UMD and USC/ISI: TREC 2010 Web Track Experiments with Ivory

Tamer Elsayed,[1] Nima Asadi,[1] Donald Metzler,[2] Lidan Wang,[1] Jimmy Lin[1]
[1]University of Maryland, College Park
[2]Information Sciences Institute, University of Southern California
{telsayed,nima,lidan}@cs.umd.edu, metzler@isi.edu, jimmylin@umd.edu

ABSTRACT

Ivory is a web-scale retrieval engine we have been developing for the past two years, built around a cluster-based environment running Hadoop, the open-source implementation of the MapReduce programming model. Building on successes last year at TREC, we explored two major directions this year: more sophisticated retrieval models and large-scale graph analysis for spam detection. We describe results of *ad hoc* retrieval experiments with latent concept expansion and a greedily-learned linear ranking model. Although neither model is novel, our experiments provide some insight on the behavior of these two approaches at scale, on collections larger than those previously studied. We also discuss our link-based spam filtering algorithm that operated on the entire web graph of ClueWeb09. Unfortunately, results in the spam track were worse than the baseline provided by the track organizers.

1. INTRODUCTION

Compared to research in industry labs and search engine companies, academic information retrieval faces two major challenges: access to data and access to computing resources. The first issue has been alleviated to some extent by the availability of ClueWeb09,[1] a recent web crawl by CMU comprising approximately one billion pages. Indeed, the dataset has allowed researchers to explore, through the TREC framework, retrieval at web scale.

Clearly, web-scale datasets cannot be easily and efficiently processed by individual machines, necessitating bringing to bear the computation capacity made available by clusters. Several convergent trends have improved researchers' access to computing resources. At the hardware level, falling prices of commodity severs and alternative forms of access such as utility computing have made computing resources more affordable than ever. At the software level, emerging frameworks for large-scale distributed programming such as MapReduce [6] and DryadLINQ [26] allow IR researchers to focus on solving IR problems, as opposed to wrestling with system-level details such as scheduling and synchronization.

For the past two years, we have been rethinking various aspects of IR in the context of a cluster-based environment built around Hadoop, an open-source implementation of MapReduce. Last year, we successfully developed, largely from scratch, a web-scale retrieval engine called Ivory [13]. Accomplishments included a scalable, distributed indexer based on MapReduce [7], a novel retrieval model whereby postings are directly fetched from the Hadoop Distributed File System (HDFS), and integration of web page classifiers for quality, spam, and adult content. Ivory was effective in the *ad hoc* task in the TREC 2009 web track, on both the 50 million and 502 million subsets of ClueWeb09.

Building on Ivory and the successes of last year, we explored two major directions for TREC 2010: more sophisticated retrieval models and large-scale link analysis for spam detection. This paper is organized as follows: Section 2 describes improvements we made in the Ivory infrastructure. Section 3 discusses our *ad hoc* retrieval experiments with latent concept expansion and a greedily-learned linear ranking model. Although neither model is novel, our experiments provide some insight on the behavior of these two approaches at scale, on collections larger than those previously studied. Section 4 describes our link-based spam filtering algorithm that operates on the entire web graph of ClueWeb09. Unfortunately, our results were worse than the baseline provided by the track organizers. We conclude in Section 5.

2. IVORY ENHANCEMENTS

We begin with a brief overview of indexing and retrieval in the previous implementation of Ivory deployed in TREC 2009 [13]. In the indexing process, the collection is divided into smaller document partitions; each is indexed separately, producing a document-partitioned index. The English portion of ClueWeb09, used in the evaluations, is divided into ten segments, each of which serves as a partition. Our inverted indexing algorithm is written in MapReduce: documents are processed in the mappers, which emit postings. Reducers receive postings grouped by term, and then write compressed postings lists to disk. For retrieval, we adopted a broker-mediated architecture: the user sends a query to the broker, which then forwards it to every partition server. Each server returns a ranked list over documents in its partition to the broker, which merges all of the ranked lists to produce the final results.

