collection.

References

[1] Antti Ajanki, David R. Hardoon, Samuel Kaski, Kai Puolamäki, and John Shawe-Taylor. Can eyes reveal interest? Implicit queries from gaze patterns. *User Modeling and User-Adapted Interaction*, 19(4):307–339, 2009.

[2] Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein-protein interactions. *Bioinformatics*, 21:i38–i46, 2005.

[3] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[4] Kitsuchart Pasupa, Craig Saunders, Sandor Szedmak, Arto Klami, Samuel Kaski, and Steve Gunn. Learning to rank images from eye movements. In *HCI '09: Proceeding of the IEEE 12th International Conference on Computer Vision (ICCV'09) Workshops on Human-Computer Interaction*, pages 2009–2016, 2009.

[5] Sylvia Pulmannová. Tensor products of Hilbert space effect algebras. *Reports on Mathematical Physics*, 53(2):301–316, 2004.

[6] Jian Qiu and William Stafford Noble. Predicting co-complexed protein pairs from heterogeneous data. *PLoS Computational Biology*, 4(4):e1000054, 2008.

Scalable Online Learning to Rank from Clicks*

Filip Radlinski
Microsoft Research
Cambridge, UK

Aleksandrs Slivkins
Microsoft Research
Mountain View, CA, USA

Sreenivas Gollapudi
Microsoft Research
Mountain View, CA, USA

{filiprad, slivkins, sreenig} @ microsoft.com

Abstract

Web search has motivated a tremendous amount of research in learning to rank. Moreover, it is increasingly recognized that the relevance of documents is not independent, and that learned ranking functions should avoid redundant results. We present an online algorithm that learns to rank by optimizing the fraction of satisfied users, building upon previous work with a scalable algorithm that also explicitly takes document similarity into account. In addition, we provide theoretical justifications for this approach as well as a provably near-optimal algorithm, while also showing that our algorithms learn orders of magnitude faster than the previous approaches.

1 Introduction

Learning to rank results to web search queries has motivated much research (e.g. [4, 8, 10, 13]). One increasingly important goal has been to learn from interactions between users and search engines, such as clicks on results. A key consideration often missing in models learned from these interactions is that the relevance of documents cannot be estimated in isolation. For example, any click is dependent on the specific documents shown earlier. Most previous work avoiding redundancy has focused on reranking results (e.g. [1, 5, 14]). Instead, we focus on directly learning a diverse ranking of documents from user interactions and improve upon results presented in [10].

Learning to rank can also be seen as an online learning problem, where we wish to minimize the time it takes to learn an ideal ranking. Multi-Armed Bandit (MAB) algorithms [6] can optimally solve many online sequential learning problems, where the algorithm must choose strategies sequentially with the largest total payoff. They are ideal for settings that admit exploration/exploitation tradeoffs such as are seen in web search. In fact, a ranking function that minimizes abandonment can be learnt by a MAB [10, 12]. However, most MAB algorithms are impractical at web scales. For instance, [10] evaluate their approach by learning to rank 50 documents using hundreds of thousands of simluated user interactions with their ranking algorithm.

With the aim of improving convergence rates in standard multi-armed bandit learning settings, prior work on MAB algorithms has considered exploiting structure in the strategy space. One approach by Kleinberg et al. [9] is well suited to our scenario as well: The authors propose a solution when the strategies available to the MAB algorithm form a metric space, with respect to which the payoff function satisfies a Lipschitz condition. The metric space allows a similarity measure to be defined, thereby allowing the algorithm to make inferences about similar strategies without having to explicitly explore them. Further, they propose a *zooming algorithm* that learns to adaptively refine regions of the strategy space where there is likelihood of higher payoff.

In web search there are additional signals that one can exploit to make the zooming approach even more effective: A search user typically scans search results top down and clicks on more relevant

*This paper is a preliminary account of an ongoing research project. A more definitive version will appear as a forthcoming conference submission. Due to the page limit, all proofs are omitted.

documents. One can therefore infer the *context* of a click: The documents at higher ranks that were skipped [7]. We also note that *correlation* between the probability that two documents are simultaneously both relevant (or not) provides useful information.

Building upon the "ranked bandits" approach in [10, 12] and the "zooming" approach in [9, 11], we present a new model that incorporates both document similarity and learning a ranking. We show that using higher ranked documents as context, and making use of correlation between documents when choosing which document to present next leads to a theoretically optimal algorithm that also has a drastically improved empricial convergence rate compared to previous work.

2 The learning problem

Users and clicks. Following [10], we learn an optimally diverse ranking of documents. Let X be a set of documents. Each "user" is represented by a function $\pi : X \to \{0, 1\}$, where $\pi(x) = 1$ means that document $x \in X$ is "relevant" to the user. Let $\mathcal{F}_X$ be a set of all functions $X \to \{0, 1\}$, i.e. the set of all possible users. In each round, a user is sampled independently from some fixed *user distribution* $\mathcal{P}$ on $\mathcal{F}_X$. An algorithm outputs a list of k results. The user clicks on the first relevant result. The goal is to learn a ranking that maximizes the fraction of users who click.

Metric spaces and correlations. Let $(X, \mathcal{D})$ be a metric space. A function $\mu : X \to \mathbb{R}$ is *Lipschitz-continuous* with respect to $(X, \mathcal{D})$ if $|\mu(x) - \mu(y)| \leq \mathcal{D}(x, y)$ for all $x, y \in X$. We define a distribution $\mathcal{P}$ to be *Lipschitz-correlated* w.r.t. $(X, \mathcal{D})$ if $\Pr_{\pi\sim\mathcal{P}}[\pi(x) \neq \pi(y)] \leq \mathcal{D}(x, y)$ for all $x, y \in X$, and *conditionally* Lipschitz-correlated if this property holds conditional on the event "all documents in set S are irrelevant", for any given $S \subset X$. For brevity, we will use *L-continuous* and *L-correlated* to denote these properties in the remainder of this paper.

Document model. Web documents are often classified into hierarchies[1], where closer pairs are more similar. For evaluation, we assume the documents X fall in such a tree, with each document $x \in X$ a leaf in the tree. On this tree, we induce the *ϵ-exponential tree metric*: the distance between two nodes is exponential in the depth of their least common ancestor, with base $\epsilon \in (0, 1)$.

Our task. Our task is to solve the *k-slot L-correlated MAB problem*. An instance of this problem consists of a triple $(X, \mathcal{D}, \mathcal{P})$, where $(X, \mathcal{D})$ is a metric space, and $\mathcal{P}$ is a distribution on $\mathcal{F}_X$ which is conditionally L-correlated. One could also consider the more permissive version where we only require the pointwise mean of $\mathcal{P}$ to be conditionally L-continuous (in the same sense).

3 Model Expressiveness

We rely on the novel *L-correlation* property, hence must argue that this property is plausible in reality. Given any metric space $(X, \mathcal{D})$ and any L-continuous function μ on it, we now provide a construction for a rich family of user distributions with pointwise mean μ that are (essentially) L-correlated with respect to $(X, \mathcal{D})$. Also, we use this construction for evaluation.

We focus on the case of exponential tree metrics, as described above. Let $\mathcal{D}$ be a metric for a tree with node set V and leaf set $X \subset V$. We start with a function $\mu : V \to (\alpha, \frac{1}{2}]$, $\alpha > 0$, that is L-continuous with respect to $(V, \mathcal{D})$. We can show (details omitted) that μ can always be extended from X to V so that $\mu : V \to [\alpha, \frac{1}{2}]$ is L-continuous with respect to $(V, \mathcal{D})$. We pick $\pi(\text{root}) \in \{0, 1\}$ at random with a suitable expectation, and then proceed top-down so that the child's click is obtained from the parent's click via a low-probability mutation. The mutation is parameterized by functions $q_0, q_1 : V \to [0, 1]$, as described in Algorithm 1.

Theorem 3.1 *Fix $\alpha > 0$ and a function $\mu : X \to [\alpha, \frac{1}{2}]$ that is L-continuous w.r.t. $(X, \mathcal{D})$. Then for any suitable functions q_0, q_1 the user distribution π constructed by Algorithm 1 has pointwise mean μ. Moreover, it is L-correlated with respect to $\mathcal{D}_\mu(x, y) \triangleq \mathcal{D}(x, y) \times \min\left(\frac{1}{\alpha}, \frac{3}{\mu(x)+\mu(y)}\right)$ and conditionally L-correlated w.r.t. $\mathcal{D}_\mu(x, y) \times \frac{2}{1-\mathcal{D}_\mu(x,y)}$.*

The theorem extends, with slightly worse constants, to arbitrary metric space via *metric embeddings*, although we omit the details here.

[1] For example, the Open Directory Project at http://dmoz.org/

Algorithm 1 User distribution for tree metrics

Input: Tree (root r, node set V); $\mu(r) \in [0,1]$
mutation probabilities $q_0, q_1 : V \to [0,1]$
Output: random click vector $\pi : V \to \{0,1\}$

function AssignClicks(tree node x)
$b \leftarrow \pi(x)$
for each child y of x **do**
$\pi(y) \leftarrow \begin{cases} 1-b & \text{w/prob } q_b(y) \\ b & \text{otherwise} \end{cases}$
AssignClicks(y)

Pick $\pi(r) \in \{0,1\}$ at random with expect. $\mu(r)$
AssignClicks(r)

Mutation probabilities must satisfy:

$$\begin{cases} \mu(y) = (1-\mu(x))\, q_0(y) + \mu(x)(1-q_1(y)) \\ q_0(y) + q_1(y) \le \mathcal{D}(x,y) / \min(\mu(x),\, \mu(y)). \end{cases}$$

whenever y is a child of x.

The former is to ensure that $\mathbb{E}[\pi(\cdot)] = \mu(\cdot)$; the latter is to bound the discorrelations.

E.g., the pair $(q_0(y),\, q_1(y))$ can be defined as

$$\begin{cases} \left(0,\, \frac{\mu(x)-\mu(y)}{\mu(x)}\right) & \text{if } \mu(x) \ge \mu(y) \\ \left(\frac{\mu(y)-\mu(x)}{1-\mu(x)},\, 0\right) & \text{otherwise.} \end{cases}$$

4 Algorithms

We now present algorithms for the k-slot L-correlated MAB problem. We start with simple adaptations of existing work, followed by two new algorithms that explicitly take context into account.

Non-Contextual Approaches. Let `Bandits` be some algorithm for the MAB problem. The `RankedBandits` algorithm for the multi-slot MAB problem is defined as per [10]: We have k slots (i.e. ranks) for which we wish to find the best document to present. In each slot, a separate copy of `Bandits` is instantiated. In each round, if a user clicks on slot i, then this slot receives a payoff of 1, and all higher (i.e. skipped) slots $j < i$ receive a payoff of 0. For slots $j > i$, the state is rolled back as if this round had never happened (i.e. assuming that the user never considered these documents). If no slot is clicked, then all slots receive a payoff of 0 (i.e. assuming the user scanned all k documents, and considered none relevant). In [10], this approach gives rise to algorithms `RankedUCB1` and `RankedEXP3`, based on bandit algorithms `UCB1` [2] and `EXP3` [3]. To take the metric space into account, we also consider algorithm `RankedZooming`, the "ranked" version of the "zooming algorithm" from [9].

Ranked contextual algorithms. In the `RankedBandits` algorithm, the slot i algorithm $\mathcal{A}_i$ is invoked if and only if none of the upper slots is clicked. Thus, the slot algorithms can make their selections sequentially: $\mathcal{A}_i$ knows the set S of documents in the upper slots. We propose to treat S as a "context" to $\mathcal{A}_i$. Abstractly, algorithm $\mathcal{A}_i$ operates in the following *contextual bandits* setting: In each round nature reveals a *context* h, $\mathcal{A}_i$ chooses an arm x, and the resulting payoff is an independent $\{0,1\}$ sample with expectation $\mu(x|h)$. Furtheremore, the algorithm is given access to a metric space $(H, \mathcal{D}_{\text{H}})$, allowing it to generalize from a given context $S \subset X$ to "similar" ones. For our setting, we define $\mathcal{D}_{\text{H}}(S,S') \triangleq \inf \sum_{i=1}^{n} \mathcal{D}(x_i, x'_i)$, where the infimum is taken over all $n \in \mathbb{N}$ and over all n-element sequences $\{x_i\}$ and $\{x'_i\}$ that enumerate, possibly with repetitions, all values in S and S'. We prove that $|\mu(x|S) - \mu(x|S')| \le O(\mathcal{D}_{\text{H}}(S,S'))$, where $\mu(x|S)$ is the conditional probability of clicking on document x given that all documents in S are non-relevant.

We will use a contextual bandit algorithm proposed by Slivkins [11] called `ContextualZooming`. Here, we present a version that is fine-tuned for the $(i+1)$-st slot in our setting,[2] with documents in an ϵ-exponential tree τ_D, in Algorithm 2. The algorithm maintains a set $\mathcal{A}$ of *active strategies* of the form (u, u'), where u is a subtree in τ_D (i.e. a set of similar documents), and u' is a set of contexts, namely a product of i subtrees of τ_D at the same depth. At any given time the active strategies partition the space of all (document, context) pairs, henceforth the *DC-space*. In each round, a context S arrives, and one of the strategies (u, u') with $S \in u'$ is chosen: namely the one with the maximal *index*. The index is, essentially, the best available upper confidence bound on expected payoffs from choosing a document $x \in u$ given a context $S \in u'$. Then a document in $x \in u$ is chosen uniformly at random. The *confidence radius* is defined as $\texttt{rad}(u,u') = \sqrt{4\log(T)} / \sqrt{1 + n(u,u')}$, where T is the time horizon, and $n(u,u')$ is the number of times this strategy has been chosen so far. The "width" of the subtree rooted at u is defined as $\texttt{W}(u,u') = \epsilon^{\text{depth}(u)} + 4i\,\epsilon^{\text{depth}(u')}$. The

[2] For $i = 1$, there are no contexts, and the algorithm reduces to the "zooming algorithm" from [9].

Algorithm 2 `ContextualZooming` in trees

initialize (context tree $\mathcal{C}$, τ_D, T)**:**
$u \leftarrow root(\tau_D)$; $u' \leftarrow root(\mathcal{C})$
$\mathcal{A} \leftarrow \{(u,u')\}$; $n(u,u') \leftarrow 0$; $r(u,u') \leftarrow 0$

select-strategy (context x)**:**
return $\operatorname{argmax}_{(u,u') \in \mathcal{A}:\, x \in u'} \mathtt{W}(u,u') + \frac{r(u,u')}{n(u,u')} + \mathtt{rad}(u,u')$

update (arm u, context c, reward r)**:**
$r(u,u') \leftarrow r(u,u') + r$; $n(u,u') \leftarrow n(u,u') + 1$
if $\mathtt{rad}(u,u') < \mathtt{W}(u,u')$ **then**
  deactivate (u,u')
  activate all pairs (children(u),children(u'))
endif

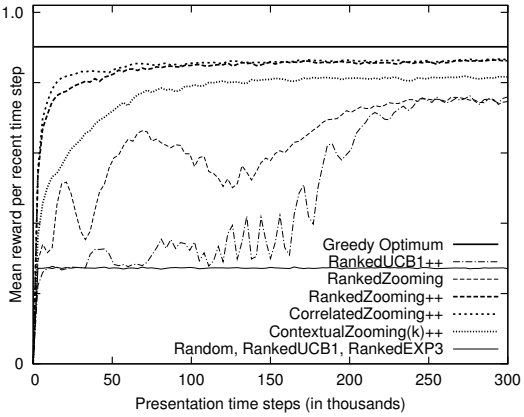


Figure 1: Comparison of learning algorithms.

crucial (de)activation rule ensures that the active strategies form a finer partition in the regions of the DC-space that correspond to higher payoffs and more frequently occurring contexts.

`ContextualZooming`(`k`) denotes an algorithm for the k-slot L-correlated MAB problem which is a "ranked" version of `ContextualZooming`, where each slot receives as context the set S of documents in the preceding slots. Using the anlysis in [11], one can prove that the expected average number of clicks of `ContextualZooming`(`k`) approaches the *greedy optimum*, the performance of an offline greedy algorithm that knows the user distribution and selects documents greedily slot by slot from top to bottom. The greedy optimum is a natural benchmark for "ranked" bandits algorithms since it corresponds to the intuition of what an "ideal" ranked bandit algorithm would do.[3] Currently this is the *only* algorithm for our problem for which we have this guarantee.

Finally, we can further exploit the conditional L-correlation as follows. Given the context S, the conditional expectation $\mu(x|S)$ is upper-bounded by $\mathcal{D}(x,S) \triangleq \min_{y \in S} \mathcal{D}(x,y)$ due to the conditional L-correlation. Thus, we may be able to decrease the index while keeping it a valid upper confidence bound on payoffs. Specifically, we can "upgrade" `ContextualZooming`(`k`) with the following *correlation rule*: cap the index of each active strategy (u,u'), $S \in u'$ at $\max_{x \in u} \mathcal{D}(x,S)$. (Recall that u is a subtree of documents, and u' is a set of contexts.) One can show that this rule does not break the original analysis of `ContextualZooming`. By abuse of notation, we define `ContextualZooming`(`k`) to use this rule. We also define a version of `RankedZooming` that uses this rule: `CorrelatedZooming`. We view it as a "lighter" version of `ContextualZooming`(`k`).

