Learning to Rank on a Cluster using Boosted Decision Trees (Extended Abstract)

Krysta M. Svore
Microsoft Research
1 Microsoft Way
Redmond, WA 98052
ksvore@microsoft.com

Christopher J. C. Burges
Microsoft Research
1 Microsoft Way
Redmond, WA 98052
cburges@microsoft.com

Abstract

We investigate the problem of learning to rank on a cluster using Web search data composed of 140,000 queries and approximately fourteen million URLs, and a boosted tree ranking algorithm called LambdaMART. We compare to a baseline algorithm that has been carefully engineered to allow training on the full dataset using a single machine, in order to evaluate the loss or gain incurred by the distributed algorithms we consider. Our contributions are two-fold: (1) we implement a method for improving the speed of training when the training data fits in main memory on a single machine; (2) we develop a training method for the case where the training data size exceeds the main memory of a single machine that easily scales to far larger datasets, i.e., billions of examples, and is based on data distribution. Results of our methods on a real-world Web dataset indicate significant improvements in training speed.

1 Introduction

With the growth of the Web, large datasets are becoming increasingly common — a typical commercial search engine may gather several terabytes per day of queries and Web search interaction information. This opens a wide range of new opportunities, both because the best algorithm for a given problem may change dramatically as more data becomes available [1], and because such a wealth of data promises solutions to problems that could not be previously approached. We investigate two synchronous approaches for learning to rank on a distributed computer which target different computational scenarios. In both cases, the base algorithm we use is LambdaMART [13, 2], which is a linear combination of regression trees, and as such lends itself to parallelization in various ways. Our approaches are detailed in a forthcoming book on distributed learning [12]. Our first method applies when the full training dataset fits in main memory on a single machine. In this case, our approach distributes the tree split computations, but not the data. Note that while this approach gives a speedup due to parallelizing the computation, it is limited in the amount of data that can be used since all of the training data must be stored in main memory on every node. This limitation is removed in our second approach, which applies when the full training dataset is too large to fit in main memory on a single machine. In this case, our approach distributes the training data samples and corresponding training computations and is scalable to very large amounts of training data. We develop two methods of choosing the next regression tree in the ensemble for our second approach, and compare and contrast the resulting evaluation accuracy and training speed. In order to accurately investigate the benefits and challenges of our techniques, we compare to a standalone, centralized version that can train on the full training dataset on a single node. To this end, the standalone version has been carefully engineered (for example, memory usage is aggressively trimmed by using different numbers of bits to encode different features).

2 Approaches to Distributing LambdaMART

We focus on the task of Web search ranking by learning boosted tree ensembles produced using LambdaMART (see Appendix A and [13, 2] for details). LambdaMART was one of the primary components of the winning ranking system in the recent Yahoo! Learning to Rank Challenge for Web search [14]. The final model f is an ensemble defined as the sum $f(\mathbf{x}, N) = \sum_{n=1}^{N} h_n(\mathbf{x})$, where each h_n is a weak hypothesis. Moreover, f is constructed incrementally as weak hypotheses are added one by one.

Our first approach, called *feature-distributed* LambdaMART, is a synchronous algorithm similar to the approach in [11], where the vertex split computations are distributed, rather than the data samples, except that our method has a communication cost that is constant in the number of training samples as opposed to linear. Our approach targets the scenario where each node can store the full training dataset in main memory; the goal is to train on a cluster more quickly than on a single machine and produce a solution which is equivalent to the solution resulting from training on all of the data on a single machine (called the *centralized* model). The algorithm proceeds as follows (see Appendix B for pseudocode). Let there be K workers and no master, and let each worker store the full training set S, consisting of M instance-label pairs, in memory. Let the set of features A be partitioned into K subsets, $A_1, \ldots, A_K$, such that each subset is assigned to one of the K workers. Every worker maintains a copy of the ensemble $f(\mathbf{x}, n)$ and updates it after the new regression tree $\{R_{\ell n}\}_{\ell=1}^{L}$ is constructed during each boosting iteration n. Each vertex in the tree is described by an optimal feature, corresponding split threshold, and change in loss, collectively denoted by φ. Each worker k computes φ_k based on its set of features A_k and sends φ_k to all other workers. Every worker, after it has received all of the φ_k's, determines the φ_k with the smallest loss, denoted by φ_*, creates the two new children for the model, and then computes which samples go left and which go right. Note that φ_* is the same for all workers, resulting in equivalent ensembles $f(\mathbf{x}, n)$ across all workers. Each worker must wait until it receives all $\varphi_k,\ k = 1, \ldots, K$, before determining φ_*. Some workers will be idle while others are still computing their φ_k's.