This year, we added a preprocessing stage prior to indexing. This stage consists of three major steps, all implemented as MapReduce jobs. In the first step, all documents are parsed into document vectors (with stemming and stopword removal), represented as associative arrays from terms to term frequencies (*tfs*). At the same time we build a table of document lengths, necessary for retrieval later. In the second step, we construct a mapping from terms to integers (term ids), sorted by descending document frequency (*df*), i.e., term 1 represents the term with the highest *df*, term 2 represents the term with the second highest *df*, etc. During

[1]http://boston.lti.cs.cmu.edu/Data/clueweb09/

this process, we discard all terms that occur ten or fewer times in the collection, since these rare terms are unlikely to be part of real-world user queries. The resulting dictionary is then compressed with front-coding [22]. Finally, in the third step a new set of document vectors are generated in which terms are replaced with corresponding integer term ids. Furthermore, within each document the terms are sorted in increasing term id, so that we are able to encode gap differences (using γ codes). The final result is a compact representation of the original document collection.

Why the addition of this preprocessing stage? Why go through the extra steps of materializing the document vectors? This is necessary to support relevance feedback, e.g., latent concept expansion (see Section 3.2). Relevance feedback requires access to document contents, so the two options are to reparse the documents on-the-fly or to store the document vectors for easy access. Storing document vectors with the actual vocabulary is inefficient, so this is why we create document vectors with terms represented by integers. Only these integerized document vectors are used in the retrieval process, but term document vectors may be useful for independent reasons.

3. AD HOC RETRIEVAL

3.1 Term proximity linear ranking models

This year, in addition to the traditional baseline BM25 model, we explored various linear models for ranked retrieval. Linear ranking models offer a simple, yet principled way to combine different query-document features to score documents effectively. Many widely used ranking models belong to this family of ranking functions [17, 1, 9]. Broadly, a linear ranking model is specified by a set of features $F = f_1, \ldots, f_N$ and the corresponding model parameters $\Lambda = \lambda_1, \ldots, \lambda_N$. Each feature f_i is a function that maps a query-document pair (q, d) to a real value. The relevance score of document d with respect to query q is computed as:

$$\text{Score}(q, d) = \sum_i \lambda_i f_i(q, d)$$

From this general form, different linear models can be instantiated by specifying their corresponding feature sets and the associated weights. Among the possible linear model instantiations, we restricted our attention to the *sequential dependence model* (SD) [17] and the *weighted sequential dependence model* (WSD) [1]. Both of these models employ a combination of term-based and term proximity features to score documents. In addition to these features, since web documents vary in quality, we also employed a document-dependent spam feature provided by the University of Waterloo [5]. The complete feature set used by our linear models is listed below:

- Number of occurrences of each unigram in the document (weighted by BM25)
- Number of occurrences of the exact phrase "$q_j\ q_{j+1}$" in the document (weighted by BM25)
- Number of occurrences of an unordered window containing $q_j\ q_{j+1}$ (window span = 8) in the document (weighted by BM25)
- Waterloo spam score for each document [5]

Feature	Description
$g_1^t(q)$	# times q occurs in the collection
$g_2^t(q)$	# documents q occurs in the collection
$g_3^t(q)$	# times q occurs in ClueWeb09
$g_4^t(q)$	# times q occurs in a Wikipedia title
$g_5^t(q)$	1 (constant feature)
$g_1^b(q_j, q_{j+1})$	# times bigram occurs in the collection
$g_2^b(q_j, q_{j+1})$	# documents bigram occurs in the collection
$g_3^b(q_j, q_{j+1})$	# times bigram occurs in ClueWeb09
$g_4^b(q_j, q_{j+1})$	# times bigram occurs in a Wikipedia title
$g_5^b(q_j, q_{j+1})$	1 (constant feature)

Table 1: Meta-features in the WSD model.

There are certainly other ways to define term-based and term proximity features. However, the SD and WSD models serve as strong baselines given previous results on TREC datasets [17, 1].

In the case of the SD model, the weight on a particular feature f_i depends on its *type*. In our experiments, all term occurrence features receive a weight of 0.82, and the term proximity features (e.g., exact phrase and unordered window features) receive a weight of 0.09. These values reflect best-practice settings and have been used in our TREC runs from last year [13]. The document-dependent spam feature receives a weight of 0.02, which is learned from a line search—given the specified SD feature weights, we scan a set of possible spam weights ranging from 0 to 1 in increments of 0.01, and select the spam weight that results in maximum MAP score when combined with the SD model.