5 Evaluation

We now evaluate the performance of the various algorithms. We generated a collection with $|X| = 2^{15} \approx 32,000$ documents[4] in a binary exponential tree metric space with edge weights $\epsilon = 0.837$[5]. Each document's expected relevance to users $\mu(x)$ was set by first identifying a small number of likely relevant documents $y_i \in Y$, then computing the relevance of other documents as the minimum allowed while obeying L-continuity and a background relevance rate μ_0: $\mu(x) = \max(\mu_0, 0.5 - \min_i \mathcal{D}(x,y_i))$. We simulated 300,000 user visits to this document collection, with users sampled from $\mathcal{P}$ as described in Section 3. Our simulation was run over a 5-slot ranked bandit setting, learning the best 5 documents. Figure 5 shows the performance of the algorithms where the collection was initialized such that two randomly chosen documents are selected as most relevant, and using $\mu_0 = 0.05$[6] (averaging over 1,000 runs). Performance within the first 50,000 impressions, typical for the number of times frequent web queries are seen by commercial search engines in a month, is essential for any practical applicability of this approach.

[3]Note that although in the worst case *greedy optimum* may have large regret relative to the optimal overall ranking (see e.g. [10] for details), in most cases it outperforms this worst case bound substantially.

[4]A realistic number of documents that may need to be considered in detail for a web search query.

[5]Chosen such that the most dissimilar documents still have a non-trivial similarity.

[6]Chosen such that all documents have at least a small chance of being relevant for a user.

Following [10], we find that RankedUCB1, RankedZooming and ContextualZooming(k) (which share the UCB1-style "index") perform better empirically if the $(\log T)$ factor in the "index" is replaced with constant 1. We denote this optimization by appending '++' to the algorithm's name, e.g. RankedUCB1++.

RankedEXP3 and RankedUCB1 perform as poorly as picking documents randomly, due to the large number of available strategies and slow convergence rates of these algorithms. This is consistent with [10], who evaluate on just 50 documents. Making the algorithms "optimistic" ('++') improved performance dramatically. We also see that RankedZooming performs comparably to optimistic RankedUCB1, and becomes extremely effective if made optimistic. Optimistic CorrelatedZooming achieves the best empirical performance, converging rapidly to close to optimal rankings.

Interestingly, the theoretically preferred ContextualZooming does not converge as quickly as CorrelatedZooming in this preliminary evaluation. This appears to be due to the much larger branching factor in the strategies activated by ContextualZooming. We plan to investigate this further in future work.

6 Conclusions

We proposed a learning to rank model that explicitly considers correlation of clicks and similarity between documents, showing that this model admits efficient multi-slot ranked bandit algorithms that can exploit the browsing behavior of a search user. This allows our algorithms to scale to much larger collections than those to which MAB-based learning to rank algorithms could be applied to in the past. We confirmed this with preliminary empirical simulations.

References

[1] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *Proceedings of WSDM*, pages 5–14, 2009.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002. Preliminary version in *15th ICML*, 1998.

[3] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002. Preliminary version in *36th IEEE FOCS*, 1995.

[4] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of ICML*, 2005.

[5] Jamie Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of SIGIR*, pages 335–336, 1998.

[6] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge Univ. Press, 2006.

[7] Harr Chen and David R Karger. Less is more: Probabilistic models for retrieving fewer relevant documents. In *Proceedings of SIGIR*, 2006.

[8] Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning*, 6:1019–1041, 2005.

[9] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-Armed Bandits in Metric Spaces. In *40th ACM Symp. on Theory of Computing (STOC)*, pages 681–690, 2008.

[10] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *25th Intl. Conf. on Machine Learning (ICML)*, pages 784–791, 2008.

[11] Aleksandrs Slivkins. Contextual Bandits with Similarity Information. http://arxiv.org/abs/0907.3986, July 2009.

[12] Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *Advances in Neural Information Processing Systems 21*, pages 1577–1584. 2009.

[13] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A Support Vector Method for Optimizing Average Precision. In *Proceedings of SIGIR*, pages 271–278, 2007.

[14] Yisong Yue and Thorsten Joachims. Predicting diverse subsets using structural SVMs. In *Proceedings of ICML*, pages 1224–1231, 2008.

Simple Risk Bounds for Position-Sensitive Max-Margin Ranking Algorithms

Stefan Riezler
Google Research
Brandschenkestrasse 110
8002 Zürich, Switzerland
riezler@google.com

Fabio De Bona*
Friedrich Miescher Laboratory
of the Max Planck Society
Spemannstrasse 39
72076 Tübingen, Germany
fabio@tuebingen.mpg.de

Abstract

We present risk bounds for position-sensitive max-margin ranking algorithms that follow straightforwardly from a structural result for Rademacher averages presented by [1]. We apply this result to pairwise and listwise hinge loss that are position-sensitive by virtue of rescaling the margin by a pairwise or listwise position-sensitive prediction loss.

1 Introduction

[2] recently presented risk bounds for probabilistic listwise ranking algorithms. The presented bounds follow straightforwardly from structural results for Rademacher averages presented by [1]. These bounds are dominated by two terms: Firstly, by the empirical Rademacher average $\mathcal{R}_n(\mathcal{F})$ of the class of ranking functions $\mathcal{F}$; secondly, by a term involving the Lipschitz constant of a Lipschitz continuous loss function. For example, for a loss function defined on the space of all possible permutations over m ranks, the Lipschitz constant involves a factor $m!$. Loss functions defined over smaller spaces involve smaller factors.

Similar risk bounds can be given for max-margin ranking algorithms based on the hinge-loss function. The bounds make use of a single structural result on Rademacher averages that reflects the structure of the output space in the Lipschitz constant of the hinge-loss function. We apply the result to pairwise and listwise hinge loss functions that are both position-sensitive by virtue of rescaling the margin by a pairwise or listwise position-sensitive prediction loss. Position-sensitivity means that high precision in the top ranks is promoted, corresponding to user studies in web search that show that users typically only look at the very top results returned by the search engine [3].

The contribution of this paper is to show how simple risk bounds can be derived for max-margin ranking algorithms by a straightforward application of structural results for Rademacher averages presented by [1]. More involved risk bounds for pairwise ranking algorithms have been presented before by [4] (using algorithmic stability), and for structured prediction by [5] (using PAC-Bayesian theory).

2 Notation

Let $Y = \{r_1, r_2, \ldots, r_m\}$ be a set of totally ordered ranks over documents. Let $S = \{q^{(i)}\}_{i=1}^n = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n$ be a training sample, where each query $q \in Q$ is represented by a set of documents $x_q = \{x_{q1}, \ldots, x_{q,n(q)}\}$, and a ranking $y_q = (y_{q1}, \ldots, y_{q,n(q)})$ on documents, where $n(q)$ is

*The work presented in this paper was done while the author was visiting Google Research, Zürich.

the size of the lists x_q or y_q for query q. Ranks can take on values from 1 to m for full rankings of all documents where $m = n(q)$. Other scenarios include multipartite ranking involving $r < n(q)$ relevance levels, or bipartite rankings involving two ranks (1 for relevant, 2 for non-relevant documents). We consider the case of full rankings where $m = n(q)$ as the default case.

Let the documents in x_q be identified by the integers $\{1, 2, \ldots, n(q)\}$. Then a permutation π_q on x_q can be defined as a bijection from $\{1, 2, \ldots, n(q)\}$ onto itself. We use Π_q to denote the set of all possible permutations on x_q, and $\pi_q(j)$ to denote the rank position of document x_{qj}. Furthermore, let (i, j) denote a pair of documents in x_q and let $\mathcal{P}_q$ be the set of all pairs in x_q.

A feature function $\phi(x_{qi})$ is associated with each document $i = 1, \ldots, n(q)$ for each $q \in Q$. Furthermore, a partial-order feature map as used in [6, 7] is created for each document set as follows:

$$\phi(x_q, \pi_q) = \frac{1}{n(q)(n(q)-1)/2} \sum_{(i,j)\in\mathcal{P}_q} \phi(x_{qi}) - \phi(x_{qj}) \text{sgn}(\frac{1}{\pi_{qi}} - \frac{1}{\pi_{qj}}).$$

We assume linear ranking functions $f \in \mathcal{F}$ that are defined on the document level as $f(x_{qi}) = \langle w, \phi(x_{qi})\rangle$ and on the query level as $f(x_q, \pi_q) = \langle w, \phi(x_q, \pi_q)\rangle$. Note that since feature vectors on document and query level have the same size, assuming that $||w|| \leq B$, $||\phi|| \leq M$, we get $||f|| \leq BM$ for all $f \in \mathcal{F}$.

The goal of learning a ranking over the documents x_q for a query q can be achieved either by sorting the documents according to the document-level ranking function $f(x_{qi}) = \langle w, \phi(x_{qi})\rangle$, or by finding the permutation π^* that scores highest according to the query-level ranking function:

$$\pi^* = \arg\max_{\pi_q \in \Pi_q} f(x_q, \pi_q) = \langle w, \phi(x_q, \pi_q)\rangle .$$

For convenience, let us furthermore define ranking-difference functions on the document level

$$\bar{f}(x_{qi}, x_{qj}, y_{qi}, y_{qj}) = \langle w, \phi(x_{qi}) - \phi(x_{qj})\rangle \text{ sgn}(\frac{1}{y_{qi}} - \frac{1}{y_{qj}}),$$

and on the query level

$$\bar{f}(x_q, y_q, \pi_q) = \langle w, \phi(x_q, y_q) - \phi(x_q, \pi_q)\rangle .$$

Finally, let $L(y_q, \pi_q) \in [0, 1]$ denote a prediction loss of a predicted ranking π_q compared to the ground-truth ranking y_q.

3 Position-Sensitive Max-Margin Ranking Algorithms

A position-sensitive pairwise max-margin algorithm can be given by extending the magnitude-preserving pairwise hinge-loss of [4] or [8]. For a fully ranked list of instances as gold standard, the penalty term can be made position-sensitive by accruing the magnitude of the difference of inverted ranks instead of the magnitude of score differences. Thus the penalty for misranking a pair of instances is higher for misrankings involving higher rank positions than for misrankings in lower rank positions. The pairwise hinge loss is defined as follows (where $(z)_+ = \max\{0, z\}$):

Definition 1 (Pairwise Hinge Loss).

$$\ell_P(\bar{f}; x_q, y_q) = \sum_{(i,j)\in\mathcal{P}_q} ((|\frac{m}{y_{qi}} - \frac{m}{y_{qj}}|) - \bar{f}(x_{qi}, x_{qj}, y_{qi}, y_{qj}))_+.$$

We use the pairwise 0-1 error ℓ_{0-1} as basic ranking loss function for the pairwise case. Clearly, $\ell_{0-1}(\bar{f}; x_q, y_q) \leq \ell_P(\bar{f}; x_q, y_q)$ for all $\bar{f}, x_q, y_q$. The 0-1 error is defined as follows (where $[\![z]\!] = 1$ if z is true, 0 otherwise):

Definition 2 (0-1 Loss).

$$\ell_{0-1}(\bar{f}; x_q, y_q) = \sum_{(i,j)\in\mathcal{P}_q} [\![\bar{f}(x_{qi}, x_{qj}, y_{qi}, y_{qj}) < 0]\!].$$

Listwise max-Margin algorithms for the prediction loss of (Mean) Average Precision (AP) [9] and NDCG [10] have been presented by [6] and [7], respectively. These ranking algorithms are position-sensitive by virtue of position-sensitivity of the deployed prediction loss L. The listwise hinge loss for general L is defined as follows:

Definition 3 (Listwise Hinge Loss).

$$\ell_L(\bar{f}; x_q, y_q) = \sum_{\pi_q \in \Pi_q \setminus y_q} (L(y_q, \pi_q) - \bar{f}(x_q, y_q, \pi_q))_+.$$

The basic loss function for the listwise case is defined by the prediction loss L itself. For example, the prediction loss L_{AP} for AP on the query level is defined as follows with respect to binary rank labels $y_{qj} \in \{1, 2\}$:

Definition 4 (AP Loss).

$$L_{AP}(y_q, \pi_q) = 1 - AP(y_q, \pi_q)$$

$$\textit{where } AP(y_q, \pi_q) = \frac{\sum_{j=1}^{n(q)} Prec(j) \cdot (y_{qj} - 1)}{\sum_{j=1}^{n(q)} (y_{qj} - 1)}$$

$$\textit{and } Prec(j) = \frac{\sum_{k:\pi_q(k) \leq \pi_q(j)} (y_{qk} - 1)}{\pi_q(j)}.$$

4 Risk Bounds

We use the usual definitions of expected and empirical risk with respect to a loss function ℓ:

$$R_\ell(\bar{f}) = \int_Q \ell(\bar{f}; x_q, y_q) P(dx_q, dy_q).$$

$$\hat{R}_\ell(\bar{f}; S) = \frac{1}{n} \sum_{i=1}^{n} \ell(\bar{f}; x_q^{(i)}, y_q^{(i)}), \text{ where } S = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n.$$

[1]'s central theorem on risk bounds using Rademacher averages can be restated with respect the definitions above as follows:

Theorem 1 (cf. [1], Theorem 8). *Assume loss functions $\tilde{\ell}(\bar{f}; x_q, y_q) \in [0, 1]$, $\ell(\bar{f}; x_q, y_q) \in [0, 1]$ where ℓ dominates $\tilde{\ell}$ s.t. for all $\bar{f}, x_q, y_q$, $\tilde{\ell}(\bar{f}; x_q, y_q) \leq \ell(\bar{f}; x_q, y_q)$. Let $S = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n$ be a training set of i.i.d. instances, and $\bar{\mathcal{F}}$ be the class of linear ranking-difference functions. Then with probability $1 - \delta$ over samples of length n, the following holds for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$R_{\tilde{\ell}}(\bar{f}) \quad \leq \quad \hat{R}_\ell(\bar{f}; S) + \mathcal{R}_n(\ell \circ \bar{\mathcal{F}}) + \sqrt{\frac{8 \ln(2/\delta)}{n}}$$

$$\textit{where} \quad \mathcal{R}_n(\ell \circ \bar{\mathcal{F}}) = \mathbb{E}_\sigma \sup_{\bar{f} \in \bar{\mathcal{F}}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i \ell(\bar{f}; x_q^{(i)}, y_q^{(i)}).$$

The complexity measure of a Rademacher average $\mathcal{R}_n(F)$ on a class of functions F quantifies the extent to which some function in F can be correlated with a random noise sequence of length n. Here the Rademacher average $\mathcal{R}_n(\ell \circ \bar{\mathcal{F}})$ is defined on a class of functions that is composed of a Lipschitz continuous loss function ℓ and a linear ranking model in $\bar{\mathcal{F}}$. It can be broken down into a Rademacher average $\mathcal{R}_n(\bar{\mathcal{F}})$ for the linear ranking models, and the Lipschitz constant L_ℓ for the loss function ℓ. The following theorem makes use of the Ledoux-Talagrand concentration inequality:

Theorem 2 (cf. [1], Theorem 12). *Let ℓ be a Lipschitz continuous loss function with Lipschitz constant L_ℓ, then $\mathcal{R}_n(\ell \circ \bar{\mathcal{F}}) \leq 2 L_\ell \mathcal{R}_n(\bar{\mathcal{F}})$.*

Furthermore, the Rademacher average for linear functions is given by the following Lemma:

Lemma 1 (cf. [1], Lemma 22). *Let $\bar{\mathcal{F}}$ be the class of linear ranking difference functions bounded by BM. Then for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$\mathcal{R}_n(\bar{\mathcal{F}}) = \frac{2BM}{\sqrt{n}}.$$

In order to apply Theorem 1, we need to normalize loss functions to map to $[0, 1]$. For full pairwise ranking, the size of the set of pairs over $m = n(q)$ ranks is $|\mathcal{P}_q| = \binom{m}{2}$. This yields normalization constants $Z_P = \binom{m}{2}(m - 1 + 2BM)$ for pairwise hinge loss, and $Z_{0-1} = \binom{m}{2}$ for 0-1 loss.

