Learning to Rank for Robust Question Answering

Arvind Agarwal*
Department of Computer Science
University of Maryland
College Park, MD 20742 USA
arvinda@cs.umd.edu

Hema Raghavan
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598 USA
hraghav@us.ibm.com

Karthik Subbian†
University of Minnesota
Minneapolis, MN 55455 USA
karthik@cs.umn.edu

Prem Melville
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598 USA
pmelvil@us.ibm.com

Richard D. Lawrence
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598 USA
ricklawr@us.ibm.com

David C. Gondek
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598 USA
dgondek@us.ibm.com

ABSTRACT

This paper aims to solve the problem of improving the ranking of answer candidates for factoid based questions in a state-of-the-art Question Answering system. We first provide an extensive comparison of 5 ranking algorithms on two datasets – from the Jeopardy quiz show and a medical domain. We then show the effectiveness of a cascading approach, where the ranking produced by one ranker is used as input to the next stage. The cascading approach shows sizeable gains on both datasets. We finally evaluate several rank aggregation techniques to combine these algorithms, and find that Supervised Kemeny aggregation is a robust technique that always beats the baseline ranking approach used by Watson for the Jeopardy competition. We further corroborate our results on TREC Question Answering datasets.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering, Retrieval models

General Terms

Algorithms

Keywords

Ranking, Question-answering, Rank-aggregation

*The part of this work was done during the internship at IBM T. J. Watson Research Center, Yorktown Heights, NY

†The work was done during the employment at IBM T. J. Watson Research Center, Yorktown Heights, NY.

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

1. INTRODUCTION

Question answering (QA) is the task of finding an exact answer to a user's natural language question. In February, 2011, the IBM Watson QA system beat human grand champions in the game of Jeopardy!, marking a major milestone in QA research. This was made possible by great advances in all stages of the QA system pipeline, including question analysis, search, hypothesis generation, and hypothesis scoring. One of the key steps at the end of this pipeline is ranking the candidate answers produced by weighing all the evidence extracted and scored in support of each answer. In Watson, this task is solved by using a Machine Learning approach, where a model is built based on a set of training questions with known answers, and then used to score new question-answer pairs. This task can be viewed as a special case of rank-learning, and as such recent advances in learning to rank [6] may be effectively applied here. In this paper, we provide an extensive study of different approaches to ranking candidate answers as applied to QA.

Most recent advances in learning to rank have been driven by web search [8]. While the task of ranking candidate answers in QA is similar to document-retrieval, there are some key differences: (1) Instead of relevance, which can be multi-graded in search, we just have binary relevance judgments, corresponding to the right or wrong answer. (2) The data has high class imbalance. While search datasets also tend to have a high ratio of irrelevant to relevant documents, in the case of QA, there is typically only one correct answer. (3) Search systems are typically evaluated on an entire ranked list or the top 10 results produced, using metrics such as ERR, NDCG, and MAP. However, in most QA systems the desired goal is to get the top-ranked answer right [15, 36]. As such Precision@1 becomes the primary metric to optimize.

The above distinctions make answer-ranking in QA a special case of the learning to rank problem, and observations made in the context of document retrieval may not hold. This paper makes the following contributions: (1) To the best of our knowledge this is the first work that provides an extensive comparison of several approaches to learning to rank, applied specifically to answer-ranking in open-domain QA (Jeopardy), as well as in the highly specialized domain of medicine. (2) We show how results can be improved using a cascading approach, where the rankings produced by one ranker are provided as inputs to another. Such a cascading approach not only gives better performance, but also speeds up runtime prediction. (3) We demonstrate how multiple rankers can be

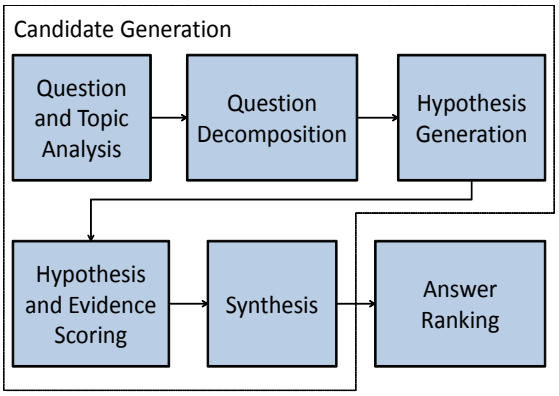

Figure 1: Overview of Watson

effectively combined using rank aggregation techniques stemming from the Social Choice Theory literature. As will be seen later in our experiments, in one dataset, logistic regression performs best, while in other, LambdaRank perform best; and a combined system performs better than any individual ranker in both cases. (4) In practice, since it is hard to find a single ranker that performs best on all datasets, our proposed system is a practical solution to this problem of having to rely on one ranker. Our system does not rely on any single best ranker, rather combines different rankers and gives the best of all. (5) The final system proposed in this paper is both efficient and robust, giving improvements of 3.4 and 2.2% points in Precision@1, over the algorithm used by Watson on the Jeopardy and Medical task respectively. We show similar improvements in performance on publicly available TREC QA datasets.

We believe our work furthers the state-of-the-art in Open Domain QA by showing how learning to rank (LETOR) can be effective for the task. At the time of running the TREC QA tasks which ended in 2007, pointwise classification methods like logistic regression and Maximum Entropy Models were state of the art for the final answer selection component [19, 27, 32]. Many advances in Machine Learning for Information Retrieval tasks have been made since then. We provide an extensive comparison of several approaches of LETOR and Rank Aggregation for QA and improve over the baseline Watson system built for Jeopardy. Given the renewed interest in QA with Jeopardy, Siri and the DARPA BOLT Program[1], this work establishes a baseline for future QA research. We emphasize that improvements on Jeopardy are significant because the system was worked on for several years and obtaining improvements has not been trivial.