An advantage of the parameter tying between the query-document features in the same type is simplicity in the ranking model. A drawback is that it cannot differentiate between "important" query concepts and less important query concepts, since they will be assigned the same weight in the ranking model if they are in the same feature class. For instance, for the query "Shenandoah Valley Tourist Attractions", the query concepts "Shenandoah Valley" and "Valley Tourist" clearly have different importance. Intuitively, we would consider "Shenandoah Valley" to be more important than "Valley Tourist". However, the SD model treats them as equally important, by assigning their term proximity features the same weight in the ranking model. The *weighted sequential dependence model* (WSD) [1] improves on the SD model by letting the feature weights (for query-dependent features) vary with the query concepts, such that features are assigned weights in accordance with the importance of the concepts they are defined over. Formally, the weight λ_i of feature $f_i(q)$ takes a parametric form:

$$\lambda_i(q) = \sum_j w_j\ g_j(q)$$

where g_j's are meta-features defined over the query for each feature i, and w_j's are the free parameters. Hence, λ_i depends on q via g_j and w_j. For the query-dependent meta-features g_j, we use both collection features (collection frequency and document frequency) and features from external sources (English Wikipedia and ClueWeb09). The meta-features are summarized in Table 1.

The free parameters w_j's in the WSD model are trained on the first segment of the ClueWeb09 collection using all

50 TREC 2009 web track queries. We employed a simple line search to identify the values of the parameters [19]. This method iteratively optimizes the retrieval metric by performing a series of one-dimensional searches. At each iteration, the optimal value for a parameter is discovered while holding all other parameters fixed. The process continues until the improvement in the objective metric drops below a threshold. Given the trained values for the w_j's, we then learned an optimal weight for the document spam feature (0.03 in this case).

3.2 LCE

Another retrieval model we explored this year is the *latent concept expansion model* (LCE) [18]. LCE is a robust query expansion model that provides a mechanism for modeling term dependencies in query expansion. Since it is based on the Markov Random Field (MRF) framework [17], it allows us to incorporate a wide range of retrieval features. In our experiments, we considered latent unigram concepts only. We used our trained WSD model as the baseline, and let the LCE model score the unigram concepts contained in the top 5 documents retrieved by WSD, and selected the top 4 terms with the highest scores to form an expanded WSD model. The weights of these unigram expansion features were computed in the same way as the WSD query-dependent features (e.g., as parametric functions of the meta-features described in Section 3.1).

3.3 Learning to Rank

We also explored the use of a simple learning-to-rank approach [15] this year. Machine learning approaches have been shown to be effective for learning highly effective ranking functions, but lack of sufficient training data limits their applicability in the TREC context. We used the TREC 2009 web track *ad hoc* queries and judgments as our training data. Given their small size, we were forced to use a relatively simple model with a small number of features to avoid over-fitting.

We used the same linear ranking function defined earlier in Section 3.1:

$$\text{Score}(q, d) = \sum_i \lambda_i f_i(q, d)$$

where λ_i's are the model parameters we need to estimate from the training data.

By limiting the complexity of the model, we discourage over-fitting. In the future, we are interested in exploring automatic methods for obtaining large amounts of (possibly noisy) relevance judgments that can then be used with more sophisticated learning to rank approaches, such as LambaRank [2] or gradient boosting [27].

We were primarily interested in three types of features that are currently supported by Ivory. These include:

- Basic IR Scores: These features are simply the scores computed using traditional IR models. The two features of this type we used are the BM25 score and the language modeling retrieval score (specifically, query likelihood with Dirichlet smoothing).
- Term Proximity Features: Term proximity has been shown to be important for effective retrieval, especially for very large collections such as the web. Therefore, we considered a large family of term proximity scores as features. We used various combinations of ordered and unordered windows, with different window sizes and scoring approaches (i.e., BM25 and language modeling). Additional details can be found in [16].
- Document Features: Query-independent features are useful for inferring the *a priori* importance of a given document. The document features that we computed are Anti-TrustRank (see Section 4 for more details), the Waterloo spam score [5], and PageRank [21].

Feature	Weight
BM25(q, d)	0.7294
BM25$_{bigram}(q, d)$	0.0616
BM25$_{pairs}(q, d)$	0.0433
Spam(d)	0.0674
AntiSpam(d)	0.0915
PageRank	0.0069
All other features	0.0

Table 2: Learned parameter values for the learning to rank model.

This results in a total of 45 features (2 basic, 40 proximity, and 3 document). It should be noted that field-specific text matching scores (e.g., anchor text match, title match, etc.) were omitted because Ivory currently does not support indexing semi-structured documents.