An application of Theorem 2 to pairwise hinge loss yields the following:

Proposition 1. *Let $\hat{\ell}_P = \frac{1}{Z_P}\ell_P$ be the normalized pairwise hinge loss. Then for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$\mathcal{R}_n(\hat{\ell}_P \circ \bar{\mathcal{F}}) \leq \frac{2}{m - 1 + 2BM}\mathcal{R}_n(\bar{\mathcal{F}}).$$

Proof. Follows directly from Theorem (2) with $L_{\hat{\ell}_P} = \sup_{\bar{f}} |\hat{\ell}'_P(\bar{f})| = |\frac{\binom{m}{2}}{\binom{m}{2}(m-1+2BM)|}|$. □

Using the 0-1 loss as dominated loss, we can combine Theorem 1 and Lemma 1 with Proposition 1 to get the following result:

Theorem 3. *Let ℓ_{0-1} be the 0-1 loss defined in Definition (2) and ℓ_P be the pairwise hinge loss defined in Definition (1). Let $S = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n$ be a training set of i.i.d. instances, and $\bar{\mathcal{F}}$ be the class of linear ranking-difference functions. Then with probability $1 - \delta$ over samples of length n, the following holds for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$R_{\ell_{0-1}}(\bar{f}) \leq \hat{R}_{\ell_P}(\bar{f}; S) + \binom{m}{2}\frac{4BM}{\sqrt{n}} + \binom{m}{2}(m - 1 + 2BM)\sqrt{\frac{8\ln(2/\delta)}{n}}.$$

Proof. Combining Theorem (1) and Proposition (1) under the use of normalized loss function $\hat{\ell}_P = \frac{1}{Z_P}\ell_P$, we get

$$R_{\hat{\ell}_P}(\bar{f}) \leq \hat{R}_{\hat{\ell}_P}(\bar{f}; S) + \frac{2}{m - 1 + 2BM}\frac{2BM}{\sqrt{n}} + \sqrt{\frac{8\ln(2/\delta)}{n}}.$$

Since for $c, F, G > 0$, the inequality $F \leq G$ implies $cF \leq cG$, we can rescale the result above to achieve a bound for the original loss functions.

$$Z_P[R_{\hat{\ell}_{0-1}}(\bar{f})] \leq Z_P[\hat{R}_{\hat{\ell}_P}(\bar{f}; S) + \frac{2}{m - 1 + 2BM}\frac{2BM}{\sqrt{n}} + \sqrt{\frac{8\ln(2/\delta)}{n}}].$$

Multiplying in the normalization constant gives

$$R_{\ell_P}(\bar{f}) \leq \hat{R}_{\ell_P}(\bar{f}; S) + \binom{m}{2}\frac{4BM}{\sqrt{n}} + \binom{m}{2}(m - 1 + 2BM)\sqrt{\frac{8\ln(2/\delta)}{n}}.$$

Finally, we can bound $R_{\ell_P}(\bar{f})$ by $R_{\ell_{0-1}}(\bar{f})$ from below since $R_{\ell_{0-1}}(\bar{f}) \leq R_{\ell_P}(\bar{f})$ follows from $\ell_{0-1} \leq \ell_P$, concluding the proof. □

Interestingly, the structure of the output space is directly reflected in the risk bounds. For full pairwise ranking over all possible pairs, a penalty of $\binom{m}{2}$ has to be paid for the exploration of the full space of all pairwise comparisons. For the case of pairwise ranking of documents at r relevance levels, including $|l_i|$ documents each, pairwise comparisons between documents at the same relevance level can be ignored. Thus, in this scenario of multipartite ranking, the number of pairs $|\mathcal{P}_q|$ is reduced from the set of all $\binom{m}{2}$ pairwise comparisons to $\sum_{i=1}^{r-1}\sum_{j=i+1}^{r}|l_i||l_j|$. A risk bound for this scenario is given by the following corollary:

Corollary 1. *Let ℓ_{0-1} be the 0-1 loss and ℓ_P be the pairwise hinge loss defined on a set of $\sum_{i=1}^{r-1}\sum_{j=i+1}^{r}|l_i||l_j|$ pairs over r relevance levels l_i. Let $S = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n$ be a training set of i.i.d. instances, and $\bar{\mathcal{F}}$ be the class of linear ranking-difference functions. Then with probability $1 - \delta$ over samples of length n, the following holds for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$R_{\ell_{0-1}}(\bar{f}) \leq \hat{R}_{\ell_P}(\bar{f}; S) + (\sum_{i=1}^{r-1}\sum_{j=i+1}^{r}|l_i||l_j|)\frac{4BM}{\sqrt{n}} + (\sum_{i=1}^{r-1}\sum_{j=i+1}^{r}|l_i||l_j|)(r - 1 + 2BM)\sqrt{\frac{8\ln(2/\delta)}{n}}.$$

Bipartite ranking of rel relevant and $nrel$ non-relevant documents involves even fewer pairs, namely $|\mathcal{P}_q| = rel \cdot nrel$. A risk bound for bipartite ranking can be given as follows:

Corollary 2. *Let ℓ_{0-1} be the 0-1 loss and ℓ_P be the pairwise hinge loss defined on a set of $rel \cdot nrel$ pairs for bipartite ranking of rel relevant and $nrel$ non-relevant documents. Let $S = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n$ be a training set of i.i.d. instances, and $\bar{\mathcal{F}}$ be the class of linear ranking-difference functions. Then with probability $1-\delta$ over samples of length n, the following holds for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$R_{\ell_{0-1}}(\bar{f}) \leq \hat{R}_{\ell_P}(\bar{f}; S) + (rel \cdot nrel)\frac{4BM}{\sqrt{n}} + (rel \cdot nrel)(1+2BM)\sqrt{\frac{8\ln(2/\delta)}{n}}.$$

For the general case of listwise hinge loss, we get the following, using a normalization constant $Z_L = m!(1+2BM)$ for a number of $|\Pi_q| = m!$ permutations over $m = n(q)$ ranks:

Proposition 2. *Let $\hat{\ell}_L = \frac{1}{Z_L}\ell_L$ be the normalized listwise hinge loss. Then for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$\mathcal{R}_n(\hat{\ell}_L \circ \bar{\mathcal{F}}) \leq \frac{2}{1+2BM}\mathcal{R}_n(\bar{\mathcal{F}}).$$

Proof. Follows directly from Theorem (2) with $L_{\hat{\ell}_L} = \sup_{\bar{f}} |\hat{\ell}'_L(\bar{f})| = |\frac{m!}{m!(1+2BM)}|$. □

A risk bound for listwise prediction loss in the general case can be given as follows.

Theorem 4. *Let ℓ_L be the listwise hinge loss defined in Definition (3). Let $S = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n$ be a training set of i.i.d. instances, and $\bar{\mathcal{F}}$ be the class of linear ranking-difference functions. Then with probability $1-\delta$ over samples of length n, the following holds for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$R_L \leq \hat{R}_{\ell_L}(\bar{f}; S) + m!\frac{4BM}{\sqrt{n}} + m!(1+2BM)\sqrt{\frac{8\ln(2/\delta)}{n}}.$$

Proof. Similar to the proof for (3) using the fact that the hinge loss ℓ_L bounds the prediction loss L from above (see [11], Proposition 2), where $R_L = \int_Q L(y_q, \pi_q)P(dy_q, d\pi_q)$. □

Specific prediction loss functions such as AP define a specific structure on the output space which is reflected in the risk bound for structured prediction for AP loss. For AP, permutations that involve only reorderings of relevant documents with relevant documents, or reorderings of irrelevant documents with irrelevant documents, are considered equal. This means that instead of $m!$ permutations over a list of size $m = n(q)$, the number of permutations is $|\Pi_q| = \frac{m!}{rel!nrel!} = \binom{m}{rel} = \binom{m}{nrel}$, where rel and $nrel$ are the number of relevant and irrelevant documents. A risk bound for listwise ranking using AP loss can be given as follows:

Corollary 3. *Let L_{AP} be the AP loss defined Definition 4 and $\ell_{L_{AP}}$ be the listwise hinge loss using L_{AP} as prediction loss function. Let $S = \{(x_q^{(i)}, y_q^{(i)})\}_{i=1}^n$ be a training set of i.i.d. instances, and $\bar{\mathcal{F}}$ be the class of linear ranking-difference functions. Then with probability $1-\delta$ over samples of length n, the following holds for all $\bar{f} \in \bar{\mathcal{F}}$:*

$$R_{L_{AP}} \leq \hat{R}_{\ell_{L_{AP}}}(\bar{f}; S) + \binom{m}{rel}\frac{4BM}{\sqrt{n}} + \binom{m}{rel}(1+2BM)\sqrt{\frac{8\ln(2/\delta)}{n}}.$$

5 Discussion

The bounds we presented were given for algorithms that compute the hinge loss by summing over all possible outputs instead of taking the arg-max to find the most violated constraint. Since $\sum x_i \geq \max_i x_i$, for all $x_i \geq 0$, the bounds still apply to approaches that take the arg-max. On the other hand, they also apply to approaches where successively adding most violated constraints is infeasible [12]. Tighter bounds may be given for arg-max and soft-max versions. This is due to future work. Furthermore, the proofs need to be extended to other listwise loss functions such as NDCG. Lastly, an empirical validation supporting the theoretical results needs to be given.

Acknowledgements

We would like to thank Olivier Bousquet for several discussions of the work presented in this paper.

References

[1] Peter L. Bartlett and Sahar Mendelson. Rademacher and gaussian complexity: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[2] Yanyan Lan, Tie-Yan Liu, Zhiming Ma, and Hang Li. Generalization analysis of listwise learning-to-rank algorithms. In *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*, Montreal, Canada, 2009.

[3] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems (TOIS)*, 25(2), 2007.

[4] Shivani Agarwal and Partha Niyogi. Generalization bounds for ranking algorithms via algorithmic stability. *Journal of Machine Learning Research*, 10:441–474, 2009.

[5] David McAllester. Generalization bounds and consistency for structured labeling. In Gökhan Bakhir, Thomas Hofmann, and Berhard Schölkopf, editors, *Prediction Structured Data*. The MIT Press, Cambridge, MA, 2007.

[6] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, Amsterdam, The Netherlands, 2007.

[7] Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharayya. Structured learning for non-smooth ranking losses. In *Proceedings of the 14th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'08)*, Las Vegas, NV, 2008.

[8] Corinna Cortes, Mehryar Mohri, and Asish Rastogi. Magnitude-preserving ranking algorithms. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, Corvallis, OR, 2007.

[9] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, NY, 1999.

[10] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions in Information Systems*, 20(4):422–446, 2002.

[11] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 5:1453–1484, 2005.

[12] Thomas Gärtner and Shankar Vembu. On structured output training: hard cases and an efficient alternative. *Jounal of Machine Learning Research*, 76:227–242, 2009.

Spectral Analysis for Phylogenetic Trees

Paige Rinker
Department of Mathematics
Dartmouth College
Hanover, NH 03755
paige.rinker@dartmouth.edu

Daniel N. Rockmore
Department of Mathematics
Dartmouth College
Hanover, NH 03755
rockmore@math.dartmouth.edu

Abstract

There is a natural bijection between phylogenetic trees on $n+1$ taxa and perfect matchings of $2n$ objects. Using this bijection we derive a symmetric group-based spectral analysis for functions on the space of $n+1$ taxa treating them as a quotient space of ranked data (i.e., the symmetric group S_{2n}). This has applications to the analysis of fitness functions on phylogenetic trees.

1 Introduction

A phylogenetic tree on $n+1$ "taxa" (distinct species) is a graphical depiction of an evolutionary process resulting in $n+1$ distinct species hailing from a common ancestor. Mathematically, a phylogenetic tree is a binary rooted tree with $n+1$ labeled leaves (see Figure 1c). Since the actual evolutionary relationships between a set of related species is unknown, finding the "true" phylogenetic tree is difficult. A nice overview of the subject can be found here [E]. There are several algorithms for determining the most suitable tree for a given set of taxa (see e.g., [E]), including the adaptation of standard (i.e., abelian) spectral analysis for the modeling, prediction, and general analysis of the evolution of genetic sequences [HPS]. At some level, these methods make use of functions which measure fitness relative to genetic data.

In this paper, we use a known bijection between these trees and pairings to introduce a different kind of spectral analysis, one derived from the spectral analysis of data on rankings, for such fitness functions, or any kind of function defined on the space of phylogenetic trees on a given number of taxa. In particular, we build on the work of Diaconis and Holmes [DH98] wherein a bijection between phylogenetic trees (rooted binary trees with $n+1$ labeled leaves) and pairings of the numbers $\{1, 2, 3, \ldots 2n\}$ is given. This correspondence gives rise to an action of the symmetric group $\mathcal{S}_{2n}$ on the space of trees, $\mathcal{X}_n$, and with it, a map by which the spectral analysis (i.e., Fourier decomposition) of matchings (viewed as homogeneous space for S_{2n}) can be interpreted as a spectral analysis for $\mathcal{X}_n$ (see [D]). [1]

2 A Bijection Between Pairings and Phylogenetic Trees

A binary tree with $n+1$ leaves has $2n$ interior nodes. A labeled binary tree on $n+1$ leaves and a pairing on $\{1, 2, \ldots, 2n\}$ are associated in the following way (see [DH98]): (1) Label the leaves of the tree from the set $\{1, 2, \ldots n, n+1\}$ (without repetition); (2) Label the parents in the tree consecutively, at each step labeling the parent with the youngest labeled pair of children; (3) Once the tree is labeled, the associated perfect matching (pairing) is as follows: the labels of each sibling pair become a match in the corresponding perfect matching. In the example depicted in Figure 1c, the sibling pairs are {1 3}, {2 5}, {6 7}, and {4 8}.

Conversely, starting with a pairing, we build the unique tree fitting the labeling rules above as follows. Among the pairs, locate the youngest pair $\{a\ b\}$ where $a, b \in \{1, 2, \ldots, n+1\}$ (youngest meaning the pair with the smallest child). Draw two nodes of the tree, label them with a and b, and label their parent with

[1] The use of $\mathcal{X}_n$ is in place of Diaconis & Holmes' use of $\mathcal{M}_n$ in [DH02].

$n+2$. Add the next youngest sibling pair of leaves, if one exists, and label its parent with the next consecutive label. If one doesn't exist, then add a node and label for the sibling of $n+2$. Continue in this fashion, at each step labeling the parent of the youngest available sibling pair with the next consecutive label. Figure 1 outlines the construction of the phylogenetic tree on five taxa from the pairing {1 3}{2 5}{6 7}{4 8}.

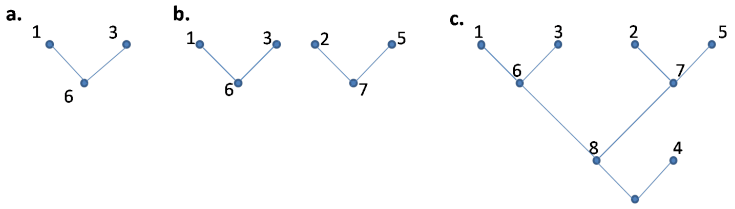


Figure 1: The construction of the phylogenetic tree corresponding to the pairing {1 3}{2 5}{6 7}{4 8}.

3 Spectral Analysis on Pairings

Let $\mathcal{X}_n$ be the set of pairings of $\{1, 2, 3, \dots 2n\}$. Then $\mathcal{S}_{2n}$ acts transitively on $\mathcal{X}_n$, and this action gives a permutation representation of $\mathcal{S}_{2n}$ on the space of functions on $\mathcal{X}_n$, $\mathrm{L}(\mathcal{X}_n)$. Since the action is transitive, we can think of the set of pairings as the quotient of $\mathcal{S}_{2n}$ by the stabilizer subgroup of any pairing $x \in \mathcal{X}_n$, and this subgroup is isomorphic to the wreath product of $\mathcal{S}_2$ by $\mathcal{S}_n$: $B_n = \mathcal{S}_2 \wr \mathcal{S}_n$ [DH02]. So we have $\mathcal{X}_n \cong \mathcal{S}_{2n}/B_n$, as sets. The representation of S_{2n} is multiplicity-free (i.e., $(\mathcal{S}_{2n}, B_n)$ is a Gelfand pair) and with irreducible decomposition (see [DH02], [IGM])

$$L(\mathcal{X}_n) \cong \bigoplus_{\lambda \vdash n} \mathcal{S}^{2\lambda}.$$

Thus, the zonal spherical functions of this Gelfand pair are indexed by the partitions of n, and are given by the following:

$$\omega^{\lambda}(s) = \frac{1}{|B_n|} \sum_{b \in B_n} \chi^{2\lambda}(sb)$$

where $\chi^{2\lambda}$ is the character of $\mathcal{S}^{2\lambda}$ and $s \in \mathcal{S}_{2n}$. Projection onto each of these subspaces, $\mathcal{S}^{2\lambda}$, gives a general decomposition of functions on pairings. Viewing these subspaces as sitting inside corresponding spaces of data on rankings gives natural meaning to each of these projections.

4 Decomposing Functions on Pairings

Let $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k)$ be a partition of n. Then define $\mathcal{X}_n^{\lambda} = \mathcal{S}_{2n}/(\mathcal{S}_{2\lambda_1} \times \mathcal{S}_{2\lambda_2} \times \cdots \times \mathcal{S}_{2\lambda_k})$, and $M^{2\lambda} = L(\mathcal{X}_n^{\lambda})$ is the space of functions on $\mathcal{X}_n^{\lambda}$. In words, $\mathcal{X}_n^{\lambda}$ is the set of all the ways to partition the set $\{1, 2, \dots, 2n\}$ into ordered subsets of sizes $2\lambda_1,\ 2\lambda_2, \dots, 2\lambda_k$ (to avoid overusing the word "partition," we shall refer to the elements of $\mathcal{X}_n^{\lambda}$ as *groupings*). Then $M^{2\lambda}$ is the space of functions whose values depend on how the numbers $\{1, 2, \dots 2n\}$ are grouped, and how these groupings are ranked [D].When $\mu = (1, 1, \dots, 1)$, we have the space $\mathcal{X}_n^{\mu} = \mathcal{S}_{2n}/(\mathcal{S}_2)^n$. While this is distinct from $\mathcal{X}_n$, any function $f \in L(\mathcal{X}_n)$ can be viewed as a function in $L(\mathcal{X}_n^{\mu})$ which just happens to take the same value for every possible ranking (ordering) of the same pairs. The projection map defined below has exactly this effect.

We define a family of maps taking $L(\mathcal{X}_n)$ into the spaces $M^{2\lambda}$ by the following averaging rule:

$$\begin{aligned} R^{\lambda} : L(\mathcal{X}_n) &\rightarrow L(\mathcal{X}_n^{\lambda}) = M^{2\lambda} \\ R^{\lambda} f(\bar{x}) &= \sum_{x \prec \bar{x}} f(x), \end{aligned}$$

where $x \in \mathcal{X}_n$, $\bar{x} \in \mathcal{X}_n^{\lambda}$ and $x \prec \bar{x}$ simply means that the numbers paired together in x are grouped together in $\bar{x}$.