2. ANSWER RANKING IN WATSON

We begin by providing an overview of the Watson QA system. Watson answers questions by first analyzing the question, generating candidate answers, and then collecting evidence over its text and knowledge base resources supporting or refuting those answers. For each answer, individual pieces of evidence are scored by an ensemble of answer scorers, yielding features capturing the degree to which evidence justifies or refutes an answer. Most features provide some measure of how justified the answer is for the question according to evidence in text passages or knowledge base information. However, there are also scores representing question-level features which indicate information about the question (for instance the answer type specified by the question) as well as answer-level features (for instance the popularity of an answer). An example of a *high recall* score is a measure of the weighted overlap between terms in a question and a passage. In contrast, an example of a *high precision* score is a measure of the alignment of grammatical and semantic relationships between the question and passage using a graphical abstraction.

Finally, these features are used to rank candidate answers, based on the likelihood the answer is correct. Crafting successful strategies for resolving thousands of answer scores into a final ranking would be difficult, if not impossible, to optimize by hand; so Watson was designed to learn over existing questions and their correct answers. From a Machine Learning perspective, a training/test instance here, is a feature vector presenting evidence for or against a question-answer pair being correct. Fig. 1 shows an overview of the Watson system pipeline. In this paper, we only focus on the final stage of ranking candidate answers. For a longer description of the machine learning component of Watson the reader is referred to [17].

Architecturally, Watson provides a confidence estimation framework which uses a common data model for registration of answer scores and performs machine learning over large sets of training data in order to produce models for answer ranking and confidence estimation. While the ranking component is critical for *deploying* a full question answering system, it is also essential for *developing* the system. Developers working on any aspect of the system, from candidate generation to answer scoring, must evaluate the system-wide impact of their component on end-to-end question answering performance. This development methodology requires performing thousands of experiments requiring retraining of the system to assess the impact of every potential change. This drives the need for machine learning which is computationally efficient, to allow for rapid experimentation, and which is highly automatic, to achieve valid results without requiring manual parameter tuning or specialist knowledge of machine learning algorithms to perform each experiment. This approach allows developers to focus on optimizing performance of their component and entrust to the framework the assessment and proper combination of the component's scores for the end-to-end question answering task.

3. EXPERIMENTAL METHODOLOGY

We now describe our datasets and evaluation approach.

3.1 Datasets

We evaluate our system on two datasets – Jeopardy and Medical. The questions and correct answers for the Jeopardy dataset are obtained from historical records of the American TV quiz show, Jeopardy. In Watson, answer candidates are generated by searching for hypotheses through several sources like Wikipedia, the Bible, IMDB, ebooks on Project Gutenberg, using the INDRI[29] search engine. NLP techniques are used to understand the focus and category of a question, find the lexical answer type, and finally narrow down on an answer-candidate in the retrieved passages. Details of the techniques used for answer-candidate generation can be found elsewhere [15, 13].

The Medical dataset was obtained from a Jeopardy-like competition held by the American College of Physicians called the *Doctor's Dilemma*. The underlying content being searched in this case includes several medical content sources such as: ACP Medicine, Merck Manual of Diagnosis and Therapy and MKSAP (a study

[1] http://bit.ly/vViNIG

	Jeopardy		Medical	
	Train	Test	Train	Test
No. of questions	9337	2961	909	607
Avg. no. of candidates per question	220	223	244	241
Avg. no. of correct answers per question	1.30	1.20	1.57	1.77

Table 1: Dataset statistics

guide from ACP). Several candidate answers are generated per question using a system similar to Watson but which had been adapted for the medical domain [14].

Although the original datasets contain questions which do not have a correct answer in the candidate set, in our experiments, as in [31], we only consider those questions that have at least one correct answer. Table 1 shows a summary of the data.

For both datasets several hundred features are extracted as described in the previous section. There are 547 features in the Jeopardy dataset and 323 in the Medical dataset. We refer the reader to the references for a detailed description of the NLP components. The focus of this paper is to study the effectiveness of various algorithms on ranking the candidate answers. An example question in the Medical dataset is *PULMONOLOGY: Diagnosis associated with "egg shell" calcification of intrathoracic lymph nodes*. Watson produces candidate answers such as *Sarcoidosis*, *Silicosis* and *Lymphadenopathy*. Our task is to correctly rank these answers, so that the correct answer(s) (*Silicosis*) is placed at the top of the list. In all experiments in this work, the feature set is kept constant and the goal is to evaluate algorithms for ranking.

3.2 Evaluation Measures

The output of our system is a *ranked list* of candidate answers for each question. However, for most QA tasks we only get credit for selecting one correct response – for instance, in the Jeopardy! challenge a contestant is only allowed to give one answer. As such, our primary evaluation metric is Precision@1 (P@1), which measures the percentage of questions for which the top-ranked answer is correct.

In some domains, such as Medical QA, there can be more than one correct answer – for instance, two valid diagnoses for the same symptoms. In order to evaluate this setting, we also measure Normalized Discounted Cumulative Gain [20] at rank K ($K = 5, 10$). NDCG@K looks at the top-K candidate answers, and assigns a higher weight to a correct answer that is ranked higher than one that is ranked lower. NDCG@K is computed as:

$$NDCG@K = \frac{1}{Q}\sum_{q=1}^{Q}\frac{DCG_q}{IdealDCG_q}, \quad (1)$$

where Q is the total number of queries, and

$$DCG_q = \sum_{i=1}^{K}\frac{2^{y_{qi}}-1}{\log_2(1+i)} \quad (2)$$

y_{qi} is the relevance level of candidate answer i for query q. $IdealDCG$ is simply DCG with the ideal ranking. While P@1 only looks at the top-ranked response, NDCG provides a finer granularity for evaluation, rewarding algorithms for having a higher rank for a correct answer, even if it is not at the top of the list. All metric numbers in this paper are reported in %.

4. LEARNING TO RANK

Over the last decade, learning to rank methods have been effectively applied to information retrieval tasks. These methods aim at learning a model that given a query and a set of relevant documents, finds the appropriate ranking of documents according to their relevancy. A question answering task is similar to the information retrieval task i.e., a question is simply a query and a set of candidate answers is analogous to a set of relevant documents. The goal is now to find the appropriate ranking of these candidate answers according to their correctness. There have been numerous learning to rank methods developed, which can be divided into three main categories: pointwise methods, pairwise methods, and listwise methods.