The model parameters (i.e., the λ_i weights) are learned using a greedy feature selection strategy [16]. In this approach, features are added to the model, one at a time, according to a greedy selection criterion. At each iteration, the feature that yields the largest gain in effectiveness (as measured by ERR [3]) after being added to the existing model is selected. This results in a sequence of one-dimensional optimizations that can be solved using simple line search techniques. The learning algorithm halts when the difference in ERR between successive iterations drops below a given threshold (10^{-4}). This training procedure is simple, fast, and yields a model with minimal correlation/redundancy between features.

Table 2 shows the values learned for the model parameters from the 2009 TREC web track *ad hoc* queries and relevance judgments, where BM25(q, d) is the traditional BM25 score, BM25$_{bigram}(q, d)$ is the BM25 score of the query bigrams, BM25$_{pairs}(q, d)$ is the BM25 score of *all* pairs of query terms, and Spam, AntiSpam, and PageRank are the three document features described previously.

It is interesting that the learned model contains just 6 features among a total of 45. The greedy feature selection strategy determined that the remaining features were mostly redundant, and thus not worth including in the model. The selected features do, in fact, capture different aspects of relevance, such as a term score (BM25), a phrase score (BM25$_{bigram}$), a proximity score (BM25$_{pair}$), and various query-independent scores.

3.4 Evaluation

We now provide an empirical evaluation of our experiments. All of the model parameters used in our runs were tuned using the TREC 2009 web track data. Furthermore, all statistical significance testing was performed using a one-tailed paired t-test at the $p < 0.05$ level.

Run ID	P@10	MAP	NDCG@20	ERR@20
IvoryBM25a	0.2313	0.1008	0.1183	0.0693
IvorySDa	0.2458	0.0992	0.1315	0.0711
IvoryWSDa	0.3354	0.1142	0.1579	0.0860
IvoryWSDb	0.3625	0.1309	0.1975	0.1045
IvoryLCEb	0.3937	0.1370	0.2143	0.1127
IvoryL2Rb	0.4000	0.1333	0.2255	0.1340

Table 3: Summary of *ad hoc* task results.

3.4.1 Ad Hoc Task

Table 3 summarizes the results of our *ad hoc* retrieval runs. The run identifiers denote the retrieval method (e.g., BM25, SD, WSD, LCE, and L2R for "learning to rank") and the subset of ClueWeb09 documents that were used for the run (the suffix "a" denotes Category A while the suffix "b" denotes Category B). The effectiveness of the runs are evaluated in terms of precision at 10 (P@10), mean average precision (MAP), normalized discounted cumulative gain at 20 (NDCG@20), and expected reciprocal rank at 20 (ERR@20).

The results show a natural progression of effectiveness from our baselines (IvoryBM25a and IvorySDa) to our more sophisticated approaches (IvoryLCEb and IvoryL2Rb). A deeper analysis reveals our results can be broken into three tiers of effectiveness. The bottom tier consists of BM25 and SD, which are statistically indistinguishable across all metrics. The WSD model, which is statistically significantly better than both BM25 and SD in terms of ERR@20, occupies the middle tier. Finally, LCE and L2R make up the top tier, as the two models are statistically indistinguishable from one another, but statistically significantly better than the WSD model with respect to ERR@20.

Our results also suggest that better effectiveness can be achieved by only retrieving documents from Category B of ClueWeb09. Similar results were also observed by ourselves and others at the TREC 2009 web track. These findings are in line with the analysis of the relative quality of Category A vs. Category B that we undertook previously [13]. Our analysis showed that Category B generally had higher quality documents, less spam, and fewer adult pages, and thus was generally a better source of relevant documents.

It is also interesting to note that our LCE run, which used the WSD model with a spam feature as its base ranking function, was highly effective. Previous attempts to apply pseudo-relevance feedback to web search have yielded inconsistent results. Our preliminary investigation suggests that highly-focused pseudo-relevance feedback (i.e., a few expansion terms selected from a few top-ranked documents), combined with an effective base ranking function, can be effective for web search. Additional analysis and experimentation are necessary to fully understand the observed effects.

Our most effective run, IvoryL2Rb, was a simple machine-learned ranking function that uses only 6 features. There are various ways that one could improve upon this model, such as using additional features (e.g., those based on anchor text and document structure), and perhaps even combining LCE, or some other form of pseudo-relevance feedback, with learning to rank.

Now that we have described the positive aspects of our runs, we briefly describe several cases where our ranking models failed. In particular, there were four topics that none of our runs retrieved any relevant documents for. The four topics were "to be or not to be that is the question" (topic 70), "kiwi" (topic 74), "the wall" (topic 92), and "titan" (topic 94).