In general, M^λ decomposes as a sum of $\mathcal{S}^\mu$'s:

$$M^\lambda = \bigoplus_{\mu \geq \lambda} m_{\mu\lambda} \mathcal{S}^\mu,$$

for some coefficients $m_{\mu\lambda}$ [S]. Since we know that $L(\mathcal{X}_n)$ is made up only of the $\mathcal{S}^{2\mu}$'s, the images $R^\lambda(L(\mathcal{X}_n)) \subset M^{2\lambda}$ must only intersect the pieces of $M^{2\lambda}$ of the form $\mathcal{S}^{2\mu}$. So, R^λ is really just a projection of $L(\mathcal{X}_n)$ onto the subspaces $\mathcal{S}^{2\mu} \subset L(\mathcal{X}_n)$ which happen to appear in $M^{2\lambda}$ (i.e., $m_{(2\mu)(2\lambda)} \neq 0$).

Now we can use these projections to determine how dependent a function is on a particular grouping.

5 Back to Trees

To answer the question of what this decomposition means in the setting of fitness functions, we must recall that the bijection labels the leaves of the tree with $1, 2, \ldots, n+1$ and interior nodes with $n+2, \ldots, 2n$ in such a way that paired numbers go to siblings (nodes with the same parent). So, if a function is heavily concentrated on a certain grouping, this means that it is very important to the fitness of the tree that the grouped labels appear as siblings in some way. Examining finer groupings can help determine whether the way things are paired within a grouping makes a difference to the fitness of the tree. There are a number of approaches to the efficient calculation of these projections (see e.g., [MOR]).

To illustrate our meaning, consider the following example when $n = 3$. There are 15 pairings of $\{1, 2, \ldots, 6\}$, since $\mathcal{X}_3 = \mathcal{S}_6/B_6$ has $|\mathcal{X}_3| = |\mathcal{S}_6|/|B_6| = 6!/(3! \cdot 2^3) = 15$. The space $L(\mathcal{X}_3)$ decomposes as: $L(\mathcal{X}_3) \cong \mathcal{S}^6 \oplus \mathcal{S}^{4,2} \oplus \mathcal{S}^{2,2,2}$.

We need to compute the projection of $L(\mathcal{X}_3)$ onto its intersection with each of M^6, $M^{4,2}$ and $M^{2,2,2}$, and interpret these projections for functions on $\mathcal{X}_3$. We shall see that in the case of $n = 3$, only one of these computations is truly interesting.

The full decomposition of the M^λ representations of $\mathcal{S}_6$ in terms of the irreducible representations $\mathcal{S}^{\lambda_i}$ is given in Figure 2.

$$\begin{aligned} M^6 &= \mathcal{S}^6 \\ M^{4,2} &= \mathcal{S}^6 \oplus \mathcal{S}^{5,1} \oplus \mathcal{S}^{4,2} \\ M^{2,2,2} &= \mathcal{S}^6 \oplus 2\mathcal{S}^{5,1} \oplus 3\mathcal{S}^{4,2} \oplus \mathcal{S}^{4,1,1} \oplus \mathcal{S}^{3,3} \oplus 2\mathcal{S}^{3,2,1} \oplus \mathcal{S}^{2,2,2} \end{aligned}$$

First, let's consider $\lambda_1 = (3) \vdash 3$. Then $\mathcal{X}_3^3 = \mathcal{S}_6/\mathcal{S}_6 \cong \{1\}$, and for $f \in L(\mathcal{X}_n)$,

$$R^6(f)(\bar{x}) = \sum_{x \in \mathcal{X}_n} f(x),$$

for $\bar{x} = \{1\ 2\ 3\ 4\ 5\ 6\} = \mathcal{X}_3^3$. Since there is only one element of $\mathcal{X}_n^n$, this projection gives very little information. But it does allow us to ignore this piece of higher dimensional projections.

On the other end, we may consider $\lambda_8 = (1,1,1) \vdash 3$. Then $\mathcal{X}_3^{1,1,1} = \mathcal{S}_6/(\mathcal{S}_2 \times \mathcal{S}_2 \times \mathcal{S}_2)$ and $M^{2,2,2} = L(\mathcal{X}_3^{1,1,1})$ is the space of functions which depend on how the set $\{1, 2, \ldots, 6\}$ is grouped into a set of three ranked/ordered pairs. However, since for any $\bar{x} \in \mathcal{X}_3^{2,2,2}$, there is a unique $x \in \mathcal{X}_3$ with $x \prec \bar{x}$, $R^{2,2,2}(f)(\bar{x}) = f(x)$. Again, this gives us very little new information.

While neither of these gives any particularly interesting information, $\lambda_3 = (2,1) \vdash 3$ will prove to be more useful. In this case, we have $\mathcal{X}_3^{2,1} = \mathcal{S}_6/(\mathcal{S}_4 \times \mathcal{S}_2)$ and $M^{4,2} = L(\mathcal{X}_3^{2,1})$ is the space of functions which depend on a distinguished pair from the set $\{1, 2, \ldots, 6\}$.

Suppose we knew that the tree best fitting a particular set of taxa corresponded to the pairing $\{1\ 2\}\{3\ 4\}\{5\ 6\}$, then we can use this projection to determine whether one of these pairs is more important to the fitness of the tree than the others. Let $\bar{x}_1 = \{1\ 2\}\{3\ 4\ 5\ 6\}$, $\bar{x}_2 = \{3\ 4\}\{1\ 2\ 5\ 6\}$, $\bar{x}_3 = \{5\ 6\}\{1\ 2\ 3\ 4\} \in \mathcal{X}_3^{\{2,1\}}$. Then comparing the computations below gives us an idea of whether one of these distinguished pairs is more

important to the fitness of the tree than the others:

$$R^{2,1}(f)(\bar{x}_1) = \sum_{x \prec \bar{x}_1} f(x) = f(\{1\ 2\}\{3\ 4\}\{5\ 6\}) + f(\{1\ 2\}\{3\ 5\}\{4\ 6\}) + f(\{1\ 2\}\{3\ 6\}\{4\ 5\})$$
$$R^{2,1}(f)(\bar{x}_2) = \sum_{x \prec \bar{x}_2} f(x) = f(\{3\ 4\}\{1\ 2\}\{5\ 6\}) + f(\{3\ 4\}\{1\ 5\}\{2\ 6\}) + f(\{3\ 4\}\{1\ 6\}\{2\ 5\})$$
$$R^{2,1}(f)(\bar{x}_3) = \sum_{x \prec \bar{x}_3} f(x) = f(\{5\ 6\}\{1\ 2\}\{3\ 4\}) + f(\{5\ 6\}\{1\ 3\}\{2\ 4\}) + f(\{5\ 6\}\{1\ 4\}\{2\ 3\}).$$

For larger n, there will naturally be more than one interesting projection. By considering each of these projections, we can systematically determine the relative importance of each distinguished sibling pair, or other grouping of labels.

	$dim(M^{\lambda_i})$	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	$dim(S^{\lambda_i})$
$\lambda_1 = (6)$	1	1	0	0	0	0	0	0	0	0	0	0	1
$\lambda_2 = (5,1)$	6	1	1	0	0	0	0	0	0	0	0	0	5
$\lambda_3 = (4,2)$	15	1	1	1	0	0	0	0	0	0	0	0	9
$\lambda_4 = (4,1^2)$	30	1	2	1	1	0	0	0	0	0	0	0	10
$\lambda_5 = (3^2)$	20	1	1	1	0	1	0	0	0	0	0	0	5
$\lambda_6 = (3,2,1)$	60	1	2	2	1	1	1	0	0	0	0	0	16
$\lambda_7 = (3,1^3)$	120	1	3	3	3	1	2	1	0	0	0	0	10
$\lambda_8 = (2^3)$	90	1	2	3	1	1	2	0	1	0	0	0	5
$\lambda_9 = (2^2,1^2)$	180	1	3	4	3	2	4	1	1	1	0	0	9
$\lambda_{10} = (2,1^4)$	360	1	4	6	6	3	8	4	2	3	1	0	5
$\lambda_{11} = (1^6)$	720	1	5	9	10	5	16	10	5	9	5	1	1

Figure 2: The decomposition of the M^λ representations of $\mathcal{S}_6$.

Acknowledgments

This work was partially supported by AFOSR award DOD Grant #FA9550-06-1-0027 (D.R.) and NSF award DMS-0811005 (P.R.).

References

[D] P. Diaconis (1988), *Group Representations in Probability and Statistics*, Institute of Mathematical Statistics Lecture Notes, Monograph Series, vol. 11, Institute of Mathematical Statistics, Hayward, CA.

[DH98] P. Diaconis and S. Holmes (1998), Matchings and phylogenetic trees, *PNAS*, **95**, 14600–14602.

[DH02] P. Diaconis and S. Holmes (2002), Random walks on trees and matchings, *Electronic Journal of Probability*, **7** no. 6, 1–17.

[E] S. N. Evans (2003), Fourier analysis and phylogenetic trees, *Modern Signal Processing, MSRI Publications* vol. 46, 117–136.

[HPS] M. D. Hendy, D. Penny, and M. A. Steel (1994), A discrete Fourier analysis for evolutionary trees, *PNAS* **91**, 3339–3343.

[IGM] I. G. MacDonald (1995), *Symmetric Functions and Hall Polynomials(2nd ed.)*, Oxford University Press, Oxford.

[MOR] D. K. Maslen, M. E. Orrison, and D. N. Rockmore (2004), Computing isotypic projections with the Lanczos iteration, *SIAM J. Matrix Anal. Appl.* **25**, no. 3, pp. 784–803.

[M] F. A. Matsen (2009), Fourier transform inequalities for phylogenetic trees, *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, **6** no. 1, 89–95.

[Sa] B. Sagan (2001), *The Symmetric Group (2nd ed.)*, Graduate Texts in Mathematics, vol. 203, Springer-Verlag, New York.

[SSE] L. A. Székely, M. A. and Steel, and P. Erdős (1993), Fourier calculus on evolutionary trees, *Adv. in Appl. Math.* **14** no. 2, 200–216.

Large Scale Learning to Rank

D. Sculley
Google, Inc.
dsculley@google.com

Abstract

Pairwise learning to rank methods such as RankSVM give good performance, but suffer from the computational burden of optimizing an objective defined over $O(n^2)$ possible pairs for data sets with n examples. In this paper, we remove this super-linear dependence on training set size by sampling pairs from an implicit pairwise expansion and applying efficient stochastic gradient descent learners for approximate SVMs. Results show orders-of-magnitude reduction in training time with no observable loss in ranking performance. Source code is freely available at: http://code.google.com/p/sofia-ml

1 Introduction: The Problem with Pairs

In this paper, we are concerned with learning to rank methods that can learn on large scale data sets. One standard method for learning to rank involves optimizing over the set of pairwise preferences implicit in the training data. However, in the worst case there are $\binom{n}{2}$ candidate pairs, creating the potential for a quadratic dependency on the size of the data set. This problem can be reduced by sharding the data (*e.g.* by query) and restricting pairwise preferences to be valid only within a given shard, but this still results in super-linear dependencies on n. Even with efficient data structures for special cases this leads to $O(n \log n)$ computation cost for training [8]. Applying such methods to internet-scale data is problematic.

Bottou *et al*. have observed that in large-scale settings the use of stochastic gradient descent methods provide extremely efficient training with strong generalization performance – despite the fact that stochastic gradient descent is a relatively poor optimization strategy [2, 1]. Thus, the main approach of this paper is to adapt the pairwise learning to rank problem into the stochastic gradient descent framework, resulting in a scalable methodology that we refer to as Stochastic Pairwise Descent (SPD). Our results show that best performing SPD methods provide state of the art results using only a fraction of a second of CPU time for training.

2 Problem Statement: Pairwise Learning to Rank

The data sets D studied in this paper are composed of labeled examples $(\mathbf{x}, y, q)$ with each $\mathbf{x} \in \mathbb{R}^n$ showing the location of this example in n-dimensional space, each $y \in \mathbb{N}$ denoting its *rank*, and each $q \in \mathbb{Z}$ identifying the particular query/shard to which this example belongs. We define the set P of *candidate pairs* implied by D as the set of all tuple-pairs $((\mathbf{a}, y_a, q_a), (\mathbf{b}, y_b, q_b))$ with $y_a \neq y_b$ and $q_a = q_b$. When $y_a > y_b$, we say that $\mathbf{a}$ is *preferred over* $\mathbf{b}$ or *ranked better than* $\mathbf{b}$. Note that in the worse case $|P|$ grows quadratically with $|D|$.

For this paper, we restrict ourselves to solving the classic RankSVM optimization problem, first posed by Joachims [7].[1] In our notation, we seek to minimize:

$$\min \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{|P|}\left(\sum_{((\mathbf{a},y_a,q_a),(\mathbf{b},y_b,q_b))\in P} \text{HingeLoss}((\mathbf{a}-\mathbf{b}), \text{sign}(y_a - y_b), \mathbf{w})\right)$$

That is, we seek a weight vector $\mathbf{w} \in \mathbb{R}^n$ that gives low hinge-loss over the set candidate pairs P and also has low complexity. Here, λ is a regularization parameter and the function sign(p) returns +1 for $p > 0$, -1 for $p < 0$. Hinge loss is computed as $\max(0, 1 - \langle \mathbf{w}, \text{sign}(y_a - y_b)(\mathbf{a} - \mathbf{b})\rangle)$.

Given $\mathbf{w}$, prediction scores are made for unseen $\mathbf{x}$ by $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x}\rangle$, and predicted rankings are inferred by sorted score.

Observe that in the special case of binary rank values of `relevant` and `not-relevant` in a single query shard, this optimization problem is equivalent to the problem of optimizing area under the ROC curve for binary-class data.

3 A Stochastic Pairwise Strategy

Bottou *et al.* point out that the generalization ability of stochastic gradient descent relies only on the number of stochastic steps taken, not the size of the data set [1]. Thus, our main idea is to sample candidate pairs from P for stochastic steps, without constructing P explicitly. This avoids dependence on $|P|$.

Algorithm 1 gives the generic framework, reducing learning to rank to learning a binary classifier via stochastic gradient descent. This reduction preserves the convergence properties of stochastic gradient descent.

Care is needed to ensure that we are sampling from P and not some other pairwise expansion of the data. We propose two implementations of `GetRandomPair`, below, including a method that samples efficiently from P using and index of D. Methods for `StochasticGradientStep` appear in Section 3.2.

Algorithm 1 Stochastic Pairwise Descent (SPD). This generic framework reduces the pairwise learning to rank problem to the learning a binary classifier via stochastic gradient descent. The CreateIndex and GetRandomPair functions, instantiated below, enable efficient sampling from P.

1: $D_{index} \leftarrow$ CreateIndex(D)
2: $\mathbf{w_0} \leftarrow \oslash$
3: **for** $i = 1$ to t **do**
4: $\quad((\mathbf{a}, y_a, q), (\mathbf{b}, y_b, q)) \leftarrow$ GetRandomPair(D_{index})
5: $\quad\mathbf{x} \leftarrow (\mathbf{a} - \mathbf{b})$
6: $\quad y \leftarrow$ sign($y_a - y_b$)
7: $\quad\mathbf{w}_i \leftarrow$ StochasticGradientStep($\mathbf{w}_{i-1}, \mathbf{x}, y, i$)
8: **end for**
9: **return** $\mathbf{w}_t$

3.1 Sampling Methods

Here, we instantiate the `GetRandomPair` method from the SPD algorithm, above.

Samping Without an Index It is important to be able to sample from P without explicitly indexing D when D does not fit in main memory or is unbounded, as with streaming data. This may be done by repeatedly selecting two examples $(\mathbf{a}, y_a, q_a)$ and $(\mathbf{b}, y_b, q_b)$ from the data stream until a pair is found such that $(y_a \neq y_b)$ and $(q_a = q_b)$. The expected number of rejected pairs per call is $O(\frac{|D|^2}{|P|})$.

[1] Using approximate SVM solvers such as ROMMA and the Passive-Aggressive Perceptron results in optimizing slight variants of this problem.

Indexed Sampling In our preliminary tests, we found that the rejection sampling method above was surprisingly efficient. However, we found that indexing gave faster sampling when D fit in memory. This index is constructed as follows. Let Q be the set of unique values q appearing in D, and $Y[q]$ map from values of $q \in Q$ to the set of unique y values appearing in examples $(\mathbf{x}, y, q') \in D$ with $q = q'$. Finally, let $P[q'][y']$ map from values $q \in Q$ and $y \in Y[q]$ to the set of examples $(\mathbf{x}, y', q') \in D$ for which $q = q'$ and $y = y'$. This index can be constructed in linear time using nested hash tables.

To sample from P in constant time using this index:

select q uniformly at random from Q
select y_a uniformly at random from $Y[q]$
select y_b uniformly at random from $Y[q] - y_a$.
select $(\mathbf{a}, y_a, q)$ uniformly at random from $P[q][y_a]$
select $(\mathbf{b}, y_b, q)$ uniformly at random from $P[q][y_b]$
return $((\mathbf{a}, y_a, q), (\mathbf{b}, y_b, q))$

Note that this method samples uniformly from P only when the query shards are of equal size, and the number of examples per unique y value are constant per shard. When these conditions are not met, sampling from P requires $O(\log |Q| + \log |Y[q]_{max}|)$ computation in the general case. This more general algorithm is omitted for lack of space, and because the constant time algorithm gives excellent results in practice. The quality of these results are due in part to the fact that the benchmark data sets have query shards that tend to be consistent in size. Furthermore, Cao *et al*. argue that this form of query-shard normalization is actually beneficial [4], because it ensures that very large query-shards do not dominate the optimization problem.