Pointwise methods treat the ranking problem as a standard classification or a regression task. These methods assume that each question-answer pair has either (a) a numerical or ordinal (rank) score associated with it or (b) a relevance label in one of two or more classes. The objective in the former formulation is to find a model that predicts this score correctly through ordinal regression methods [10]. In the latter case, the problem is reduced to classification and can be solved by methods such as SVMs or logistic regression [25].

Pairwise methods like FRank [34], SVMRank [21], RankNet [5], RankBoost [16] aim to learn the pairwise preference of candidate answers rather than their absolute rank. The intuition behind these approaches is that in information retrieval one cares about metrics like NDCG and MAP which reward a system for a ranking of results as opposed to an absolute prediction of relevance, and modeling preferences is closer to that final objective. In these ranking methods, given a ranked set of candidate answers for a query, preferences expressing that one answer is preferred over the other are constructed from each pair of answers. More specifically, each pair of candidate answers is given a binary label $\{+1, -1\}$ based on if the first answer in the pair has a higher rank than the second answer. This construction transforms the original ranking problem into a binary classification problem of predicting these pairwise preferences. The goal is now to learn a binary classifier that minimizes the number of incorrectly ordered pairs. A total ordering of candidates is inferred from the predicted pairwise preferences.

Listwise methods operate on the entire list of candidate answers. Unlike pointwise and pairwise methods where a loss based on the rank of the individual candidate answer or of a pair is minimized; in listwise methods, a direct loss (an appropriate evaluation measure defined by the user) is minimized between the true ranks of the list and the estimated ranks of the list. These methods are the most sophisticated and have been shown to outperform the other two types of methods for information retrieval tasks [28]. Many of these methods allow for the optimization of the final metric relevant to the application. Examples of listwise methods are LambdaRank [28], Coordinate-Ascent [24], AdaRank [37], ListNet [7] and [40].

In this work, we compare the following representative learning to rank methods for each of the 3 categories, applied to the task of question-answering. We emphasize here that these different rankers were selected based their ease of availability and implementation, and also based on their ability to run on a large dataset.

Logistic Regression (LR) is a binary classification method giving a score that is a probability of relevance, and is therefore a natural as well as effective choice for question-answering tasks [27, 19] including for Watson for the Jeopardy challenge [13, 35]. It is a pointwise method, and simply minimizes the logistic loss between the true label of an answer and the estimated label (represented as a real value). At the test time, it assigns a probability of relevance to each candidate answer which can be sorted to get the appropriate

Dataset	LR	Rank Boost	Ada-Rank	Coord-Ascent	Lambda Rank
Jeopardy	69.0	62.8	55.3	-	66.9
	80.9	76.3	71.2	-	79.0
Medical	43.3	39.2	33.4	28.8	39.2
	54.6	51.7	44.0	41.4	52.0

Table 2: Performance of different learning to rank methods. Top entry in each cell is P@1 while bottom is NDCG@10.

ranking. Although logistic regression has not received much attention in the information retrieval task because of the other sophisticated learning to rank methods, it has been successfully used in the question answering task e.g. Watson system, to rank answers. The team working on the Watson system has consistently found logistic regression to beat other point-wise methods and logistic regression was used in the final Jeopardy game[17].

RankBoost [16] is a pairwise ranking method based on boosting. RankBoost first constructs the pairs (as would be done for any pairwise method) based on the preference order, and assigns each pair a label +1 if the first answer is ranked higher than the second, and -1 if the second answer is ranked higher that the first; and transforms the ranking task into a binary classification task. RankBoost then applies the boosting algorithm on this classification task. It starts with equal weights assigned to all pairs, and in each round, uses weak rankers to reassign the weights. The pairs that are correctly ranked are given lower weights while those incorrectly ranked are given higher weights. In the end, a linear combination of these weak rankers is used for the final ranking.

AdaRank [37] is similar to RankBoost except that it is a listwise approach. The weak rankers in AdaRank are learned by directly optimizing ranking measures such as NDCG or MAP, unlike in RankBoost, where a pairwise loss is minimized.

Coordinate-Ascent is a listwise method proposed by [24]. It is a linear feature-based method that directly optimizes the ranking measure using the well-known optimization method i.e., coordinate ascent. Optimization is performed cyclically, optimizing one parameter at-a-time while keeping others fixed.

LambdaRank [28] is a listwise method based on the pairwise ranking method, RankNet [5]. LambdaRank is based on the idea that in order to learn a model the actual value of the loss function is not needed, in fact only the gradient of the loss function is sufficient. Once a gradient is known, it can be used with standard optimization methods e.g. gradient descent to minimize the original cost function. An intuitive technique is used to compute the gradient for different evaluation measures like NDCG and MAP and is used to learn the model.

Note that our datasets are relatively large ($\sim$4GB, both train and test) and some methods, such as SVMRank [21] took more than a day to train on the smaller (Medical) dataset, and were therefore discarded for this study.

As a sanity check of our implementations, we experimented and found that all rankers (except logistic regression, as it is unsuitable for multivalued data) were quite competitive when evaluated on the Yahoo! dataset provided as a part of a recent learning to rank challenge [8].

4.1 Experiments

We evaluated the performance of the different learning to rank methods on the Jeopardy and Medical datasets. For logistic regression, we used the implementation in WEKA [18]. For RankBoost, Coordinate-Ascent, and AdaRank, we used RankLib [11]; where AdaRank and Coordinate-Ascent optimized P@1. For LambdaRank, we used our own implementation, written to optimize NDCG (as it cannot directly optimize P@1). For all rankers, we used the default hyperparameters suggested by authors, without any tuning.