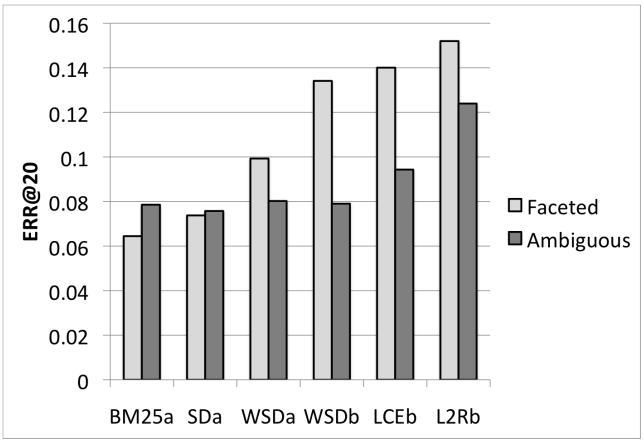

Figure 1: Comparison of ERR@10 for faceted and ambiguous queries across runs.

3.4.2 Query Types and Diversity

Each of the topics used in this year's track were manually annotated as being either "faceted" or "ambiguous". Faceted topics have a well-defined focus. The "facets" arise from the fact that different users may be interested in different aspects of the general focus. For example, "horse hooves" (topic 51) has facets that include "caring for hooves", "pictures of horse hooves", "diseases of horse hooves", and "anatomy of horse hooves". Ambiguous topics are less well-defined and it may be difficult to infer the actual intent from a single query alone. For example, "avp" (topic 52) can refer to "Association of Volleyball Players", "AVP anti-virus software", "Avon products company", and so on.

Figure 1 compares the effectiveness of our runs for faceted and ambiguous queries. The results suggest that our baseline runs (BM25 and SD) are actually more effective on ambiguous queries than faceted ones, whereas our more advanced runs (WSD, LCE, and L2R) are markedly better for faceted queries. The WSD-based runs (WSDa, WSDb, and LCEb) show substantial gaps between faceted and ambiguous queries, which suggests that the methods may not adequately account for query ambiguity. On the other hand, the learning to rank approach shows a significant increase in ambiguous query effectiveness over these approaches, although the gap between ambiguous and faceted queries still remains.

Finally, we evaluated the effectiveness of our runs with respect to several diversity-aware metrics. It is important to note that we did not officially participate in the diversity task and none of our runs perform any diversity-specific processing. We include these results for completeness and to better understand the strengths and weaknesses of our approaches.

The results of the diversity evaluation are provided in Table 4. The diversity results closely track the *ad hoc* results presented in Table 3. It is unclear if this is the result of intrinsic correlations between diversity-agnostic and diversity-

Run ID	ERR-IA@20	α-NDCG@20	NRBP
IvoryBM25a	0.1448	0.2300	0.1049
IvorySDa	0.1459	0.2284	0.1087
IvoryWSDa	0.1716	0.2601	0.1320
IvoryWSDb	0.2214	0.3277	0.1777
IvoryLCEb	0.2201	0.3169	0.1799
IvoryL2Rb	0.2847	0.3934	0.2468

Table 4: Summary of diversity task results.

aware metrics or if our more advanced approaches implicitly account for diversity in some way.

4. SPAM FILTERING

4.1 Model

Given the large size of the ClueWeb09 collection, spam has led to rising concerns about the effectiveness of classic retrieval methods. Cormack et al. [5] presents a simple, minimally-trained, content-based classifier that post-processes ranked lists to yield significant improvements on a variety of metrics. In an attempt to further explore the structure of spam in ClueWeb09 and to properly counterbalance its effect on retrieval, we tackled this problem from a different approach.

Web graphs provide important information about their components' interconnections and there has been a broad range of studies [10, 12, 25, 24, 8, 28, 23] on how to exploit link-based characteristics to detect more sophisticated forms of spam, e.g., link spamming. Link spamming involves boosting the rank of certain pages by setting up specific link structures among them. It cannot be properly modeled by content-based filters, as these filters are generally designed to recognize term spam (i.e., misleading document content) [8, 28, 20].