Our second approach, called *data-distributed* LambdaMART, distributes the training data across the nodes and does not produce a model equivalent to the centralized model. It differs from previous methods [4, 5, 7, 9, 10] in that we use minimal communication cost that is constant in, as opposed to scaling with, the number of training samples. Since it distributes by data sample, the amount of training data can scale with cluster size, which is usually more desirable than scaling with the number of features since the number of training samples tends to far exceed the number of features. Within this approach, we consider two weak hypothesis selection methods: (1) *full* — the master picks the weak hypothesis that maximizes the evaluation score, and (2) *sample* — the master picks a weak hypothesis at random. Let there be a master and K workers. The training set S is partitioned into K subsets, $S_1, \ldots, S_K$, each residing on one of the K workers. Assume that the master has an ensemble $f(\mathbf{x}, N-1)$ composed of $N-1$ weak hypotheses, and the task is to choose the next weak hypothesis. In full selection, each worker k has a copy of $f(\mathbf{x}, N-1)$ and trains a candidate weak hypothesis on its dataset S_k to generate the next weak hypothesis $h_{N,k}(\mathbf{x})$ and sends it to all other workers. Each worker k now evaluates the set of candidates constructed by the other workers, $\{h_{N,k}(\mathbf{x})\}_{[K]\setminus\{k\}}$, where $f_k(\mathbf{x}, N) = f(\mathbf{x}, N-1) + h_{N,k}(\mathbf{x})$, and calculates the set of values $\{C_k(f_k(\mathbf{x}, N))\}_{[K]\setminus\{k\}}$ and returns them to the master, where C is the evaluation measure. The master then chooses the candidate with the largest evaluation score C on the entire training set S. This cross-validation adds further regularization to the training. The master sends the index k of the selected weak hypothesis to all workers. Each worker then updates the model: $f(\mathbf{x}, N) = f(\mathbf{x}, N-1) + h_{N,k}(\mathbf{x})$. On the next iteration, all of the workers attempt to add another weak learner to $f(\mathbf{x}, N)$. The communication cost depends on the size of the weak hypothesis and the number of workers, but not the size of the training data. In addition, communication only occurs *once* per boosting iteration, removing the need to communicate once per vertex split computation. To further reduce the communication cost, we take advantage of the power of sampling in the *sample* selection method. Each worker k has the same ensemble $f(\mathbf{x}, N-1)$ and uses its dataset to construct $f_k(\mathbf{x}, N)$. The master chooses a random worker k's hypothesis $h_{N,k}(\mathbf{x})$ as the next hypothesis (replacing Steps 15–17 in Algorithm 3 in Appendix B with random selection of a hypothesis). The random selection requires only communicating the chosen candidate weak hypothesis from the master to the workers and eliminates the expensive evaluation step in the full method.

3 Experiments

We evaluate our proposed methods on a real-world Web dataset that contains $140,000$ queries sampled from a commercial search engine and corresponding URLs (14,533,212 query-URL pairs), each with a vector of several thousand feature values and a human-generated relevance label $l \in \{0, 1, 2, 3, 4\}$, with 0 meaning document d is not relevant to query q and 4 meaning d is highly relevant to q. We divide the dataset into train/valid/test sets by selecting a random $80\%/10\%/10\%$ queries and corresponding URLs, respectively. We ran all of our experiments on a 40-node MPI cluster (see Appendix C for details). We swept a range of parameter values for each experiment and determine the best based on the evaluation accuracy on a validation set: we varied the learning rate η from 0.05 to 0.5 and the number of leaves L from 20 to 220, and trained for $N = 1000$ boosting iterations. We evaluate using NDCG [8] and perform a t-test with a significance level of 0.05.