Results are presented in Table 2 for two metrics P@1 (top entry in each cell), and NDCG@10 (bottom). Given its computational complexity, Coordinate-Ascent did not terminate on Jeopardy (after 2 days), and as such we only present its results on the smaller dataset (Medical). We see that despite being the simplest, the pointwise method logistic regression performs best on both datasets and on both metrics, followed by LambdaRank and RankBoost. This behavior is in contrast to the one noted by the information retrieval community, where listwise methods usually outperform pairwise methods, and pairwise methods outperform pointwise methods [28]. This could be the case for the following reasons: (1) Pointwise methods may be desirable in this degenerate case of learning to rank, where we only have binary relevance and mostly just 1 correct answer amongst many incorrect answers. We will see in the next section, that the other rankers often outperform logistic regression when there is less of a class imbalance. (2) The features generated over the years of Watson development were tested with logistic regression. So there might be an inherent bias in feature generation and selection in favor of logistic regression. This ordering in performance is not what we expect in general – as we have independently verified by applying the above rankers on web search data.

In the section, we will provide a mechanism that will not only improve on each of these results, but also speed up the training time, allowing us to fill in the missing entries corresponding to Coordinate-Ascent in Table 2.

5. PRUNING

As discussed previously Question Answering for factoid questions is a high accuracy task and one is typically interested in providing one (or very few) precise answers. Matveeva et al. ([23]) showed that training a learning to rank algorithm (Rank-Net in their case) using the top N results from an initial ranking helped improve precision at the top of the ranked list for web-search. In this section we study the impact of restricting the training set for the learning to rank algorithms to the top N candidates determined by the baseline Logistic Regression trained on all the training data. The idea behind such a pruning technique is that if the first stage is reasonably accurate, the re-ranking phase can focus on improving the precision at high ranks. The motivation is different from active learning or boosting where the most difficult examples are added to the training set in subsequent training phases.

In the experiments in this section we train a base logistic regression using all training data. A new model is trained using the top N answers for each query. For each such model trained we test its performance on the top N candidates as ranked by the base logistic regression on the test data. We repeat this procedure for all rankers including logistic regression and the results are shown in Figure 2.

From the figure, all rankers including logistic regression show significant improvements due to pruning on Jeopardy. When $N = 5$, logistic regression achieves a P@1 of 72.2% which is a *statistically significant* improvement compared to the baseline model's P@1 of 69.0%. When $N = 5$, the performance of LambdaRank (P@1=71.8%) is on par with that of logistic regression and their performances are statistically indistinguishable.

On the medical data all rankers show improvements in going from using all the data to pruning at 50. Unlike Jeopardy however, all the rankers achieve their peak performance at different pruning thresholds. LambdaRank peaks at $N = 20$ and further pruning seems to hurt its performance. To understand this effect, we

list the Success@K metric for the two datasets for the base Logistic Regression model. The Success@K metric is the percentage of queries for which a relevant result exists in the top-K results. While on the Jeopardy data, even for $K = 5$ the base Logistic Regression model achieves 90% success, on the Medical data this value sharply drops after $K = 20$. We suspect that the quality of the initial ranker is a factor that determines the extent of pruning that can be done for a corpus. The impact of pruning on other metrics like NDCG was similar to P@1.

We also experimented with using the models trained on the top N candidates to re-score the *entire* test data but found that our approach of pruning the test data was slightly better. Such pruning of data for training and testing the second pass rankers has the advantage of speeding both train and test times allowing us to use more complex rankers that might not be feasible on a large dataset. For example, while we were unable to run Co-ordinate Ascent on the entire Jeopardy data, pruning the results to $N = 50$ allowed us to evaluate and use this ranker. An additional advantage of pruning is that it reduces the class bias and therefore allows one to benefit from more sophisticated learning to rank methods, an observation also made in [35]. In our experiments, we have observed (see Figure 2) that in case of full data, pointwise methods such as logistic regression outperform pairwise and listwise methods, but when the data is pruned, listwise methods e.g., LambdaRank start to take over.

K	Jeopardy	Medical
50	98.31%	94.39%
30	97.46%	90.44%
20	96.55%	87.80%
10	94.25%	80.23%
5	90.74%	71.82%

Table 3: Success@K for the baseline ranker, on the two corpora

In the next section, we will improve on these results even further by combining these different rankers which will be the output of our final ranking system.

6. RANK AGGREGATION

The performance of individual rankers vary significantly across different datasets and training set sizes, as seen in Fig. 2. This suggests that instead of relying on the single best ranker, it may be better to combine all rankers to produce a more robust and accurate ranking. Since individual rankers produce an ordering of elements, and not a pointwise score that can be meaningfully aggregated, we can leverage approaches to aggregating rankings that have been studied in Social Choice Theory, and successfully applied to meta-search [12].

We begin by formally defining the rank aggregation task. Given a set of entities S, let V be a subset of S; and assume that there is a total ordering in V. We are given r individual rankers $\tau_1, ..., \tau_r$ who specify their order preferences of the m candidates, where m is size of V, i.e., $\tau_i = [d_1, ..., d_m], i = 1, ..., r$, if $d_1 > ... > d_m, d_j \in V, j = 1, ..., m$. If d_i is preferred over d_j we denote that by $d_i > d_j$. Rank aggregation function ψ takes input orderings from r rankers and gives τ, which is an aggregated ranking order. If V equals S, then τ is called a *full list* (total ordering), otherwise it is called a *partial list* (partial ordering).

All commonly-used rank aggregation methods, satisfy one or more of the following desirable *goodness* properties: Unanimity, Non-dictatorial Criterion, Neutrality, Consistency, Condorcet Criterion and Extended Condorcet Criterion (ECC) [1]. We will primarily focus on ECC, defined below:

Algorithm 1 Supervised Kemeny Ranking (SKR)

Input: $\tau_i = [\tau_{i1}, ..., \tau_{im}], \forall i = 1, ..., r$, ordered arrangement of m candidates for r rankers.
$w = [w_1, ..., w_r]$ – where w_i is the weight of ranker i
$\mu = [\mu_1, ..., \mu_m]$ – initial ordered arrangement of m candidates
k – the number candidates to consider in each ranker's preference list ($k \leq m$)
Output: τ – rank aggregated arrangement of candidates in decreasing order of relevance