Gyöngyi et al. [10] exploits the intuition that good pages, i.e., those of high quality, are unlikely to point to low quality or spam pages. Having collected a set of good pages as its seed in a supervised manner, their proposed algorithm, called *TrustRank*, then propagates "trust" throughout the web graph. The propagation is done iteratively with a biased PageRank algorithm. Pages are then labeled by applying a certain threshold to the TrustRank scores. Krishnan and Raj [12], on the other hand, address the problem in a slightly different way and show their proposed algorithm outperforms TrustRank. Following the intuition that spam pages are unlikely to be pointed to by good pages, their algorithm, called *Anti-TrustRank*, starts with a set of spam pages as its seed and then runs a biased PageRank to propagate "anti-trust" in the reverse direction. Wu et al. [24] show that combining trust and anti-trust (i.e., distrust) is more effective than using only trust scores. Given the trust and distrust values, they propose a linear combination in the form below for a page P_i as a way to take into account both properties:

$$\text{Score}(P_i) = \alpha \cdot \text{Trust}(P_i) - \beta \cdot \text{AntiTrust}(P_i)$$

where $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$ are two coefficients that give different weights to trust and anti-trust scores.

Despite the fact that these two approaches are mathematically well-formulated and easy-to-implement, there are certain drawbacks to consider. As discussed by Gyöngyi et al., selection of seed pages can skew the final scores, which, after applying the threshold, will alter the effectiveness of the filter. Moreover, in most cases a set of seed pages does not guarantee full coverage of the entire web graph. This gap in coverage comes from the possibility that certain parts of the graph might not be reachable from the seeds. To reduce the effect of these problems, Jiang et al. [11] suggest the use of a large, automatically-generated seed set, as opposed to a small, carefully-selected seed set.

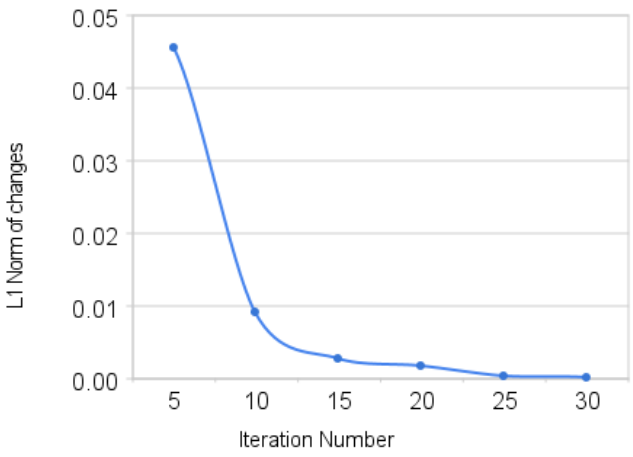

Figure 2: Convergence of spam scores.

Instead of developing our own classifier, our goal for the spam task was to use Wu et al.'s approach [24] by interpolating between TrustRank and Anti-TrustRank scores for each page in the collection. Additionally, we studied the effects of seed page selection. After extracting the complete web graph from the ClueWeb09 collection, we developed our MapReduce implementation of PageRank [14] in order to have the necessary tools to efficiently run the TrustRank and Anti-TrustRank algorithms. In our experiments, we limit the number of iterations to 30. Based on changes between the iterations measured by the difference of the L1 norm of the score vectors, we can detect algorithm convergence. Figure 2 plots the convergence (difference in L1 norm of score vectors) versus the iteration number.

To compute an interpolated score, we assign a value of 1 to both coefficients α and β in the equation above. The page scores are converted to a percentile rank for the purpose of comparing effectiveness [5].

TrustRank creates a seed set by manually labeling pages with high PageRank scores in the inverse web graph [10], while in Anti-TrustRank, pages with high PageRank scores in the original web graph are labeled manually [12]. For TrustRank, good pages are used as the seed set while in Anti-TrustRank, spam pages are used as the seed set. Due to the large size of our web graph, in lieu of manually labeling the pages, we define thresholds θ_{Trust} and $\theta_{\text{AntiTrust}}$ and apply them to Waterloo spam scores [5] to generate seed sets. Pages with a spam score of θ_{Trust} or higher are then used as the seed pages for TrustRank and pages with a spam score of $\theta_{\text{AnitTrust}}$ or lower are chosen to form the seed set for Anti-TrustRank. This allows us to easily generate larger seed sets that cover larger portions of the web graph.