We first examine speed improvements and communication requirements. Figure 1(a) shows the difference in total training time between centralized and feature-distributed LambdaMART for the number of workers $K = \{1, \ldots, 32\}$ and 4000 features. For feature-distributed LambdaMART, $\frac{|A|}{K}$ features are assigned to each node. The parameters are set to $\eta = 0.1$, $N = 500$, and $L = 200$. The total train time is evaluated on two types of clusters: Type I, as previously described, and Type II, which has 32.0 GB RAM and two quad-core Intel Xeon 5430 processors running at 2.67 GHz. As

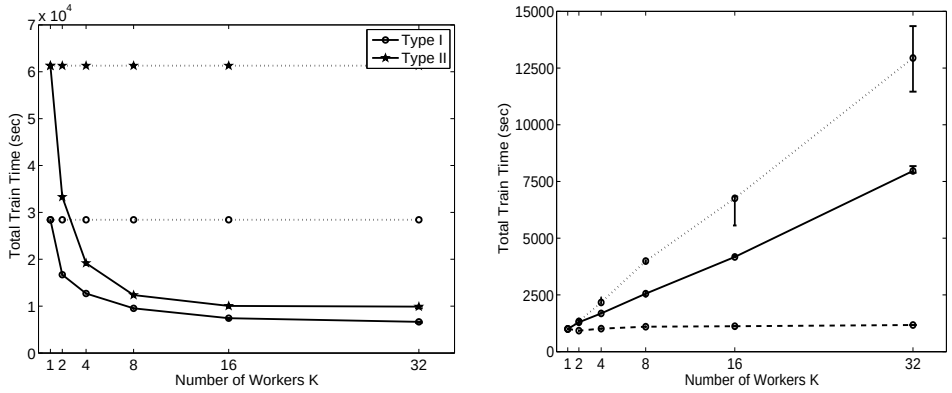
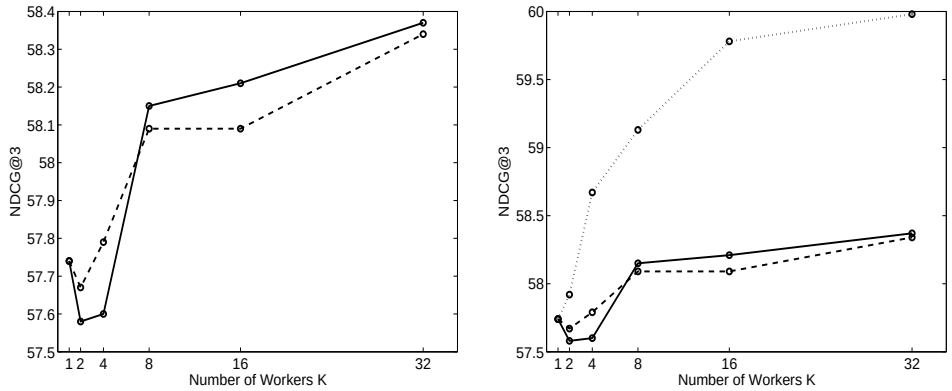


Figure 1: Number of Workers K versus Total Training Time in seconds for (a) centralized (dotted) and feature-distributed (solid) LambdaMART, for 4000 features and two cluster types. Centralized was trained on the full dataset for all K. (b) Total data used = $3500K$ queries, for centralized (dotted), full data-distributed (solid), and sample data-distributed (dashed) LambdaMART with $L = 20$ leaves. Each experimental setting was run three times; times are shown by the bars around each point. Invisible bars indicate times are roughly equivalent.

seen in Fig. 1(a), feature-distributed LambdaMART on 32 workers achieves a factor of 6 speed-up over centralized LambdaMART when trained on Type II and a factor of 3 speed-up on Type I.