1. Initialize majority table $M_{i,j} \Leftarrow 0, \forall i, j = 1, ..., m$
2. For each ranker $p = 1$ to r
3. For each candidate $i = 1$ to k-1
4. For each candidate $j = i + 1$ to k
5. $M_{\tau_{pi}, \tau_{pj}} \Leftarrow M_{\tau_{pi}, \tau_{pj}} + w_p$
6. Quick sort μ, using M_{μ_i, μ_j}. If $M_{\mu_i, \mu_j} - M_{\mu_j, \mu_i} > 0$ then μ_i is greater than μ_j. If $M_{\mu_i, \mu_j} - M_{\mu_j, \mu_i} = 0$ then μ_i is equal to μ_j. If $M_{\mu_i, \mu_j} - M_{\mu_j, \mu_i} < 0$ then μ_i is less than μ_j.
7. Return τ

DEFINITION 1. *The Extended Condorcet Criterion [33] requires that if there is any partition $\{C, R\}$ of S, such that for any $d_i \in C$ and $d_j \in R$ a majority of rankers prefer d_i to d_j, then the aggregate ranking τ should prefer d_i to d_j.*

The ECC property is highly desirable in our domain, as it eliminates the possibility of inferior candidates in a ranking to affect the choice between superior candidates. In other words, it offers the property of Independence of Irrelevant Alternatives. Additionally, ECC is a relaxed form of Kemeny optimal aggregation (defined below), where the partition C and R are arranged in the "true" order, but not necessarily the elements within partitions C and R. In addition to the desirable theoretical properties, ECC proves to be very valuable in ranking in practice [12, 30], as is further corroborated in our experiments.

We will focus on two classical rank aggregation techniques in this paper: Borda and Kemeny, described below:

Borda Aggregation: In Borda aggregation [4] each candidate is assigned a score by each ranker. The score for a candidate is the number of candidates below it in each ranker's preferences. The Borda aggregation is the descending order arrangement of the average Borda score for each candidate averaged across all ranker preferences. Though Borda aggregation satisfies neutrality, monotonicity, and consistency, it does not satisfy the Condorcet Criterion [38] and ECC. In fact, it has been shown that no method that assigns weights to each position and then sorts the results by applying a function to the weights associated with each candidate satisfies the Extended Condorcet Criterion [12]. This includes pointwise classifiers like logistic regression. This motivates the use of order-based methods for rank aggregation that do satisfy ECC.

Kemeny Aggregation: A Kemeny optimal aggregation [22] is an aggregation that has the minimum number of pairwise disagreements with all rankers, i.e., a choice of τ that minimizes:

$$K(\tau, \tau_1, ..., \tau_r) = \frac{1}{r} \sum_{i=1}^{r} k(\tau, \tau_i);$$

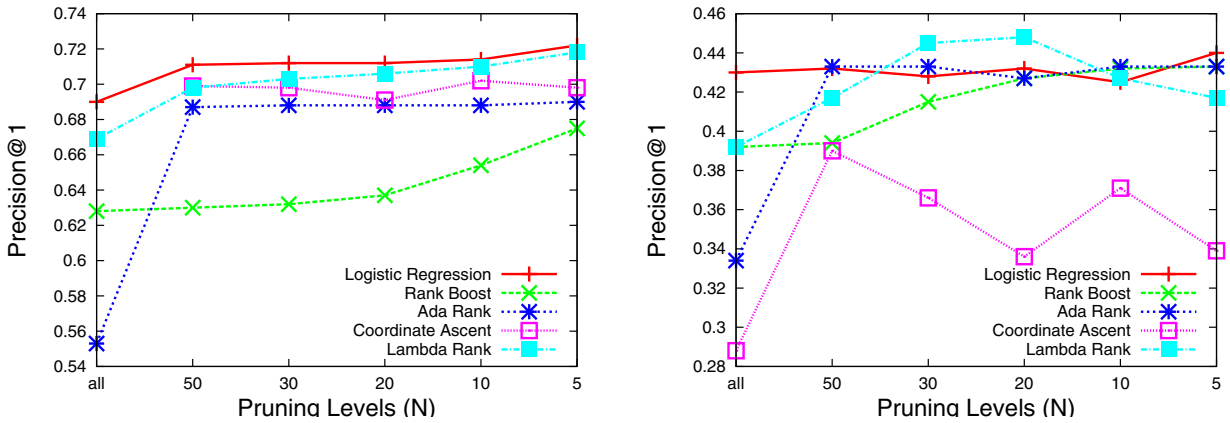

Figure 2: Performance (P@1) for various pruning levels (N) for Jeopardy (left) and Medical (right) datasets.

Datasets	Rankers						Aggregation					
							Unsupervised			Supervised		
	LR	Rank Boost	AdaRank	Coord Ascent	Lambda Rank	Mean	Borda	Kemeny	Kemeny Top-k	Borda	Kemeny	Kemeny Top-k
Jeopardy	72.2	67.5	69.0	69.8	71.8	70.1	72.2	72.3	72.0	72.0	**72.4**	**72.4**
(pruned@5)	82.3	80.2	80.9	81.0	82.2	81.3	82.3	82.3	82.1	82.3	**82.4**	**82.4**
Medical	43.2	42.7	42.7	33.6	44.8	41.4	43.3	44.0	43.8	44.0	**45.5**	**45.5**
(pruned@20)	54.7	53.6	54.6	48.0	**55.7**	53.3	55.0	54.9	54.9	55.3	**55.7**	**55.7**

Table 4: Performance of rank-aggregation methods along with the learning to rank methods on Medical and Jeopardy data. The top entry in each cell is Precision@1, while the bottom is NDCG@10. The best results for each setting are presented in bold.

where the function $k(\sigma,\tau)$ is the *Kendall tau* distance measured as $|\{(i,j)|i<j,\sigma(i)>\sigma(j),$but $\tau(\mathrm{i})<\tau(\mathrm{j})\}|$, where $\sigma(i)$ is used to denote the position of i in ranking σ.