3.2 Stochastic Gradient Descent Methods

Because we are interested in the RankSVM optimization problem in this paper, the stochastic gradient descent methods we examine are all based on hinge-loss (also sometimes called SVM-loss) and are more properly referred to as stochastic sub-gradient descent methods. We review these variants briefly by noting how they perform updates of the weight vector $\mathbf{w}$ on each step.

SGD SVM The simple stochastic sub-gradient descent SVM solver updates only when the hinge-loss for an example $(\mathbf{x}, y)$ is non-zero [14], using the rule: $\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x} - \eta \lambda \mathbf{w}$. We set the learning rate η for using the Pegasos schedule $\eta_i \leftarrow \frac{1}{\lambda i}$ on step i [13]. (We also experimented with a more traditional update rule $\eta \leftarrow \frac{c}{c+i}$ finding no improvements.)

Pegasos (SVM) Pegasos is an SVM solver that adds a hard constraint: $||\mathbf{w}|| \leq \frac{1}{\sqrt{\lambda}}$. Pegasos takes the same sub-gradient step as SGD SVM, but then projects $\mathbf{w}$ back into the L2-ball of radius $\frac{1}{\sqrt{\lambda}}$, which gives theoretical guarnatees for fast convergence [13].

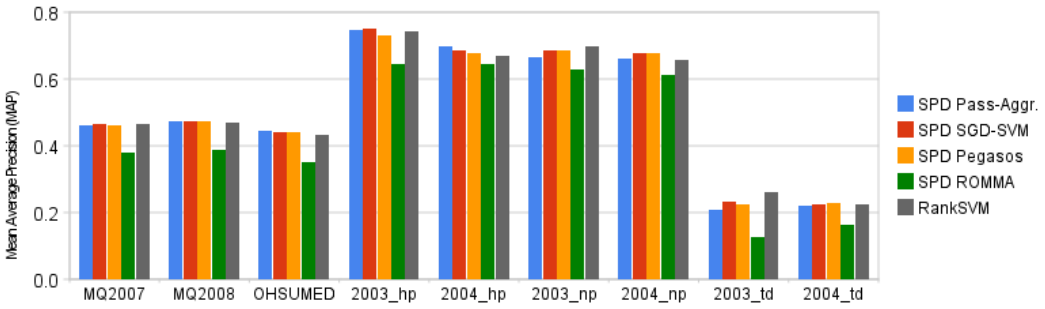
Passive-Aggressive Perceptron The PA-I algorithm [5] takes variable-size steps to minimize hinge loss and uses a regularization parameter C to control complexity, similar to SVM in several respects. The update rule is $\mathbf{w} \leftarrow \mathbf{w} + \tau y \mathbf{x}$ where $\tau \leftarrow \min(C, \frac{\text{HingeLoss}(\mathbf{x}, y, \mathbf{q})}{||\mathbf{x}||^2})$.

ROMMA The ROMMA algorithm attempts to maintain a maximum-margin hypothesis; we apply the aggressive variant that updates on any example with positive hinge loss [11]. The update rule is $\mathbf{w} \leftarrow c\mathbf{w} + d\mathbf{x}$, where $c \leftarrow \frac{||\mathbf{x}||^2||\mathbf{w}||^2 - y(\langle \mathbf{x}, \mathbf{w} \rangle)}{||\mathbf{x}||^2||\mathbf{w}||^2 - \langle \mathbf{x}, \mathbf{w} \rangle^2}$ and $d \leftarrow \frac{||\mathbf{w}||^2(y - \langle \mathbf{x}, \mathbf{w} \rangle)}{||\mathbf{x}||^2||\mathbf{w}||^2 - \langle \mathbf{x}, \mathbf{w} \rangle^2}$.

3.3 Related Methods

Joachims gave the first RankSVM optimization problem, solved with SVM-light [7], and later gave a faster solver using a plane-cutting algorithm [9] referred to here as SVM-struct-rank. Burgess *et al.* used gradient descent for ranking for non-linear neural networks and applied a probabilistic pairwise cost function [3]. Elsas *et al.* adapted voted Perceptron for ranking, but their method maintained a quadratic dependence on $|D|$ [6].

Figure 1: **Mean Average Precision (MAP) for LETOR Benchmarks.** SPD Pegasos, SPD Passive-Aggressive and SPD SGD-SVM all give results statistically equivalent to RankSVM with far less cost.



3.4 Things That Did Not Help

Neither Perceptron nor Perceptron with Margins were competitive methods. Expanding feature vectors $\mathbf{x} \in \mathbb{R}^n$ into cross-product feature-vectors $\mathbf{x}' \in \mathbb{R}^{n \times n}$ gave no improvement in preliminary trials. Our efforts to include pairs with ties (where $y_a = y_b$) yielded no additional benefit, agreeing with a similar finding in [3].

4 LETOR Experiments

Experimental Setup We first wished to compare the ranking performance of the SPD methods with RankSVM. To do so, we applied each SPD method on the LETOR 3.0 and LETOR 4.0 benchmark data sets for supervised learning to rank [12], using the standard LETOR procedures for tuning, training, and testing. We used 100,000 iterations for each SPD method; this number of iterations was picked during initial tests and held constant for all methods. The RankSVM comparison results are previously published benchmark results on these tasks [12].

Ranking Performance The Mean Average Precision (MAP) results for each LETOR task are given in Figure 1. To our surprise, we found not only that SPD Pegasos was statistically indistinguishable from RankSVM, but also that SPD SGD-SVM and SPD Passive-Aggressive were equivalent to RankSVM as well. These results are extremely encouraging; a range of simple SPD methods perform as well as RankSVM on these standard benchmarks. SPD ROMMA was not competitive.

Training Speed All experiments were run on a dual-core 2.8GHz laptop with 2G RAM with negligible additional load. The SPD methods each averaged between 0.2 and 0.3 CPU seconds per task for training time, which includes the construction of the sampling index and 100,000 stochastic steps. This was up to 100 times faster than the SVM-light implementation [7] of RankSVM, and up to 5 times faster than the SVM-struct-rank implementation [9]. However, this small scale setting of at most 70,000 examples was not large enough to fully highlight the scalability of the SPD methods.

Table 1: **Large-Scale Experimental Results**. Results for ROC area and CPUs for training time are given for the E311 task in the RCV1 data set.

LEARNER	ROC AREA	CPUS TRAINING
SVM-struct-rank	0.9992	30,716.33s
SVM-perf-roc	0.9992	31.92s
SPD Pass-Aggr	0.9990	.22s
SPD SGD-SVM	0.9992	.20s
SPD Pegasos	0.9992	.20s

5 Large Scale Experiments

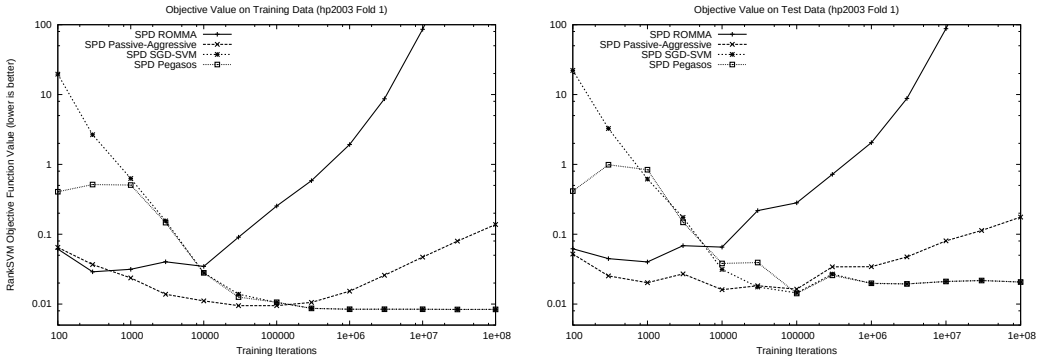
Experimental Setup To assess the ability of the SPD to perform at scale, we needed a larger ranking task than the LETOR benchmark data sets provided. Thus, we ran experiments on the much larger RCV1 data set for text categorization [10], treating the goal of optimizing ROC area (ROCA) as a large-scale ranking problem. (Recall that optimizing ROCA is equivalent to solving a ranking problem with binary ranks and a single query shard.)

The training set we used had 781,265 examples, with 23,149 examples in the test set. For space, we report results only the E311 topic, a minority class topic with a 0.2% positive class distribution. (Results on 20 other RCV1 minority-class topics from this data set are similar; omitted for space.) Parameters were tuned by cross-validation on the training set alone.

Scalability Results The results in Table 1 show that SPD methods were not affected by this increase in scale, using no more CPU time on this task than on the smaller LETOR tasks. SPD methods were more than 150 times faster than the SVM-perf-roc implementation of RankSVM tuned to ROCA optimization [8], and were many orders of magnitude faster than SVM-struct-rank (which not is optimized for large query shards). The ROCA performance of all ranking methods exceeded that of a binary-class SVM with tuned class-specific regularization parameters.

These results highlight the ability of simple SPD methods to give good ranking performance on large scale data sets with low computation cost for learning.

Figure 2: **Convergence Results.** These graphs shows the value of the RankSVM objective function for each SPD method on the training data (left) and on the test data (right).



6 Convergence Trials

After these results were digested, we went back to examine how quickly each SPD learner converged to a stable value of the RankSVM objective function. Because each of the SPD methods was intended to solve or approximate the RankSVM objective function given in Section 2, we used the value of this objective function as our test for convergence for all learners. (However, we keep in mind that SPD ROMMA and SPD Passive-Aggressive are not intended to solve this exact RankSVM problem.) Objective function values were observed at increasing intervals from 10^2 iterations (a small fraction of a second in CPUs for training) to 10^8 iterations (up to 500 CPUs for training). We performed these tests on the `hp_2003` task of the LETOR data set, using the train/test split given in Fold 1 of the cross-validation folds.

The results, given in Figure 2, show several interesting trends. Both SPD SGD-SVM and SPD Pegasos converge to stable, apparently optimal values on training data within 10^6 iterations. Continued training does not materially degrade performance on the test objective values. In contrast, both SPD Passive-Aggressive and SPD ROMMA give much better objective values than the other methods at small numbers of iterations (less than 10^5). However, their performance does degrade with additional training, with SPD ROMMA degrading especially rapidly. We attribute this to the fact that unlike SPD Pegasos and SPD SGD-SVM these methods do not optimize the RankSVM objective

function directly, and to the fact that these methods can take large update steps on noisy examples regardless of the number of past iterations.

These results suggest that it may be fruitful to combine the best elements of both approaches, perhaps by using SPD Passive-Aggressive in early iterations and SPD Pegasos or SPD SGD-SVM in later iterations. We leave this investigation to future work.

References

[1] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008.

[2] L. Bottou and Y. LeCun. On-line learning for very large datasets. *Applied Stochastic Models in Business and Industry*, 21(2):137–151, 2005.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, 2005.

[4] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking SVM to document retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[5] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7, 2006.

[6] J. L. Elsas, V. R. Carvalho, and J. G. Carbonell. Fast learning of document ranking functions with the committee perceptron. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, 2008.

[7] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[8] T. Joachims. A support vector method for multivariate performance measures. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, 2005.

[9] T. Joachims. Training linear svms in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[10] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.

[11] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Mach. Learn.*, 46(1-3), 2002.

[12] T.-Y. Liu, T. Qin, J. Xu, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR 2007: Workshop on Learning to Rank for Information Retrieval, in conjunction with SIGIR 2007*, 2007.

[13] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, 2007.

[14] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, 2004.

Semi-Supervised Ranking via Ranking Regularization

Martin Szummer `szummer@microsoft.com`

Emine Yilmaz `eminey@microsoft.com`

Abstract

We propose a learning algorithm for ranking functions in the semi-supervised setting, with a training set consisting of a small set of ranked items, and a large set of unranked items. We articulate a regularization principle that exploits unranked data by favoring similar items having similar ranks. This principle is implemented as a semi-supervised extension of LambdaRank. We apply this learning algorithm to a relevance-feedback scenario in information retrieval, and show that semi-supervised learning gives improved rankings.

1 Introduction

In many machine learning settings, there is a small amount of labeled training data, and a large amount of unlabeled data. We would like to leverage both sets of data for training. This task is referred to as semi-supervised learning, or learning with partially labeled data, and there is a wealth of work in semi-supervised classification and regression.

Here we study semi-supervised learning in the context of ranking, where the goal is to learn a function that orders items correctly, rather than learning function values or classes. In this case, the labeled data consists of ranking relationships between items, whereas the unlabeled data is unranked. There is comparatively little prior work in the ranking scenario [4, 6].

The key component of semi-supervised learning is a principle to connect the structure of the unlabeled data with the function between inputs and ranks in the labeled data. In classification, a commonly used principle is the cluster assumption [7], which states that data points in each high-density region (cluster) should have the same labels. Essentially, the unlabeled data define the extent of the clusters, whereas the labeled data determine the class of the clusters. In regression, a principle is to assume that the function value changes slowly in high-density regions. In this paper, we formulate such a principle for ranking: similar items should have similar ranks. We embody this principle in a ranking regularizer that exploits unlabeled data.

The regularizer must be implemented in a learning algorithm. The challenge is that the rank regularizer smooths the rank order of items, rather than class membership or function values. The rank order is a discontinuous function of the parameters we are learning, making semi-supervised ranking more difficult than semi-supervised classification or regression. Fortunately, there exists a flexible learning method called LambdaRank [3], which is very effective for supervised ranking, and which we adapt to our semi-supervised setting.

Our main contributions are:

- a ranking regularizer for semi-supervised learning
- a practical, near-linear time learning algorithm embodying this principle

2 Ranking Regularization

In the semi-supervised ranking problem, we have a set of items i with features $X = \{\boldsymbol{x}_i \in \mathbb{R}^n\}$, to which we would like assign ranks $R = \{r_i \in \mathbb{N}\}$, where $r_i = 1$, denotes the top (preferred) rank. For ranked (labeled) items, $i \in L$, we in addition have some information about their rank, such as pairwise preference relations, or complete or partial orderings. These are not available for the unranked (unlabeled) items, $i \in U$. For notational convenience, we have displayed just a single ranking instance (consisting of a single set of items, and a single rank ordering); in practice, we will have a training consisting of multiple item sets with associated orderings, $\{X^{(k)}, R^{(k)}\}$, from which we learn a single ranking function that takes an item set with features X^* and assigns a complete ranking R^*.

In order for unranked items to help the learning, we need to connect properties of the unranked data distribution with that of the ranked data distribution. In other words, we must make an assumption of how $P(X)$ affects $P(R|X)$. Intuitively, we assume that similar items have similar ranks. This has the same flavor as the cluster assumption for semi-supervised classification and regression. In the ranking context it is a weaker assumption, as it only constrains item order, and not function value (as in regression) or class.

We will focus on the cluster assumption in the context of item pairs, as many ranking algorithms operate on pairwise preferences. In this case the principle can be stated as:

$$d_X(\boldsymbol{x}_i, \boldsymbol{x}_j) \text{ small} \Rightarrow d_R(r_i, r_j) \text{ small.} \tag{1}$$

Here d_X is a dissimilarity measure in item feature space, such as Euclidean distance, and d_R is difference in rank measure, such as $|r_i - r_j|$. This principle works for unlabeled data, as it involves no information about the preference of item i over j. Because it does not rely on labels, the principle does not impose any order of i over j, but merely states that they should be close in the ranking.

We shall also note that the principle does not imply that very dissimilar items should have different ranks, i.e. $d_X(\boldsymbol{x}_i, \boldsymbol{x}_j)$ large $\not\Rightarrow$ $d_R(r_i, r_j)$ large. Such an assumption does not seem reasonable especially for low ranks, where item i and item j can both have bad ranks but for different reasons.

The principle is still abstract, and we must still specify the exact relation between feature space (X) and rank space (R). The challenge is that the feature space can be continuous and high-dimensional, whereas for R we just have a discrete ordering (where for unlabeled data we do not know if $r_i > r_j$ or vice-versa), which in some sense is less than one-dimensional.

We address this by considering a notion that can be defined in both spaces: the notion of near-neighbors. This can be stated as

$$\boldsymbol{x}_i \text{ and } \boldsymbol{x}_j \text{ are mutual near-neighbors} \Rightarrow r_i \text{ and } r_j \text{ are near-neighbors} \tag{2}$$

In practice, we would like to exploit our distance or dissimilarity measure d_X, which we incorporate by defining "soft" nearest neighbors. Let the neighbor weight of i with respect to j be

$$q_{ij} = \mathrm{e}^{-d_{ij}/\sigma_q} \Big/ \sum_k \mathrm{e}^{-d_{ik}/\sigma_q} \tag{3}$$

The strength that i and j are mutual neighbors is given by

$$p_{ij} = q_{ij} q_{ji} = \frac{\mathrm{e}^{-d_{ij}/\sigma_q} \mathrm{e}^{-d_{ji}/\sigma_q}}{\sum_k \mathrm{e}^{-d_{ik}/\sigma_q} \sum_k \mathrm{e}^{-d_{jk}/\sigma_q}} \tag{4}$$

Our final regularizer will penalize rank dissimilarity $d_R(r_i, r_j)$ proportionally to p_{ij}, namely how strongly they are nearest neighbors.