We next consider the case where the training data S cannot fit in the main memory of a single machine. We simulate memory constraints by assuming one worker can store at most 3500 queries; in order to exploit more training data, the data must reside on separate workers. As the number of workers increases, it simulates the case where more and more training data is available, but the memory capacity of a single worker remains the same. Figure 1(b) shows the number of workers versus the total training time in seconds, for weak hypotheses with varying numbers of leaves, for centralized, and full and sample data-distributed LambdaMART. The parameter settings used are $L = 20$, $N = 1000$, and $\eta = 0.1$. The x-axis indicates the number of workers K, where each worker trains on $\frac{|S|}{32} \approx 3500$ queries; with respect to centralized LambdaMART, the x-axis indicates the number of training queries residing on the single worker, $\frac{|S|}{32}K$. The point at $K = 1$ represents training centralized LambdaMART on $\frac{|S|}{32} \approx 3500$ queries. As K increases, the total train time increases since the communication costs grow with the number of workers K. Since the evaluation and communication costs are almost negligible in sample data-distributed LambdaMART, the total



a) NDCG@3. b) NDCG@3 with Centralized.

Figure 2: Number of Workers K versus NDCG@3 for full (solid) and sample (dashed) data-distributed LambdaMART. Each worker trains on 3500 queries. Figure (b) includes centralized LambdaMART (dotted) trained on $3500K$ queries at each x-axis point. Significant differences are stated in the text.

train time is roughly equivalent to training on a single node, even though the amount of training data across the cluster increases with K.

We next evaluate the corresponding prediction accuracy of our data-distributed algorithm using the full and sample selection strategies[1]. Figure 2(a) plots the number of workers K versus NDCG for full and sample data-distributed LambdaMART. The training data distributed among the workers in the cluster acts as additional validation data since it is used for the evaluation and selection of the weak hypothesis. Full and sample selection strategies result, for each K, in similar NDCG scores, and exhibit NDCG accuracy increases as K increases. Having $3500K$ queries in the cluster, for $K = \{8, 16, 32\}$, yields significant gains in NDCG@3 over training on 3500 queries ($K = 1$). Thus, additional data, although mostly used for validation, significantly increases NDCG accuracy. In Figure 2(b), we analyze the effect of lifting the memory constraint and plot the centralized algorithm accuracy trained on $3500K$ queries (dotted line) on a single worker, for increasing values of K. For $K = \{4, 8, 16, 32\}$, the resulting model is significantly better than the corresponding data-distributed models trained on $3500K$ queries, indicating that when able to use additional data directly for training, it is preferable to using it for cross-validation. Even though the central model is superior in accuracy to our data-distributed models (assuming memory of a single worker is not constrained, further discussed in Appendix E), our data-distributed algorithms exhibit significant gains when the memory of a single worker is exhausted.

4 Conclusions and Future Work

In summary, we have presented two approaches for distributing LambdaMART. The feature-distributed approach requires that the full training set fit in main memory on each node in the cluster and achieves up to a 6-fold significant speed-up over centralized LambdaMART with the same accuracy. Our data-distributed approach, which can scale to billions of training samples, employs either the full or sample weak hypothesis selection strategy, and we have shown that both selection strategies offer significant speed-ups over centralized LambdaMART. Sample data-distributed LambdaMART demonstrates no significant accuracy loss compared to full data-distributed LambdaMART, and achieves even more significant training time speed-ups. Our data-distributed algorithms, however, indicate that using data for massive cross-validation results in significant accuracy loss. In the future, it is worth determining a distributed method that can scale to billions of examples, but with accuracy that is equivalent or superior to training on centralized data; we have developed a first step toward achieving this goal by presenting a method where the communication is independent of the number of samples.

[1]Recall that the feature-distributed algorithm outputs the same model as the centralized algorithm and thus has the same prediction accuracy.

5 Acknowledgements

We thank Ofer Dekel for his insightful ideas, his invaluable contributions to code and cluster development, and his assistance in running experiments.