Kemeny aggregation satisfies neutrality, consistency, and the Extended Condorcet Criterion. Kemeny optimal aggregation also has a good maximum likelihood interpretation [39]. Suppose there is an underlying "correct" ordering σ of S, and each order $\tau_1,...,\tau_r$ is obtained from σ by swapping pairs of elements with some probability less than $1/2$. That is, the τ's are "noisy" versions of σ. A Kemeny optimal aggregation of $\tau_1,...,\tau_r$ is one (not necessarily unique) that is maximally likely to have produced the τ's.

While Kemeny aggregation is optimal in the sense described above, computing a Kemeny aggregation is NP-Hard for $r \geq 4$ [12]. So in practice, we use an approach that produces a 2-approximation of Kemeny optimal aggregation, referred to as Approximate Kemeny in [30]. The Approximate Kemeny can be described simply as a Quick Sort on elements using the majority precedence relation $\succ$ as a comparator, where $d_i \succ d_j$ if the majority of input rankings has ranked d_i before d_j. In [30], Approximate Kemeny is shown to satisfy the ECC property. Unless otherwise specified, we will use *Kemeny* to refer to this approximation in the rest of the paper.

Instead of using total orderings provided by each ranker, we can use partial orderings (for a subset of candidates). Since identifying relevant candidates at the top of the list is more important, we use partial orderings corresponding to the top k candidates for each ranker. In our experiments, we use the top-ranked 50% of candidates for each ranker, and refer to this variant as *Kemeny Top-k*.

Supervised Rank Aggregation: Kemeny and Borda aggregation, being motivated from Social Choice Theory, strive for *fairness* and hence treat all rankers as equally important. However, fairness is not a desirable property in our setting, since we know that some individual rankers perform better than others in answer-ranking. If we knew *a priori* which rankers are better, we could leverage this information to produce a better aggregate ranking. In fact, given the ordering of a (validation) set of candidates, we can estimate the performance of individual rankers and use this to produce a better ranking on a new set of candidates.

In order to accommodate such supervision, *Supervised Kemeny Ranking* [30] extends Approximate Kemeny aggregation to incorporate weights associated with each input ranking. The weights correspond to the relative utility of each ranker; so for our experiments we use weights proportional to the P@1 computed on the training set. The pseudo-code for Supervised Kemeny Ranking is presented in Algo. 1. Analogously to the unsupervised setting, we use *Supervised Kemeny Top-k* to refer to SKR with k set to 50% of m.

As in SKR, for *Supervised Borda* (See Algo. 2), we incorporate performance-based (Precision@1) weights in Borda aggregation by taking weighted averages of Borda scores instead of simple averages. A similar approach to supervised Borda was used in [2].

6.1 Experiments

Given the 5 base rankers as inputs, we compare the 6 rank aggregation methods described above. Since we are trying to establish if rank aggregation can further improve results, we pick the pruning setting with the best individual ranker performance, i.e. pruning at 20 for Medical and 5 for Jeopardy. The results of these experiments are presented in Table 4. Following Matveeva et al. ([23]), any metric@K for datasets pruned@N when $K > N$, was computed by appending the datasets with the next $K-N$ examples from the unpruned but ranked data. For example, we appended Jeopardy pruned@5 with the 5 next examples from the logistic regression

Algorithm 2 Supervised Borda

Input: $\tau_i = [\tau_{i1}, ..., \tau_{im}], \forall i = 1, ..., r$, ordered arrangement of m candidates for r rankers.
$w = [w_1, ..., w_r]$ – where w_i is the weight for ranker i
Output: τ – rank aggregated arrangement of candidates in decreasing order of importance

1. Initialize $\beta_i \Leftarrow 0, \forall i = 1, ..., m$, β_i is the borda score of candidate i
2. For each ranker $p = 1$ to r
3. For each candidate $i = 1$ to m
4. $\beta_i \Leftarrow \beta_i + ((m - i + 1)w_p)$
5. Sort β_i in descending order, such that $\tau = [d_1, ..., d_m], \beta_{d_i} \geq \beta_{d_j}, \forall i, j = 1, ..., m, i \neq j$
6. Return τ

pruning step to compute NDCG@10. This is reasonable for an application where K results must be shown. In addition to individual ranker performance, we also present the mean performance across rankers. It is clear from the results that all rank aggregation techniques perform better than the mean performance of rankers, both in terms of P@1 and NDCG@10. While better than the mean, the unsupervised aggregation techniques are outperformed by the best single ranker. Additionally, note that the best individual ranker is different for each corpus; while it is logistic regression for Jeopardy, it is LambdaRank for Medical. So in the absence of hindsight, it is always beneficial to use rank aggregation for more robust results, with lower variance.

The unsupervised aggregation techniques implicitly assume all rankers are equally valuable; as such the performance of the best ranker may be pulled down by other rankers. However, by incorporating performance-based weights for each ranker, Supervised Kemeny Ranking is able to do even better than the best individual rankers in terms of P@1 without loss on NDCG@10. Since we are already aggressively pruning to the top 20 and top 5 in these datasets, further focusing on the top 50% of these candidates, as done in Kemeny Top-k does not improve results. Also Borda, the weaker aggregation technique, is not competitive even with supervision. See [30] for a deeper analysis of Kemeny versus Borda.

Based on these results, we recommend always using multiple base rankers and supervised rank aggregation, rather than relying on the best individual ranker.

7. DISCUSSION

Figure 3 shows our proposed system with its different stages. The initial set of candidates are ranked by logistic regression and then the top N candidates are re-ranked by different rankers. The output of the rankers is aggregated and the top K results as seen fit for the application are shown to the user. Since logistic regression and the different re-rankers use the same feature-set in our setup and only the top N (20 or less) candidates are re-ranked, the overhead due to the additional stages is minimal.