3 A Semi-supervised Ranking Algorithm

We now implement the ranking regularizer in a learning to rank algorithm. We are learning a function $s_i = f(\boldsymbol{x}_i; \boldsymbol{w})$ that assigns a score s_i to an item with features $\boldsymbol{x}_i$. The function has parameters $\boldsymbol{w}$, and in our case it is a 1-hidden layer neural network, although any differentiable

function will do. The function induces a ranking $(i_1, i_2, \ldots, i_m)$ (from high to low) if and only if $s_{i_1} \geq s_{i_2} \geq \cdots s_{i_m}$ (if there are no items with duplicate features). This is akin to the ranking induced by a Thurstonian model. This particular ranking can also be written using the rank variables $r_{i_1} = 1, r_{i_2} = 2, \ldots, r_{i_m} = m$.

The challenge in learning the ranking function $f(\boldsymbol{x}; \boldsymbol{w})$ is that the ranks are discontinuous in the parameters $\boldsymbol{w}$; in particular, the ranking is determined by sorting the scores $\{s_i\}$. Our ranking learning objective can be expressed as a cost function of these sorted scores $C(s_1, s_2, \cdots, s_m; \boldsymbol{w})$ and the training labels (which can be pairwise preferences or ordinal assignments). LambdaRank [3] is an ingenious algorithm that can optimize objectives that have discontinuous gradients. It makes the observation that it can be easier to define the gradient of the cost function with respect to the sorted scores, and optimize it, rather than working with the original cost function, which can be left implicit. LambdaRank is a practical algorithm that successfully learns ranking functions for web-scale information retrieval tasks.

In standard supervised LambdaRank, the gradient of the cost function with respect to score i is written as

$$\mathrm{d}C/\mathrm{d}s_i = -\lambda_i^L(s_1, r_1, l_1, \ldots, s_L, r_L, l_L) \tag{5}$$

The λ^L-functions express how we want the learning objective to change with respect to a change in score s_i, for the current parameters $\boldsymbol{w}$ that yield the ranking scores $s_1, \cdots, s_L$, corresponding ranks $r_1, \ldots, r_L$, with associated ranking labels $l_1, \ldots, l_L$. Positive λ_i suggests an increase in the rank of i. We refer the reader to [3] for details on LambdaRank.

We define semi-supervised LambdaRank as a weighted sum of labeled λ^L and unlabeled gradients λ^U.

$$\mathrm{d}C/\mathrm{d}s_i = -\lambda_i^L(s_1, r_1, l_1, \ldots, s_L, r_L, l_L) - \beta\lambda_i^U(s_1, r_1, \ldots, s_{L+U}, r_{L+U}) \tag{6}$$

The unlabeled gradients are functions of scores alone (and thus can include both labeled and unlabeled items), and this is where we insert the ranking regularizer.

We focus on pairwise preferences: here, we include all possible labeled pairs of items $\{i, j\}$ in λ_L, and all pairs (both labeled and unlabeled) of items in λ_U, although one could use a subset. The gradients decompose as $\lambda_i^L = \sum_{j \in L} \lambda_{ij}^L$ and $\lambda_i^U = \sum_{j \in L \cup U} \lambda_{ij}^U$. One choice of gradients that works well in information retrieval scenarios has the general form

$$\lambda_{ij}^L = L(l_i, l_j) R(r_i^L, r_j^L) S^L(s_i, s_j, l_i, l_j) \tag{7}$$

$$\lambda_{ij}^U = \quad p_{ij} R(r_i^U, r_j^U) S^U(s_i, s_j) \tag{8}$$

This form accomodates the ingredients used by several information retrieval ranking objectives, such as normalized discounted cumulative gain (NDCG) [5], and mean average precision (MAP). These objectives focus on the top of the ranking by discounting items according to rank. The absolute difference of discounts of items at ranks r_i and r_j is given by $R(r_i, r_j)$. Next, $L(l_i, l_j)$ defines how much we prefer label l_i over l_j, and is referred to as the "difference in label gains". This difference is positive if l_i is preferred, and negative if l_j is.

The last factors $S^L(\cdot)$ and $S^U(\cdot)$ incorporates the current ranking scores smoothly into the gradient. For the labeled data we follow [3] and use the (negative) gradient of the RankNet [2] cost, setting $S^L(s_i, s_j, l_i, l_j) = 1/(1 + \exp((\delta_{l_i > l_j} - \delta_{l_j > l_i})(s_i - s_j)))$, where $\delta_{l_i > l_j}$ is 1 if l_i is preferred or 0 otherwise. For the unlabeled data, we use the gradient of the RankNet cost assuming no preference, so $S^U(s_i, s_j) = 1/(1 + \exp(s_i - s_j)) - 0.5$.

The ranking regularizer weight p_{ij} serves the same role as the difference in label gains. One difference between the labeled and unlabeled gradients, is that the unlabeled gradient always tries to reduce the difference in ranks between i and j; the gradient sign is controlled by $S^U(s_i, s_j)$ which is positive if $s_i < s_j$, or 0 if equal, or negative otherwise. In contrast, the labeled gradients can either increase or decrease the rank difference depending on the labels, as the sign is controlled by $L(l_i, l_j)$.

For labeled gradients, we have chosen to measure the discount using ranks r_i^L including only labeled points; alternatively, one could use discounts based on all data points r_i^U in both rankings.

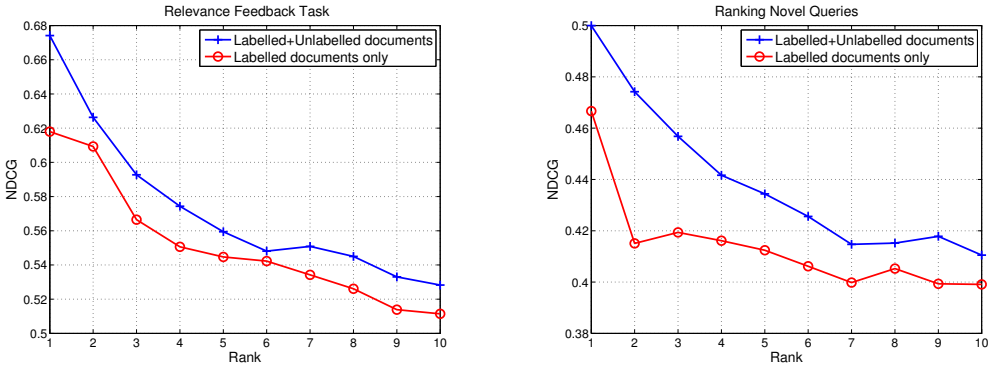


Figure 1: Semi-supervised ranking vs. supervised ranking for the relevance feedback task (left), ranking novel queries (right) (preliminary experiment).

3.1 A (near) linear time algorithm

In semi-supervised learning, we often want to use very large unlabeled datasets, thus computational efficiency is required. We can make the algorithm scale approximately linearly in the number of unlabeled items, by regularizing only with respect to the K nearest neighbors of the item. In particular, let $N_K(i)$ be the K nearest neighbors of i, and define

$$q_{ij} = \begin{cases} \mathrm{e}^{-d_{ij}/\sigma_q} / \sum_{k \in N_K(i)} \mathrm{e}^{-d_{ik}/\sigma_q} & \text{if } j \in N_K(i) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The labeled part of LambdaRank already scales nearly linearly in the number of labeled items, thus yielding an approximately linear algorithm in the total number of data points.

The above also has the benefit of trusting the feature space dissimilarity d_X only in local neighborhoods, giving it a local manifold behavior. We have $p_{ij} = q_{ij}q_{ji}$ as before.

4 Application to Information Retrieval

In order to compare the quality of our proposed semi-supervised ranking algorithm with that of supervised ranking, we focus on the relevance feedback task and the task of ranking novel queries from information retrieval. In the relevance feedback task, given a ranking shown to the user with respect to a query, the user marks a few documents as relevant or non-relevant. These limited labelled documents are then used to rerank the results in order to produce a better ranking. In the task of ranking novel queries, given some initial training set, the goal is to learn to rank a set of documents with respect to any unseen queries. For both scenarios, we use NDCG [5] as the test metric as it is one of the most commonly used metrics for evaluating the quality of a ranking in the case of search.

In our preliminary experiments, we use the TREC collection that was used by Aslam et al. [1]. The document corpus, the queries and the labels in this collection are obtained from TREC 6, 7 and 8 Adhoc retrieval track. Collectively, the dataset contains 150 queries divided into training, validation and testing sets with 90, 30 and 30 queries, respectively. For each query, there are approximately 1000 judged documents available. In order to imitate the case of limited labels, we reduce the amount of available labelled data in the training set by forming a 1% random sample of all the labels for each query. We used $L(l_i, l_j) = 2^{l_i} - 2^{l_j}$ and the absolute difference of discounts $R(r_i, r_j) = |1/\log(r_i) - 1/\log(r_j)|$ /$\mathrm{DCG}_{\max}(Q)$ where $\mathrm{DCG}_{\max}(Q)$ is the maximum DCG value achieved by an ideal ranking for query Q. The $d_X()$ was Euclidean distance between tf-idf document vectors, and the learning parameters $\beta = 1.5$, K=5, $\sigma = \infty$, yielding $q_{ij} = 0.2$ for mutual K neighbors or 0 otherwise.

The relevance feedback task in our scenario corresponds to using the *reduced* (1%) judgments for each query in the training set to rerank the documents in that set. Hence, we train our semi-supervised method and the traditional supervised method using the reduced judgments and evaluate the performance of the algorithms using all the labelled data. The left plot in Figure 1 corresponds to the results of this experiment using NDCG at rank r as the test metric. It can be seen that the semi-

supervised ranking method (using labelled and unlabelled documents) consistently outperforms the traditional supervised ranking algorithm.

The task of ranking novel queries corresponds to using the *reduced* (1%) judgments in the training set to learn to rank new queries. Hence, we train the supervised and semi-supervised learning algorithms using the reduced judgments on the training set. We use the validation set to pick the best performing learning epoch and use the fully judged test set to evaluate the quality of the two ranking algorithms. In the right plot in Figure 1, it can be seen that the semi-supervised ranking method again outperforms the supervised ranking method. The differences between the two algorithms are significant only in some of the cases. Note that this may be caused by the few number of queries used in the test set (30) as the significance tests may not be able to detect significant differences when so few queries are available.

References

[1] Javed A. Aslam, Evangelos Kanoulas, Virgiliu Pavlu, and Emine Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. In *SIGIR '09: Proceedings of the 32nd annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2009. ACM.

[2] Chris Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML*, 2005.

[3] Christopher J.C. Burges, Robert Ragno, and Quoc Viet Le. Learning to rank with nonsmooth cost functions. In *NIPS*, 2006.

[4] K. Duh and K. Kirchhoff. Learning to rank with partially-labeled data. In *SIGIR*, 2008.

[5] Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48, New York, NY, USA, 2000. ACM.

[6] Ming Li, Hang Li, and Zhi-Hua Zhou. Semi-supervised document retrieval. *Information Processing & Management*, 45(3):341 – 355, 2009.

[7] Matthias Seeger. Learning with labeled and unlabeled data. Unpublished. `http://www.dai.ed.ac.uk/homes/seeger/`, February 2001.

SampleRank: Learning Preferences from Atomic Gradients

Michael Wick, Khashayar Rohanimanesh, Aron Culotta*, Andrew McCallum
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
{*mwick,khash,culotta,mccallum*}*@cs.umass.edu*

Abstract

Large templated factor graphs with complex structure that changes during inference have been shown to provide state-of-the-art experimental results on tasks such as identity uncertainty and information integration. However, learning parameters in these models is difficult because computing the gradients require expensive inference routines. In this paper we propose an online algorithm that instead learns preferences over hypotheses from the gradients between the atomic steps of inference. Although there are a combinatorial number of ranking constraints over the entire hypothesis space, a connection to the frameworks of sampled convex programs reveals a polynomial bound on the number of rankings that need to be satisfied in practice. We further apply ideas of passive aggressive algorithms to our update rules, enabling us to extend recent work in confidence-weighted classification to structured prediction problems. We compare our algorithm to structured perceptron, contrastive divergence, and persistent contrastive divergence, demonstrating substantial error reductions on two real-world problems (20% over contrastive divergence).

1 Introduction

The expressive power of probabilistic programming languages (Richardson and Domingos, 2006; Milch et al., 2006; Goodman et al., 2008; McCallum et al., 2009) has given rise to complex factor graphs that preclude exact training methods because traditional machine learning algorithms involve expensive inference procedures as subroutines. For example, maximum likelihood gradients require computing marginals and perceptron gradients require decoding. Furthermore, these inference routines must be invoked for each update, and therefore lie in some of the inner loops of learning. A number of approaches address this issues to various degrees (Hinton, 2002; Hal Daumé and Marcu, 2005; Tieleman, 2008). For example, LASO learns to score incomplete configurations based on a binary loss function that determines if a partial configuration could lead to the ground-truth; contrastive divergence (CD) approximates gradients by sampling in the neighborhood of the ground-truth to obtain inexpensive updates.

In this paper we present *SampleRank* (Culotta, 2008), an alternative to contrastive divergence that computes *atomic* gradients between neighboring configurations according to a loss function (for example, F1 score). This signal induces a preference over the samples, and parameters are learned to reflect these preferences. Because SampleRank is concerned only with the ranking of hypothesized samples, and not with approximating likelihood gradients, the algorithm is not required to be governed by a strict Markov chain. In particular, this allows large parameter updates to be made between intermediate samples, an immediate advantage over persistent contrastive divergence (PCD) which

*Southeastern Louisiana University (culotta@selu.edu)

must ensure updates between neighboring samples are sufficiently small. Therefore, SampleRank enjoys greater freedom in updating the parameters, and indeed we can apply ideas from passive aggressive algorithms (MIRA) (Crammer et al., 2006) and feature-specific confidence-weights (Dredze et al., 2008) in order to achieve even greater performance.

In our experiments, we compare SampleRank to several alternative learning algorithms and demonstrate that it reduces error over several variations of CD, PCD, and perceptron on three different datasets. We also explore different update rules including MIRA and confidence weighting, and finally compare CD and SampleRank in chains not governed by strict ergodic properties.

2 Preliminaries

A factor graph $\mathcal{G} = \langle V, \Psi \rangle$ is a bipartite representation of a probability distribution π, that decomposes into a set of factors $\Psi = \{\psi\}$. Random variables V can further be divided into observables X and hidden variables Y. Using lowercase letters (e.g., x, y) to denote values from the domains of the random variables, the conditional distribution given by the factor graph can be written: $\pi(Y = y \mid X = x; \theta) = \frac{1}{\mathcal{Z}_X} \prod_{\psi \in \Psi} \psi(y^i, x^i)$. Where $\mathcal{Z}_{\mathbf{X}}$ is the input-dependent normalizer, and factor ψ takes assignments to sets of hidden y^i variables and observables x^i as arguments. We define the feasible region $\mathcal{F} \subseteq Y$ of the factor graph to contain only non-zero-probability configurations $\mathcal{F} = \{y \in Y \mid \pi(y|x) > 0\}$.

Parameter learning in factor graphs generally involves the update rule: $\theta \leftarrow \theta + \eta\nabla$, where ∇ is a correction that is applied to the current estimate of the parameters, and η is the learning rate. For example, maximum likelihood gradients ($E_{\mathcal{D}} \langle \Phi \rangle - E_\theta \langle \Phi \rangle$) involve the $\#\mathcal{P}$-hard problem of computing marginals for feature expectations under the model, and Collins' structured perceptron gradient ($\phi(y^\star_{\mathcal{D}}) - \phi(y^\star_\theta)$) requires the $\mathcal{NP}$-hard problem of computing the MAP configuration $y^\star_\theta$.

3 SampleRank

In this section we describe our algorithm, SampleRank, which learns configuration rankings from atomic gradients. Recall that this work is motivated by the fact that most gradient methods require an expensive black-box inference routine $\mathbb{B}$ (e.g., for returning $y^\star$). Now, we will assume $\mathbb{B}$ is no longer a black-box, and we can indeed observe the underlying procedure that performs inference. Furthermore, we will assume that each step of that procedure produces a configuration pair $\langle y', y \rangle$ (as is the case with MCMC and local search methods). More precisely, let $\delta : \mathcal{F} \rightarrow \mathcal{F}$ be the nondeterministic transition function that represents a draw from a proposal distribution $\mathbb{Q} : \mathcal{F} \times \mathcal{F} \rightarrow [0, 1]$ s.t. $\sum_{y'} \mathbb{Q}(y'|y) = 1$. At each step of inference δ is invoked to yield a new configuration y' from a current configuration y. Let $\phi : Y \times X \rightarrow \mathbb{R}^{|\theta|}$ denote the sufficient statistics of a configuration, then we define the *atomic gradient* ∇ (from inference step $y' = \delta(y)$) as

$$\nabla = \phi\left(\delta(y), x\right) - \phi(y, x) = \phi(y', x) - \phi(y, x) \tag{1}$$

Let G_θ be a factor graph representation of a probability distribution π parameterized by θ, with feasible region $\mathcal{F}$; and let $\mathbb{P} : Y \times Y \rightarrow \{0, 1\}$ be a preference function:

$$\mathbb{P}(y', y) = \begin{cases} 1 & \text{if } y' \text{ is preferred} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

then SampleRank is given in Algorithm 1. Despite its apparent simplicity, SampleRank is actually quite general. In fact, SampleRank provides tremendous flexibility, enabling both proposal distributions and preference functions to be customized to a particular domain or setting. For example, in unsupervised settings, the preference function could by governed by prior knowledge (or by measuring a generative process); in supervised settings, preference functions can exploit ground-truth labels (e.g., comparing F1 scores). We have established a convergence proof in our technical report (Rohanimanesh et al., 2009) that is completely agnostic to the preference function and is applicable to nearly arbitrary (including non-ergodic) proposal distributions. Many inference algorithms of interest (such as Gibbs, and Metropolis-Hastings), are covered by the convergence results.