In our experiments we used the pairs $(70, 30)$ and $(90, 10)$

System	ham%	spam%	lam%	1-ROCA%
Baseline	24.32	18.60	21.36	13.6051
IVORY.70.30	29.49	20.93	25.00	21.52
IVORY.90.10	33.11	27.44	30.23	19.36

Table 5: Summary of spam filtering results.

as values for $(\theta_{\text{Trust}}, \theta_{\text{AntiTrust}})$. The first pair produces a larger seed set but leaves fewer unlabeled pages. However, the tradeoff is, as we lower θ_{Trust} and raise $\theta_{\text{AntiTrust}}$, although we obtain larger seed sets, we increase the chance of having inaccurate seed sets, i.e., a good seed set that might contain spam pages and a bad seed set that might contain good pages.

4.2 Evaluation

We now provide an evaluation of our spam filters. Evaluation data provided by NIST use the metrics introduced by Cormack et al. [4]. More specifically, *ham%* is the percentage of misclassified non-spam pages (i.e., non-spam pages that are classified as spam), *spam%* is similarly the percentage of misclassified spam pages (i.e., spam pages that are classified as non-spam), and, *logistic average misclassification percentage* (lam%) defined as

$$lam\% = logit^{-1}\left(\frac{logit(ham\%) + logit(spam\%)}{2}\right)$$

where $logit(x) = log(\frac{x}{100\%-x})$ [4]. Additionally, since both *ham%* and *spam%* measure failure rather than effectiveness, the spam track reports the area above the Receiver Operating Characteristic (ROC) curve. This last metric is equivalent to (1 - ROCA%), where ROCA% is the area under the ROC curve. The dataset against which the filters are tested contains 16956 ham (non-spam) pages and 645 spam pages. Waterloo spam [5] scores are used as a baseline in the provided evaluation.

As mentioned earlier, our model takes two parameters θ_{Trust} and $\theta_{\text{AntiTrust}}$ as thresholds to form the seed sets. We submitted two runs, IVORY.70.30 and IVORY.90.10, corresponding to the $(70, 30)$ and $(90, 10)$ threshold settings, respectively. Table 5 summarizes the evaluation results for the baseline and the two variants of our spam filter. Unfortunately, neither of our runs beat the baseline (unaltered Waterloo spam scores).

A simple comparison between these two filters shows that IVORY.90.10 has a higher rate of misclassification but a lower (1-ROCA%). Looking at the ROC curves offers an explanation for the behavior of these filters. As depicted in Figure 3, IVORY.90.10 outperforms IVORY.70.30 when the false positive rate is low. This behavior lies in the choice of thresholds for our filter, $(90, 10)$ versus $(70, 30)$. A threshold of 70 for θ_{Trust} is relatively more likely to include more spam pages in the "good" seed set (set of non-spam pages used in the TrustRank algorithm). A threshold of 70 assigns high trust scores to some pages that are in fact spam. This explains why IVORY.90.10 is more effective when the false positive rate is low. However, a threshold of 90 is more likely to cover a smaller portion of the web graph when compared to a threshold of 70, thus propagating trust to fewer pages in the web graph. This might explain why IVORY.90.10 has a higher rate of misclassification than IVORY.70.30.

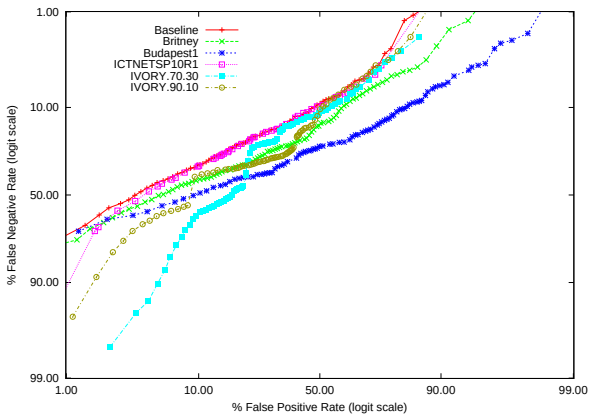

Figure 3: ROC curves for the spam filtering systems.

5. CONCLUSIONS

Participation in both last and this year's TREC web tracks has taught us valuable lessons in working with web-scale collections. In addition to sharing experimental results, Ivory is publicly-available as an open-source software package.[2] Code necessary to replicate most of the experiments here are included. We hope that this makes it easier for others to build on our results.

6. ACKNOWLEDGMENTS

This work was supported in part by the NSF under award IIS-0836560 and IIS-0916043; Google and IBM, via the Academic Cloud Computing Initiative (ACCI). Any opinions, findings, conclusions, or recommendations expressed in this paper are the authors' and do not necessarily reflect those of the sponsors. The last author is grateful to Esther and Kiri for their loving support.