Table 5 summarizes the performance of our proposed system in comparison with the baseline logistic regression. We also report two new metrics in this table: RR@5 and RR@10. The reciprocal rank (RR) is the average of the reciprocal of the rank at which the first correct answer is found. RR@K limits the ranked list per question to K. All metrics show improvement on both datasets. P@1

Metrics	Jeopardy		Medical	
	Baseline	**Supervised Kemeny** (Pruned @5)	**Baseline**	**Supervised Kemeny** (Pruned @ 20)
P@1	69.0	72.4	43.3	45.5
NDCG@5	79.3	80.8	50.5	51.3
NDCG@10	80.9	82.4	54.6	55.7
RR@5	77.9	80.0	53.6	54.8
RR@10	78.4	80.5	54.7	56.2

Table 5: Supervised Kemeny vs baseline. An underline indicates significance with a paired t-test at the 0.05 level.

on Medical is significant at the 0.1 level (p=0.09) and NDCG@10 and RR@10 are significant at the 0.05 level. For our Medical application, the P@1 improvement of 2.2% points is substantial in practice. However, it appears that our test set was not large enough to establish statistical significance here. On the larger Jeopardy test set, all metrics show significant improvements even at the 0.01 level. Our final system therefore always shows improvement regardless of the performance of the individual rankers, and thus an application developer can use it as a black box without evaluating individual rankers. Rank aggregation therefore provides for a robust system combination strategy that we hope to test on other domains that Watson can be applied to.

8. ADDITIONAL EXPERIMENTS ON TREC DATA

In addition to our application domains of Jeopardy! and Medical, we also ran experiments on the publicly-available TREC Question Answering datasets. The TREC tracks have largely focused on factoid questions e.g. "What is David Lee Roth's birthday?" We used data from the TREC 8-10 and TREC 12 evaluations [36] available at the NIST website (http://trec.nist.gov/data/qamain.html). We only considered questions that had at least one candidate answer from the first pass retrieval stage, giving a total of 1,128 questions of which 794 were randomly partitioned into the training set and the remaining 334 made the test set. On average both the training and test sets contain 2 correct answers per question from about 93 candidate answers retrieved from the initial retrieval.

In addition to the features used for the Jeopardy! dataset, 3 domain adaptation changes were made: first, the question analysis component was modified to address the differences in question formats between Jeopardy! and TREC. Second, we included Ephyra [26] and PIQUANT [9] question answer systems' static type based candidate generation components. Third, we included the AQUAINT corpus, the 3 GB news wire corpus used in TREC evaluation, in our searches. The domain adaptation resulted in 6 new features over those that were used on Jeopardy. Previous work [15] has shown the competitiveness of these additional features on top of the Jeopardy features for the TREC task.

Table 6 shows the performance of individual rankers with a pruning setting of 50. The learning to rank methods are quite competitive on both P@1 and NDCG@10 with LambdaRank equaling the performance of logistic regression on P@1 and Coordinate-Ascent being slightly better. All aggregation methods show improvement over the mean performance of individual rankers. As before, we see that the best performance is produced by Supervised Kemeny Ranking, both in terms of P@1 and NDCG@10. Table 7 compares Supervised Kemeny with a pruning threshold of 50 to the baseline logistic regression trained on all candidates. The results show that we achieve consistent improvements over the baseline on 5 differ-

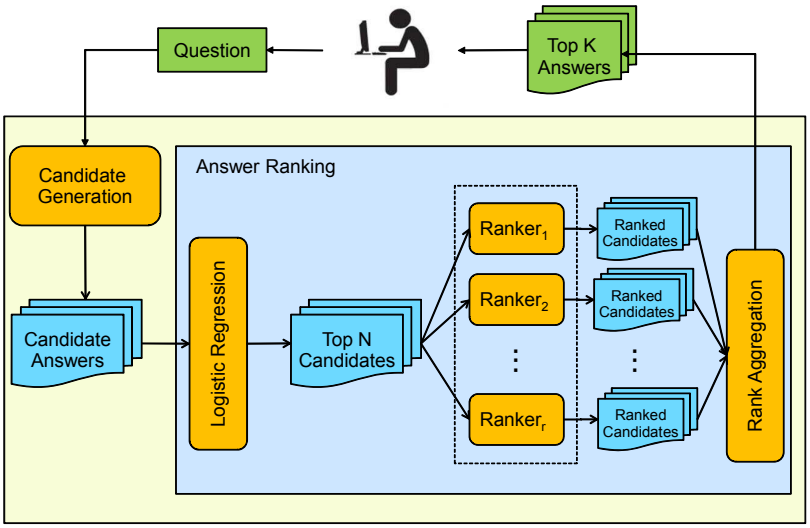

Figure 3: Our system pipeline.

Metrics	TREC	
	Base-line	**Supervised Kemeny** (Pruned @50)
P@1	65.9	66.8
NDCG@5	66.6	67.2
NDCG@10	70.6	71.1
RR@5	73.8	74.8
RR@10	74.4	75.4

Table 7: Supervised Kemeny vs baseline on TREC data.

ent metrics. In additional experiments on testing different pruning settings we find that, while Supervised Kemeny improves over the baseline, it is not always the approach with the highest performance. However, the best performance, in general, is still achieved by one of the rank aggregation approaches.

9. RELATED WORK

Related work in the area of QA and learning to rank has been already introduced at various points. In this section we compare our work with relevant work in learning to rank for QA. Although learning to rank methods have extensively been applied to web search, the application to QA has been limited. In fact, to the best of our knowledge, Verberne et al. ([35]) is the only work where learning to rank methods have actually been applied to the QA task with the focus on the learning component – unlike others [3, 31] where the focus has been on feature generation. Although similar in spirit, Verberne et al. only study the behavior of various learning to rank methods on the QA task; while in this work, we not only study this behavior, but also provide a robust multi-stage system that is able to improve over the results obtained by simply applying learning to rank methods. Additionally, their work was restricted to *why* questions while we work on *factoid* questions. We note that our findings from the ranking stage of this multi-stage system corroborate the observations of Verberne et al., that pairwise and listwise methods are not always superior to pointwise methods.

10. CONCLUSION

We have presented a multistage approach to learning to rank candidate answers in a QA system. The final system proposed is efficient and robust, giving significant improvements over the state-of-the-art Watson baseline, both in open-domain QA as well as in the specialized discipline of Medicine. Our approach is also easily extensible, in that, many more base rankers may be added to our aggregation, which may lead to further improvements. While we evaluated our methods on candidate answers generated by Watson, the results are applicable to any QA system, and may be more broadly applicable to other rank-learning tasks.