Algorithm 1 Atomic Gradient Method (SampleRank)

Inputs: training data $\mathcal{D}$ with factor graph $\mathcal{G} = \langle X, Y, \Psi, \theta \rangle$
Initialization: set $\theta \leftarrow \mathbf{0}$, set $y \leftarrow y_0 \in \mathcal{F}$
Output: parameters θ
for $t = 1, 2, \ldots$ until convergence **do**
  $y' \sim \mathbb{Q}(\cdot|y)$
  $\nabla \leftarrow \phi(y', x) - \phi(y, x)$
  if $[\theta \cdot \nabla > 0 \wedge \mathbb{P}(y, y')]$ **then**
    $\theta \leftarrow \theta - \eta\nabla$
  else if $[\theta \cdot \nabla \leq 0 \wedge \mathbb{P}(y', y)]$ **then**
    $\theta \leftarrow \theta + \eta\nabla$
  end if
  if chooseToAccept(y, y', θ) **then** $y \leftarrow y'$
end for

Finally, we note that although the learning rate η is traditionally a scalar, it can be adjusted by a passive aggressive method (MIRA), or be vector-valued (as in confidence weighting). We adapt these methods to our structured setting by casting each update as a binary classification problem where the configuration pair is the data instance and the preference function serves as the label. The sufficient statistics of the classification problem are then the components of the atomic gradient ∇.

3.1 Efficient Gradient Computations

Equation 1 implies that computing ∇ requires obtaining sufficient statistics from two configurations y and y', which can be expensive. However, due to the local nature of search, this can be avoided entirely. Taking advantage of the fact that sufficient statistics present in both y and y' cancel, we can compute ∇ directly as: $\nabla = \sum_{\phi \in \nu(\Delta')} \phi(y^i, x) - \sum_{\phi \in \nu(\Delta)} \phi(y^i, x)$, where Δ' is the new setting to the variables that have changed, and Δ is the previous setting to those variables before the transition. The neighborhood function $\nu(\Delta')$ returns the sufficient statistics that require these variable settings as arguments. For many commonly used transition functions (e.g., Gibbs or split-merge (Jain and Neal, 2004)), we save computing an order of n factors over the brute force method.

3.2 Sample Complexity

Although our algorithm considers a combinatorial number of ranking constraints (in the configuration space of the factor graph), SampleRank can alternatively be viewed as an instance of *randomized constraint sampling* (Farias and Roy, 2004, 2006) or *sampled convex programs (SCP)* (Calafiore and Campi, 2005), where errors are bounded on approximations to convex optimization problems involving an intractable number of constraints. The unifying idea of these frameworks is the notion of a relaxed optimization problem that considers just a manageable set of *i.i.d.* constraints. This manageable set is actually sampled from the full constraint set according to a probability distribution ρ. Solutions are then obtained by optimizing the relaxed problem over the subset of constraints. The underlying intuition of this idea is that most constraints are either (a) inactive, (b) redundant (captured by other constraints), or (c) negligible (have only a minor impact on the solution). The fundamental question that these frameworks address is how many samples are required such that the solution to the resulting relaxed optimization problem violates only a *small* subset of constraints. It has been shown (Farias and Roy, 2004) that in particular, for a problem with K variables, with a number of sampled constraints given by: $N = \mathcal{O}\left(\frac{1}{\epsilon}\left(K \ln \frac{1}{\epsilon} + \ln \frac{1}{\delta}\right)\right)$ any optimal solution to the relaxed problem with a probability at least $(1 - \delta)$ violates a set of constraints $\mathcal{V}$ with measure $\rho(\mathcal{V}) \leq \epsilon$, where $\rho(.)$ is a probability distribution over the constraint space from which *i.i.d.* sample constraints are generated.

SampleRank can be described as the following SCP:

$$\begin{cases} \text{minimize} & c^T\theta \\ \text{subject to} & \theta \cdot \phi(x, y^-) - \theta \cdot \phi(x, y^+) \leq 0, \quad \forall \langle y^-, y^+ \rangle \in \Omega_{\mathbb{Q}} \end{cases}$$

where c is a vector of importance weights, and $\Omega_{\mathbb{Q}}$ is a set of sampled constraints generated by SampleRank throughout the course of local search (e.g. MCMC) guided by a proposal distribution

$\mathbb{Q}(.))$[1]. Taking $K = |\theta|$ reveals that a reasonable model can be learned by sampling a polynomial size subset of the constraints.

4 Experiments

In this section we demonstrate how SampleRank can be used to train a conditional random field (CRF) with first-order logic features defined over sets of instances. In particular, we focus on two clustering problems: ontology alignment and noun-phrase coreference resolution. In ontology alignment, all concepts belonging to the same cluster are considered equivalent; similarly, in coreference, all mentions belonging to the same cluster are considered coreferent.

Setup:
The CRF contains variables for each possible cluster (with a factor measuring the cluster's compatibility) and variables between mention-pairs across clusters (with a factor measuring their disparity), resulting in a combinatorial number of variables and factors. For more details about this CRF, please see (Culotta et al., 2007), or our technical report (Rohanimanesh et al., 2009). For MAP inference, we use Metropolis-Hastings, where the proposal distribution randomly picks a data-point then randomly moves it to another cluster (or creates a new cluster). SampleRank treats each proposal as an atomic inference step[2]; our preference function for both problems exploits the ground-truth labels and is defined to be $\mathbb{P}(y', y) = 1$ if accuracy(y')>accuracy(y), and 0 otherwise.

Data:
For coreference experiments we use the ACE 2004 dataset, which contains 443 documents; 336 for training and 107 for testing. We run each online method over the training set (ten times), performing 4000 proposals (inference steps) per document. For the ontology experiments we use two domains from the Illinois Semantic Integration Archive (ISIA): *course catalog*, and *company profile* (for more discussion on these domains see Doan et al. (2002)).

Results:
First, we compare the BCubed F1 (in coreference) of three learning rates η: constant unit updates (f1=77.6), MIRA updates (f1=80.5), and the approximate version of confidence weighted updates (f1=81.5). Confidence weighted updates have previously been shown to improve results in classification problems, and we were pleased to see a similar improvement in a structured prediction setting. Next (Table 1), we compare SampleRank to variants of contrastive divergence, persistent contrastive divergence, and perceptron on three datasets (ACE newswire coreference, course catalog ontology alignment, and company profile ontology alignment). We observe substantial error reductions over variants of contrastive divergence (more than 20% on ACE coreference—a new state-of-the-art result); in particular, we observe even greater improvements (over CD) in chains lacking detailed balance. Columns indicated as *valid MCMC chain* use a proposer that moves a single variable and obeys detailed balance. The column indicated as *not valid MCMC chain* uses a more sophisticated proposer that adapts to the model, but does not necessarily obey detailed balance. Note how the sophisticated proposal distribution hinders performance for CD, but actually helps SampleRank.

5 Acknowledgements

This work was supported in part by the Center for Intelligent Information Retrieval, in part by SRI International subcontract #27-001338 and ARFL prime contract #FA8750-09-C-0181, in part by The Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grant #IIS-0326249, in part by Army prime contract number W911NF-07-1-0216 and University of Pennsylvania subaward number 103-548106, and in part by UPenn NSF medium IIS-0803847. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

[1]Note that in this particular case the choice of the importance weight vector c is unimportant (e.g., we can chose $c = \mathbf{0}$) if the goal is to find a feasible solution for θ. For a quadratic program, the optimization objective should be replaced by $\theta^T\theta$.

[2]Despite the large graph, computing the atomic gradients requires evaluating only a constant number of cluster-wise factors.

	ACE coreference				Ontology alignment	
	valid MCMC chain		not valid MCMC chain		valid MCMC chain	
Method	**F1 (B^3)**	**F1 (PW)**	**F1 (B^3)**	**F1 (PW)**	**F1 (Course)**	**Match F1 (Company)**
SampleRank	**80.1**	**45.1**	**81.5**	**51.0**	**89.8**	**82.1**
CD-1	75.1	22.4	75.1	22.4	76.9	64. 8
CD-10	76.03	33.7	73.1	19.3	72.4	67.8
PCD-10	77.9	37.3	75.7	19.5	67.9	74.6
Perceptron	—	—	—	—	69.7	60.2

Table 1: Comparison of SampleRank with other training methods

References

Calafiore, G. and Campi, M. C. (2005). Uncertain convex programs: Randomized solutions and confidence levels. *Mathematical Programming*, 102:25–46.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585.

Culotta, A. (2008). *Learning and inference in weighted logic with application to natural language processing*. PhD thesis, University of Massachusetts.

Culotta, A., Wick, M., Hall, R., and McCallum, A. (2007). First-order probabilistic models for coreference resolution. In *HLT*, pages 81–88.

Doan, A., Madhavan, J., Domingos, P., and Halevy, A. Y. (2002). Learning to map between ontologies on the semantic web. In *WWW*, page 662.

Dredze, M., Crammer, K., and Pereira, F. (2008). Confidence-weighted linear classification. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 264–271, New York, NY, USA. AC-M.

Farias, D. P. D. and Roy, B. V. (August 2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478.

Farias, V. F. and Roy, B. V. (2006). Tetris: A study of randomized constraint sampling. *Probabilistic and Randomized Methods for Design Under Uncertainty, G. Calafiore and F. Dabbene, eds.*

Goodman, N. D., Mansighka, V. K., D. Roy, K. B., and Tenenbaum, J. B. (2008). A language for generative models. In *UAI*.

Hal Daumé, I. and Marcu, D. (2005). Learning as search optimization: approximate large margin methods for structured prediction. In *ICML*, pages 169–176, New York, NY, USA. ACM.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800.

Jain, S. and Neal, R. M. (2004). A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13:158–182.

McCallum, A., Rohanimanesh, K., Wick, M., Schultz, K., and Singh, S. (2009). Factorie:probabilistic programming via imperatively defined factor graphs. In *Neural Information Processing Systems(NIPS)*, Vancouver, BC, Canda.

Milch, B., Marthi, B., and Russell, S. (2006). *BLOG: Relational Modeling with Unknown Objects*. PhD thesis, University of California, Berkeley.

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62:107–136.

Rohanimanesh, K., Wick, M., and McCallum, A. (2009). Inference and learning in large factor graphs with adaptive proposal distributions. Technical Report #UM-CS-2009-028, University of Massachusetts, Amherst.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. In *ICML*, pages 1064–1071, New York, NY, USA. ACM.

Learning to Rank Through the Wisdom of Crowds*

Jennifer Wortman Vaughan
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
jenn@seas.harvard.edu

Abstract

In this talk, I will describe the interesting synergy that exists between prediction markets and machine learning, and show how it can be leveraged to run an efficien market for predicting rankings. I will begin with a brief introduction to prediction markets, focusing on the popular Logarithmic Market Scoring Rule [7, 8], and go on to provide an overview of recent work detailing how prediction markets can be used to efficientl learn rankings from public opinion [5]. This work relies heavily on the recently discovered connection between the Logarithmic Market Scoring Rule and the Weighted Majority algorithm, which I will also discuss.

1 Introduction

Suppose we are interested in learning an accurate estimate of the probability that a federal health insurance plan will be approved by the US government by the end of the year. We could spend days poring over news articles and weighing various opinions against each other, eventually coming up with a reasonably informed estimate. However, we might be able to save ourselves a lot of hassle (and get a more accurate estimate at the same time) by appealing to the wisdom of crowds.

A *prediction market* is a betting market designed to aggregate opinions [15]. A typical binary prediction market allows bets along one dimension, for example, for or against a federal health plan being approved. In this case, bettors would trade shares of a security that pays off \$1 if and only if a plan is approved by the year's end. If the current market price of a share is \$$p$, then a rational, risk-neutral bettor should be willing to buy shares if he believes the true probability of approval is greater than p. Conversely, he should be willing to sell shares if he believes the true probability is lower. In this sense, the current price per share provides an estimate of the population's collective belief about how likely it is that a health plan will be approved.

It is natural to ask how accurate these estimates are. Under certain assumptions (in particular, the assumption that all bettors are risk-neutral Bayesians with a common prior), it can be shown that the trading price will converge to a *rational expectations equilibrium* that reflect the probability estimate that one would obtain by combining the private side information of each member of the population [6]. Perhaps more convincingly, the forecasts obtained through prediction markets in practice have frequently proved more accurate than forecasts provided by domain experts. The price of orange juice futures has been a better predictor of weather than the National Weather Service forecasts [16], Oscar markets have been more accurate at predicting winners than expert colum-

*Pieces of this extended abstract are based on material that originally appeared in Chen et al. [5]. This material is included with the permission of my coauthors.

nists [14], and election markets are often more accurate than national polls [2]; see Pennock and Sami [15] or Ledyard et al. [11] for a range of other examples.

Thousands of one- or small-dimensional markets exist today, each operating independently. At the racetrack, betting on a horse to win does not directly impact the odds that the horse will show (i.e., finis among the top two), as logically it should, because the two types of bets are handled separately. This can lead to obvious opportunities for arbitrage. On the contrary, a *combinatorial prediction market* is a central clearinghouse for handling logically-related bets define on a combinatorial space, such as a set of rankings. For example, the outcome space might be all $n!$ possible rankings of n horses in a horse race (or equivalently, of n candidates in an election), while bets may be on properties of these rankings such as "horse A finishe third" or "horse A beats horse B."

There has been some recent work showing that by restricting the betting language appropriately (in particular, allowing bets only of the form "horse i finishe in position j"), combinatorial markets can be run tractably in a call auction setting [4, 1]. However, call auctions tend to suffer from the *thin market problem*, and many trades are refused. To combat this, Chen et al. [5] examined the possibility of using Hanson's Logarithmic Market Scoring Rule [7, 8] (described below) for pricing bets on ranks. The Logarithmic Market Scoring Rule (or LMSR) satisfie a variety of desirable properties, including infinit liquidity; a bettor may always purchase shares of any security if he is willing to pay enough. Unfortunately, Chen et al. showed that problem of pricing bets exactly in such markets is #P-hard and set out to fin a way to approximate prices instead. In the process, they discovered a striking connection between LMSR and the standard Weighted Majority algorithm. By exploiting this connection, they were able to show that Helmbold and Warmuth's recent PermELearn algorithm [9], an extension of Weighted Majority for permutation learning, can be used to approximate prices (and therefore also probability estimates) in markets over rankings.

These results hint there is a powerful (and still largely unexplored) synergy between prediction markets and machine learning. In the case of predicting rankings, prediction markets allow us to collect and aggregate the data that we need to learn, and machine learning gives us the tools we need to run efficien markets. The goal of this talk is to introduce members of the machine learning community to prediction markets, point out and explain the connections that exist between the fields and show how these connections apply to learning rankings.

2 The Logarithmic Market Scoring Rule

Originally proposed by Hanson [7, 8], a Logarithmic Market Scoring Rule is a type of automated market maker mechanism. LMSRs have grown popular in practice because they satisfy many desirable properties. They offer consistent pricing for combinatorial events, with no opportunities for arbitrage. They provide infinit liquidity by allowing trades on any security at any time. The market maker is guaranteed to have a bounded loss in the worst case, making it more enticing to run the market in the firs place. Finally, it is a dominant strategy for a (myopic, risk-neutral) bettor to bet truthfully, revealing his true beliefs, which is crucial if we wish to use the market to learn. LMSRs are used by a number of companies, including Microsoft, inklingmarkets.com, thewsx.com, and yoonew.com, and are the subject of a number of research studies [3, 11].

Let Ω be a set of mutually exclusive and exhaustive outcomes of a future event. A generic (non-combinatorial) LMSR offers a security corresponding to each $\omega \in \Omega$. The security associated with outcome ω pays off \$1 if ω happens, and \$0 otherwise. Let q_ω indicate the current number of outstanding shares of the security for outcome ω. The market maintains a *cost function*

$$C(\vec{q}) = b \log \sum_{\omega \in \Omega} \mathrm{e}^{q_\omega / b}, \tag{1}$$

where $b > 0$ is a parameter related to the depth of the market. This cost function captures the total amount of money currently wagered in the market. If a trader wishes to purchase r_ω shares of the security associated with each outcome ω (where r_ω can be zero or negative), the cost for this

purchase is $C(\vec{q}+\vec{r}) - C(\vec{q})$. Thus the *instantaneous price* of shares for outcome ω is simply the partial derivative of the cost function, i.e.,

$$p_\omega(\vec{q}) = \frac{\partial C(\vec{q})}{\partial q_\omega} = \frac{\mathrm{e}^{q_\omega/b}}{\sum_{\omega' \in \Omega} \mathrm{e}^{q_{\omega'}/b}} \,. \tag{2}$$

This instantaneous price represents the current cost per share of an infinitel small quantity of the security for ω. It also represents the market's current estimate of the probability that ω will occur; a rational, risk-neutral bettor who disagrees with the probability distribution implied by the current price vector has an incentive to trade shares until the prices are aligned with his own beliefs.

3 Markets over Rankings

Suppose we would like to run a market to obtain a probability distribution over all possible rankings of candidates in an election. In this setting, the outcome space Ω is the set of all permutations over candidates $\{1, \cdots, n\}$. Each $\sigma \in \Omega$ represents the outcome in which each candidate i ends up in position $\sigma(i)$. Running a standard LMSR over the full outcome space is both infeasible (since $|\Omega| = n!$) and unintuitive for human bettors, who may have trouble reasoning about such probabilities. We might hope to overcome this problem by placing restrictions on the betting language used.