7. REFERENCES

[1] M. Bendersky, D. Metzler, and W. B. Croft. Learning concept importance using a weighted dependence model. In *Third ACM International Conference on Web Search and Data Mining*, pages 31–40, New York, USA, 2010.

[2] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 193–200, 2007.

[3] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th International Conference on Information and Knowledge Management (CIKM 2009)*, pages 621–630, Hong Kong, China, 2009.

[4] G. V. Cormack and T. Lynam. TREC 2005 spam track overview. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005)*, Gaithersburg, Maryland, 2005.

[5] G. V. Cormack, M. D. Smucker, and C. L. A. Clarke. Efficient and effective spam filtering and re-ranking for large Web datasets. *CoRR*, abs/1004.5168, 2010.

[2] http://ivory.cc/

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150, San Francisco, California, 2004.

[7] T. Elsayed, F. Ture, and J. Lin. Brute-force approaches to batch retrieval: Scalable indexing with MapReduce, or why bother? Technical Report HCIL-2010-23, University of Maryland, College Park, Maryland, October 2010.

[8] D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: Using statistical analysis to locate spam Web pages. *Proceedings of the 7th International Workshop on the Web and Databases (WebDB), Paris, France*, June 2004.

[9] J. Gao, J.-Y. Nie, G. Wu, and G. Cao. Dependence language model for information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2004)*, pages 170–177, Sheffield, United Kingdom, 2004.

[10] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Data Base (VLDB 2004)*, pages 576–587, Toronto, Canada, 2004.

[11] Q. Jiang, L. Zhang, Y. Zhu, and Y. Zhang. Larger is better: Seed selection in link-based anti-spamming algorithms. In *Proceeding of the 17th International Conference on World Wide Web*, pages 1065–1066, Beijing, China, 2008.

[12] V. Krishnan and R. Raj. Web spam detection with anti-trust rank. *Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web*, pages 37–40, 2006.

[13] J. Lin, D. Metzler, T. Elsayed, and L. Wang. Of Ivory and Smurfs: Loxodontan MapReduce experiments for web search. In *Text Retrieval Conference (TREC 2009)*, Gaithersburg, Maryland, 2009.

[14] J. Lin and M. Schatz. Design patterns for efficient graph algorithms in MapReduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs Workshop (MLG-2010)*, pages 78–85, Washington, D.C., 2010.

[15] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3), 2009.

[16] D. Metzler. Automatic feature selection in the Markov Random Field model for information retrieval. In *Proceedings of the 16th International Conference on Information and Knowledge Management (CIKM 2007)*, pages 253–262, Lisbon, Portugal, 2007.

[17] D. Metzler and W. B. Croft. A Markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2005)*, pages 472–479, Salvador, Brazil, 2005.

[18] D. Metzler and W. B. Croft. Latent concept expansion using Markov random field model. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2007)*, pages 311–318, Amsterdam, the Netherlands, 2007.

[19] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 2006.

[20] A. Ntoulas, M. Najor, M. Manasse, and D. Fetterly. Detecting spam Web pages through content analysis. In *Proceedings of the 15th International Conference on World Wide Web*, pages 83–92, Edinburgh, Scotland, 2006.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Stanford Digital Library Working Paper SIDL-WP-1999-0120, Stanford University, 1999.

[22] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishing, San Francisco, 1999.

[23] B. Wu and K. Chellapilla. Extracting link spam using biased random walks from spam seed sets. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web*, Banff, Alberta, Canada, 2007.

[24] B. Wu, V. Goel, and B. D. Davison. Propagating trust and distrust to demote web spam. In *Proceedings of the WWW 2006 Workshop on Models of Trust for the Web*, Edinburgh, Scotland, 2006.

[25] B. Wu, V. Goel, and B. D. Davison. Topical TrustRank: Using topicality to combat web spam. In *Proceedings of the 15th International Conference on World Wide Web*, pages 63–72, Edinburgh, Scotland, 2006.

[26] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Úlfar. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th Symposium on Operating System Design and Implementation (OSDI 2008)*, pages 1–14, San Diego, California, 2008.

[27] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, , and G. Sun. A general boosting method and its application to learning ranking functions for web search. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, 2008.

[28] B. Zhou, J. Pei, and Z. Tang. A spamicity approach to web spam detection. In *Proceedings of the 2008 SIAM. International Conference on Data Mining (SDM'08)*, pages 277–288, Atlanta, Georgia, 2008.