11. ADDITIONAL AUTHORS

Additional authors: James Fan(IBM T. J. Watson Research Center, email: `fanj@us.ibm.com`).

12. REFERENCES

[1] K. Arrow. Social choice and individual values. New Haven: Cowles Foundation, 2nd Edition 1963.

[2] J. A. Aslam and M. Montague. Models for metasearch. In *Proceedings of the International ACM SIGIR conference on Information Retrieval*, pages 276–284, 2001.

[3] M. Bilotti, J. Elsas, J. Carbonell, and E. Nyberg. Rank learning for factoid question answering with linguistic and semantic constraints. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 459–468, 2010.

[4] J. Borda. Memoire sur les elections au scrutin. In *Histoire de l'Academie Royale des Sciences*, 1781.

[5] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the International Conference on Machine learning*, pages 89–96, 2005.

Datasets	Rankers						Aggregation					
							Unsupervised			Supervised		
	LR	Rank Boost	AdaRank	Coord Ascent	Lambda Rank	Mean	Borda	Kemeny	Kemeny Top-k	Borda	Kemeny	Kemeny Top-k
TREC	65.0	62.6	65.9	66.2	65.0	64.9	65.3	65.9	66.5	65.6	**66.8**	**66.8**
(pruned@50)	70.1	66.4	70.6	70.4	69.9	69.5	70.4	70.6	70.7	70.6	**71.5**	71.1

Table 6: Performance of rank-aggregation methods along with the learning to rank methods on TREC. The top entry in each cell is Precision@1, while the bottom is NDCG@10. The best results for each metric are presented in bold.

[6] C. J. C. Burges, K. M. Svore, P. N. Bennett, A. Pastusiak, and Q. Wu. Learning to rank using an ensemble of lambda-gradient models. In *Journal of Machine Learning Research*, pages 253–35, 2011.

[7] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the International Conference on Machine learning*, pages 129–136, 2007.

[8] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *the Journal of Machine Learning Research - Proceedings Track*, 14:1–24, 2011.

[9] J. Chu-Carroll, P. A. Duboué, J. M. Prager, and K. Czuba. Ibm's piquant ii in trec 2005. In *TREC*, 2005.

[10] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems 14*, pages 641–647, 2001.

[11] V. Dang. Ranklib - a library of ranking algorithms. `http://www.cs.umass.edu/~vdang/ranklib.html`.

[12] C. Dwork, R. Kumar, R. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *the International World Wide Web Conference*, pages 613–622, 2001.

[13] D. Ferrucci. Building watson: An overview of the deepQA project. `http://techtalks.tv/talks/54458/`, 2011.

[14] D. Ferrucci, A. Levas, S. Bagchi, D. Gondek, and E. Mueller. Watson: Beyond Jeopardy! *Artificial Intelligence (to appear)*, 2012.

[15] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.

[16] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *The Journal of Machine Learning Research*, 4:933–969, 2003.

[17] D. C. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. A. Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty. A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development*, 56(3.4):14:1 –14:12, may-june 2012.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[19] A. Ittycheriah and S. Roukos. IBM's statistical question answering system - trec-11. In *Proceedings of the Text REtrieval Conference*, 2001.

[20] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[21] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002.

[22] J. Kemeny. Mathematics without numbers. In *Daedalus*, volume 88, pages 571–Ű591, 1959.

[23] I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. High accuracy retrieval with multiple nested ranker. In *Proceedings of the International ACM SIGIR conference on Information Retrieval*, pages 437–444, 2006.

[24] D. Metzler and B. Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

[25] R. Nallapati. Discriminative models for information retrieval. In *Proceedings of the International ACM SIGIR conference on Information Retrieval*, pages 64–71, 2004.

[26] S. Nico, J. Ko, J. Betteridge, M. Pathak, E. Nyberg, E, and G. Sautter. Semantic extensions of the ephyra qa system for trec 2007. In *TREC-07*, 2007.

[27] J. Prager, E. Brown, A. Coden, and D. Radev. Question-answering by predictive annotation. In *Proceedings of the International ACM SIGIR conference on Information Retrieval*, pages 184–191, 2000.

[28] C. Quoc and V. Le. Learning to rank with nonsmooth cost functions. In *Proceedings of the Advances in Neural Information Processing Systems 19*, pages 193–200, 2007.

[29] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: a language-model based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, 2005.

[30] K. Subbian and P. Melville. Supervised rank aggregation for predicting influence in networks. In *Proceedings of the IEEE International Conference on Social Computing*, 2011.

[31] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics*, 37(2):351–383, 2011.

[32] J. Suzuki, Y. Sasaki, and E. Maeda. Svm answer selection for open-domain question answering. In *Proceedings of the 19th international conference on Computational linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[33] M. Truchon. An extension of the condorcet criterion and kemeny orders. In *J. Eco. Lit.*, 1998.

[34] M. Tsai, T. Liu, T. Qin, H. Chen, and W. Ma. Frank: a ranking method with fidelity loss. In *Proceedings of the International ACM SIGIR conference on Information Retrieval*, pages 383–390, 2007.

[35] S. Verberne, H. van Halteren, D. Theijssen, S. Raaijmakers, and L. Boves. Learning to rank for why-question answering. *Information Retrieval*, 14(2):107–132, 2011.

[36] E. M. Voorhees. Overview of the TREC 2003 question answering track. pages 54–68, 2003.

[37] J. Xu and H. Li. Adarank: a boosting algorithm for

information retrieval. In *Proceedings of the International ACM SIGIR conference on Information Retrieval*, pages 391–398, 2007.

[38] H. Young and A. Levenglick. A consistent extension of condorcet's election principle. In *SIAM Journal on Applied Mathematics*, volume 25, pages 163–177, 1978.

[39] H. P. Young. Condorcet's theory of voting. In *American Political Science Review*, volume 82, pages 1231–1244, 1998.

[40] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proceedings of the International ACM SIGIR conference on Information Retrieval*, pages 271–278, 2007.