Much recent work has focused on a simplifie market in which traders may bet only on securities of the form $\langle i|j \rangle$ which pay out \$1 if and only if candidate i is in position j in the fina ranking. Chen et al. [4] and Agrawal et al. [1] showed that restricting attention to this betting language leads to tractable algorithms in a *call auction* setting. In this setting, bettors submit limit orders for bets they would like to place (for example, "I would like to purchase 10 shares of security $\langle i|j \rangle$ at no more than \$0.20 per share") and after all bets have been collected, the auctioneer determines which bets to accept and what prices to charge.

The LMSR has many advantages over call auctions. It is run in an online manner, so bettors receive immediate feedback as to whether or not their bets are accepted. Perhaps more importantly, it gets around the "thin market problem" (or low liquidity) faced by call auctions, always allowing trades of any security at some price. It is natural then to ask whether this same restricted betting language admits tractable pricing in combinatorial LMSRs.

We firs examine how to extend the definitio of LMSR to this new setting. Let $q_{i,j}$ be the total number of outstanding shares for security $\langle i|j \rangle$ in the market. Let $Q = (q_{i,j})$ denote the outstanding shares for all securities. Note that if the fina permutation is σ, the number of winning shares will be $\sum_{k=1}^{n} q_{k,\sigma(k)}$. Using this and Equation 1, we can write the LMSR cost function as

$$C(Q) = b \log \sum_{\sigma \in \Omega} \mathrm{e}^{\sum_{k=1}^{n} q_{k,\sigma(k)}/b} = b \log \sum_{\sigma \in \Omega} \prod_{k=1}^{n} \mathrm{e}^{q_{k,\sigma(k)}/b} \,.$$

Similarly, from Equation 2, we can see that the instantaneous prices can be written

$$p_{i,j}(Q) = \frac{\sum_{\sigma \in \Omega : \sigma(i)=j} \mathrm{e}^{\sum_{k=1}^{n} q_{k,\sigma(k)}/b}}{\sum_{\sigma \in \Omega} \mathrm{e}^{\sum_{k=1}^{n} q_{k,\sigma(k)}/b}} = \frac{\sum_{\sigma \in \Omega : \sigma(i)=j} \prod_{k=1}^{n} \mathrm{e}^{q_{k,\sigma(k)}/b}}{\sum_{\sigma \in \Omega} \prod_{k=1}^{n} \mathrm{e}^{q_{k,\sigma(k)}/b}} \,.$$

Unfortunately, in contrast to the positive results for call auctions, Chen et al. [5] showed that computing the instantaneous prices, the cost function, or the payments of transactions for LMSR betting under this restricted betting language is #P-hard. However, by pointing out an interesting connection between prediction markets and machine learning, they were able to use machine learning techniques to approximate the prices. This connection is described below.

4 The Connection Between LMSR and Learning from Expert Advice

Before jumping into the connection between LMSR and Weighted Majority, we briefl review the standard setting for learning from expert advice. At each time $t \in \{1, \cdots, T\}$, each *ex-*

pert $i \in \{1, \cdots, n\}$ receives a *loss* $\ell_{i,t} \in [0, 1]$. The *cumulative loss* of expert i at time T is $\mathcal{L}_{i,T} = \sum_{t=1}^{T} \ell_{i,t}$. An algorithm $\mathcal{A}$ maintains a current weight $w_{i,t}$ for each expert i, where $\sum_{i=1}^{n} w_{i,t} = 1$. These weights can be viewed as a distribution over the experts. The algorithm then receives its own instantaneous loss $\ell_{\mathcal{A},t} = \sum_{i=1}^{n} w_{i,t}\ell_{i,t}$, which can be interpreted as the expected loss the algorithm would receive if it always chose an expert to follow according to the current distribution. The cumulative loss of $\mathcal{A}$ up to time T is define in the natural way as $\mathcal{L}_{\mathcal{A},T} = \sum_{t=1}^{T} \ell_{\mathcal{A},t} = \sum_{t=1}^{T} \sum_{i=1}^{n} w_{i,t}\ell_{i,t}$. The typical goal in this setting is to minimize the algorithm's *regret*, define to be the difference between the cumulative loss of the algorithm and the loss of the best performing expert in hindsight (formally, $\mathcal{L}_{\mathcal{A},T} - \min_{i\in\{1,\cdots,n\}} \mathcal{L}_{i,T}$). No statistical assumptions are made about these losses, and in general, algorithms are expected to perform well even if the sequence of losses is chosen by an adversary.

The popular Weighted Majority (WM) algorithm [13], sets the weight of each expert i at time t to be $w_{i,t} = \mathrm{e}^{-\eta\mathcal{L}_{i,t}} / \sum_{j=1}^{n} \mathrm{e}^{-\eta\mathcal{L}_{j,t}}$, where $\eta > 0$ is the *learning rate*. It is well known that the regret of WM after T trials can be bounded as $\mathcal{L}_{WM(\eta),T} - \min_{i\in\{1,\cdots,n\}} \mathcal{L}_{i,T} \leq \eta T + \log(n)/\eta$. When T is known in advance, setting $\eta = \sqrt{\log(n)/T}$ yields the standard $\sqrt{T\log(n)}$ regret bound.

There is a manifest similarity between the WM weights and the LMSR prices; simply compare the WM weights with the form of Equation 2. One might ask if the results from the experts setting can be applied to the analysis of prediction markets. It turns out that they can. To gain intuition, we will firs describe the result of Chen et al. [5] that shows that it is possible to use the WM regret bound to rediscover the well-known bound of $b\log(n)$ for the loss of an LMSR market maker with n outcomes.[1] Below we show how this connection can be exploited in the combinatorial setting.

We begin by breaking up each LMSR purchase (or sale) of q shares into $\lceil q/\epsilon \rceil$ sequential purchases of no more than ϵ shares each. Note that the total number of trades needed to execute such a sequence of purchases is proportional to $1/\epsilon$. We will create a set of n experts, each corresponding to one of the n market outcomes, and construct their sequence of losses in such a way that the weights that WM assigns to each expert are precisely the instantaneous prices of the corresponding outcomes in the LMSR, with each time step in the expert setting corresponding to a single trade in the LMSR.

Let $p_{i,t} \in [0, 1]$ be the instantaneous price of security i after the tth trade has been made, and let $q_{i,t} \in [-\epsilon, \epsilon]$ be the number of shares of security i purchased during the tth trade. Let $Q_{i,t}$ be the total number of shares of security i that have been purchased up to time t. Defin the instantaneous loss of each expert as $\ell_{i,t} = (\epsilon - q_{i,t})/(\eta b)$. First notice that this loss is always in $[0, 1]$ as long as $\eta \geq 2\epsilon/b$. Furthermore, from Equation 2 and the definitio of WM, at each time t,

$$p_{i,t} = \frac{\mathrm{e}^{Q_{i,t}/b}}{\sum_{j=1}^{n} \mathrm{e}^{Q_{j,t}/b}} = \frac{\mathrm{e}^{\epsilon t/b - \eta\mathcal{L}_{i,t}}}{\sum_{j=1}^{n} \mathrm{e}^{\epsilon t/b - \eta\mathcal{L}_{j,t}}} = \frac{\mathrm{e}^{-\eta\mathcal{L}_{i,t}}}{\sum_{j=1}^{n} \mathrm{e}^{-\eta\mathcal{L}_{j,t}}} = w_{i,t} \ .$$

Setting $\ell_{i,t} = (\epsilon - q_{i,t})/(\eta b)$ in the standard WM regret bound and letting $\eta = 2\epsilon/b$ (which, as we mentioned above, is the smallest value we can choose for η while guaranteeing that each expert's instantaneous loss is in $[0, 1]$) gives us

$$\max_{i\in\{1,\cdots,n\}} \sum_{t=1}^{T} q_{i,t} - \sum_{t=1}^{T}\sum_{i=1}^{n} p_{i,t}q_{i,t} \leq 4\epsilon^2 Tb + b\log(n). \tag{3}$$

Let's examine this expression. The firs term on the left-hand side is the worst-case payment that the market maker could potentially need to make when the true outcome is revealed. As ϵ approaches 0, the second term on the left-hand side approaches the total payment received by the market maker (since $p_{i,t}$ is the price per share of purchasing an infinitel small quantity of security i at time t). Finally, since $T = O(1/\epsilon)$, the term $4\epsilon^2 Tb$ goes to 0 as ϵ becomes very small. Thus in the limit as ϵ

[1] While we will not discuss it here, it turns out that the reverse is also true; that is, the market maker bound can be used to re-derive the well-known $O(\sqrt{T\log(n)})$ bound on the regret of WM.

approaches 0, we get the well-known result that the worst-case loss of the market maker is bounded by $b \log(n)$.

There is an interesting interpretation of this connection. In essence, the market maker is learning a probability distribution over outcomes by treating each security trade as a training instance. Unlike most learning settings, the training data is actively adapting to help the learning algorithm (bettors trade shares only if they believe the algorithm's current distribution is wrong given everything they know about all bets which have been made). This could give us a hint about why prediction markets tend to yield such accurate predictions in practice.

5 Using Machine Learning to Approximate LMSR Prices for Rankings

Helmbold and Warmuth [9] recently showed how to extend results from expert learning to a setting in which the goal is to compete with the best *permutation* over n items. In this setting, each permutation suffers a loss every time step, and the goal is to maintain a weighting over permutations such that the cumulative regret of the algorithm with respect to the best permutation is small. It is generally infeasible to treat each permutation as an expert and run a standard no-regret algorithm since this would require updating $n!$ weights at each time step. Instead, Helmbold and Warmuth show that when the loss has a certain structure (in particular, when the loss of a permutation is the sum of the losses of each of the n mappings), it is possible to get away with maintaining only n^2 weights in the form of an $n \times n$ doubly stochastic matrix.

Formally, let W^t be a doubly stochastic matrix of weights maintained by the algorithm $\mathcal{A}$ at time t. Here $W^t_{i,j}$ is the weight corresponding to the probability associated with item i being mapped into position j. Let $L^t \in [0,1]^{n \times n}$ be the loss matrix at time t. The instantaneous loss of a permutation σ at time t is $\ell_{\sigma,t} = \sum_{i=1}^{n} L^t_{i,\sigma(i)}$. The instantaneous loss of $\mathcal{A}$ is $\ell_{\mathcal{A},t} = \sum_{i=1}^{n}\sum_{j=1}^{n} W^t_{i,j} L^t_{i,j}$, the matrix dot product between W^t and L^t. Notice that $\ell_{\mathcal{A},t}$ is equivalent to the expectation over permutations σ drawn according to W^t of $\ell_{\sigma,t}$. The goal of the algorithm is to minimize the cumulative regret to the best permutation, $\mathcal{L}_{\mathcal{A},T} - \min_{\sigma \in \Omega} \mathcal{L}_{\sigma,T}$ where the cumulative loss is define as before.

Helmbold and Warmuth present an algorithm called PermELearn that updates the weight matrix in two steps. First, it creates a temporary matrix W', such that for every i and j, $W'_{i,j} = W^t_{i,j} \mathrm{e}^{-\eta L^t_{i,j}}$. It then obtains $W^{t+1}_{i,j}$ by alternately rescaling the rows and columns of W' until the matrix is doubly stochastic, a process known as Sinkhorn balancing [17]. Although there are cases in which Sinkhorn balancing does not converge in finit time, many results show that the number of iterations needed to scale a matrix so that row and column sums are $1 \pm \epsilon$ is polynomial in $1/\epsilon$ [10, 12].

Let $\mathcal{L}_{\mathcal{A}(\eta),T}$ denote the cumulative loss of the PermELearn algorithm with parameter η after T time steps. Helmbold and Warmuth show that $\mathcal{L}_{\mathcal{A},T} \leq (n \log(n) + \eta \min_{\sigma \in \Omega} \mathcal{L}_{\sigma,T})/(1 - \mathrm{e}^{-\eta})$.

Using the PermELearn algorithm, it is possible to approximate LMSR prices for the combinatorial LMSR in which traders may purchase securities of the form $\langle i, j \rangle$ described above in polynomial time. We start with an $n \times n$ price matrix P^1 in which all entries are $1/n$. As in Section 4, each time a trader purchases or sells q shares, the purchase or sale is broken up into $\lceil q/\epsilon \rceil$ purchases or sales of ϵ shares or less, where $\epsilon > 0$ is a small constant.[2] Thus we can treat the sequence of purchases as a sequence of T purchases of ϵ shares or less, where $T = O(1/\epsilon)$. Let $q^t_{i,j} \in [-\epsilon, \epsilon]$ be the number of shares of security $\langle i|j \rangle$ purchased at time t.

The price matrix is updated in two steps. First, a temporary matrix P' is created where for every i and j, $P'_{i,j} = P^t_{i,j} \mathrm{e}^{q^t_{i,j}/b}$ where $b > 0$ is a parameter playing a similar role to b in Equation 2. Next, P' is Sinkhorn balanced to the desired precision, yielding an (approximately) doubly stochastic matrix P^{t+1}. Chen et al. [5] show that computing updates in this way leads to a price matrix that is

[2] We remark that dividing purchases in this way has the negative effect of creating a polynomial time dependence on the quantity of shares purchased. However, this is not a problem if the quantity of shares bought or sold in each trade is bounded to start, which is a reasonable assumption. The additional time required is then linear only in $1/\epsilon$.

equivalent to the weight matrix of PermELearn when $L^t_{i,j} = (\epsilon - q^t_{i,j})/(\eta b)$ for all i, j, and t, for any $\eta \geq 2\epsilon/b$. The following theorem is the analog of Equation 3 for the ranking setting.

Theorem 1 (Chen et al. [5]) *For any sequence of valid purchases q^t where $q^t_{i,j} \in [-\epsilon, \epsilon]$ for all t, i, and j, let $P^1, \cdots, P^T$ be the price matrices obtained by running the ranking approximation algorithm. Then*

$$\max_{\sigma \in S_n} \sum_{t=1}^{T} \sum_{i=1}^{n} q^t_{i,\sigma(i)} - \sum_{t=1}^{T} \sum_{i=1}^{n} \sum_{j=1}^{n} P^t_{i,j} q^t_{i,j} \leq \left(\frac{2\epsilon/b}{1 - \mathrm{e}^{-2\epsilon/b}} \right) bn \log(n) + \left(\frac{2\epsilon/b}{1 - \mathrm{e}^{-2\epsilon/b}} - 1 \right) \epsilon n T \, .$$

Let us again examine this expression as ϵ approaches 0. The term $(2\epsilon/b)/(1 - \mathrm{e}^{-2\epsilon/b})$ goes to 1 in the limit. Additionally, T scales inversely with ϵ since each lump purchase of q shares must be broken into $\lceil q/\epsilon \rceil$ individual purchases. Thus in the limit as ϵ approaches 0, we see that the loss of the market maker is bounded by $bn \log(n)$.

This bound is comparable to worst-case loss bounds achieved using alternate methods for operating LMSRs over rankings. A single LMSR operated on the entire outcome space has a guaranteed worst-case loss of $b \log(n!)$, but is, of course, intractable to operate. A set of n LMSRs operated as n separate markets, one for each position, would also have a total worst-case loss $bn \log(n)$, but could not guarantee consistent prices. In the limit, our approximation algorithm achieves the same worst-case loss guarantee as if we were operating n separate markets, but prices remain consistent at all times, allowing us to learn a consistent probability distribution over the rankings.

References

[1] S. Agrawal, Z. Wang, and Y. Ye. Parimutuel betting on permutations. In *Proceedings of the 4th International Workshop on Internet and Network Economics*, 2008.

[2] J. Berg and T. Rietz. Accuracy and forecast standard error of prediction markets. Technical report, University of Iowa, College of Business Administration, 2002.

[3] Y. Chen and D. M. Pennock. A utility framework for bounded-loss market makers. In *Proceedings of the 23rd Conference on Uncertainty in Artificia Intelligence*, pages 49–56, 2007.

[4] Y. Chen, L. Fortnow, E. V. Nikolova, and D. M. Pennock. Betting on permutations. In *Proceedings of the Eighth ACM Conference on Electronic Commerce*, 2007.

[5] Y. Chen, L. Fortnow, N. Lambert, D. M. Pennock, and J. Wortman. Complexity of combinatorial market makers. In *Proceedings of the Ninth ACM Conference on Electronic Commerce*, 2008.

[6] S. J. Grossman. An introduction to the theory of rational expectations under asymmetric information. *Review of Economic Studies*, 48(4):541–559, 1981.

[7] R. Hanson. Combinatorial information market design. *Information Systems Frontiers*, 5(1):105–119, 2003.

[8] R. Hanson. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets*, 2007.

[9] D. Helmbold and M. Warmuth. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10:1705–1736, 2009.

[10] Bahman Kalantari and Leonid Khachiyan. On the complexity of nonnegative-matrix scaling. *Linear Algebra and its applications*, 240:87–103, 1996.

[11] J. Ledyard, R. Hanson, and T. Ishikida. An experimental test of combinatorial information markets. *Journal of Economic Behavior and Organization*, 2008. To appear.

[12] N. Linial, A. Samorodnitsky, and A. Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica*, 20(4):545–568, 2000.

[13] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108 (2):212–261, 1994.

[14] D. Pennock, C. Giles, and F. Nielsen. The real power of artificia markets. *Science*, 291:987–988, 2001.

[15] D. M. Pennock and R. Sami. Computational aspects of prediction markets. In N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[16] R. Roll. Orange juice and weather. *American Economic Review*, 74(5):861–880, 1984.

[17] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

Author Index

NOTES