References

[1] M. Banko and E. Brill. Scaling to very very large corpora for natural language disambiguation. In *Association for Computational Linguistics (ACL)*, pages 26–33, 2001.

[2] C.J.C. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, Microsoft Research, 2010.

[3] C.J.C. Burges, R. Ragno, and Q.V. Le. Learning to rank with non-smooth cost functions. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[4] P. Domingos and G. Hulten. Mining high-speed data streams. In *SIGKDD Conference on Knowledge and Data Mining (KDD)*, pages 71–80, 2000.

[5] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *International Conference on Machine Learning (ICML)*, 2001.

[6] P. Donmez, K.M. Svore, and C.J.C. Burges. On the local optimality of lambdarank. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2009.

[7] W. Fan, S. Stolfo, and J. Zhang. The application of AdaBoost for distributed, scalable and online learning. In *SIGKDD Conference on Knowledge and Data Mining (KDD)*, pages 362–366, 1999.

[8] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 41–48, 2000.

[9] A. Lazarevic. The distributed boosting algorithm. In *SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 311–316, 2001.

[10] A. Lazarevic and Z. Obradovic. Boosting algorithms for parallel and distributed learning. *Distributed and Parallel Databases*, 11:203–229, 2002.

[11] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. PLANET: Massively parallel learning of tree ensembles with MapReduce. In *International Conference on Very Large Databases (VLDB)*, 2009.

[12] K.M. Svore and C.J.C. Burges. Distributed learning to rank using boosted decision trees. In R. Bekkerman, M. Bilenko, and J. Langford, editors, *Large-scale Machine Learning*. 2010.

[13] Q. Wu, C.J.C. Burges, K.M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Journal of Information Retrieval*, 2009.

[14] Yahoo! Learning to Rank Challenge. *http://learningtorankchallenge.yahoo.com/*, 2010.

A LambdaMART

LambdaRank is a general method for learning to rank given an arbitrary cost function, and it circumvents the problem that most information retrieval measures have ill-posed gradients. It has been shown empirically that LambdaRank can optimize general IR measures [6]. A key idea in LambdaRank is to define the derivatives (of the cost with respect to the model scores) *after* the documents have been sorted by the current model scores, which circumvents the problem of defining a derivative of a measure whose value depends on the sorted order of a set of documents. These derivatives are called λ*-gradients*. A second key observation in LambdaRank is to note that many training algorithms (for example, neural network training and MART) do not need to know the cost directly; they only need the derivatives of the cost with respect to the model scores.

For example, the λ-gradient for NDCG [8] for a pair of documents D_i and D_j, where D_i is more relevant to query q than D_j, can be defined as the product of the derivative of a convex cost C_{ij} and

the NDCG gained by swapping the two documents:

$$\lambda_{ij} \equiv \left| \Delta \text{NDCG} \frac{\delta C_{ij}}{\delta o_{ij}} \right| \tag{1}$$

where o_{ij} is the difference in the model scores of the two documents. The λ-gradient for a single document is computed by marginalizing over the pairwise λ-gradients: $\lambda_i = \sum_{j \in P} \lambda_{ij}$, where the sum is over all pairs P for query q which contain document i.

MART is a class of boosting algorithms that may be viewed as performing gradient descent in function space, using regression trees. The final model maps an input feature vector $\mathbf{x} \in \mathbb{R}^d$ to a score $f(\mathbf{x}) \in \mathbb{R}$. MART is a class of algorithms, rather than a single algorithm, because it can be trained to minimize general costs (to solve, for example, classification, regression or ranking problems). The final score f can be written as

$$f(\mathbf{x}, N) = \sum_{n=1}^{N} \alpha_n f_n(\mathbf{x}) \ ,$$

where each $f_n(\mathbf{x}) \in \mathbb{R}$ is a function modeled by a single regression tree and the $\alpha_n \in \mathbb{R}$ are weights. Both the f_n and the α_n are learned during training. We refer to $\alpha_n f_n$ as the weak hypothesis h_n. A given f_n maps a given $\mathbf{x}$ to a real value by passing $\mathbf{x}$ down the tree, where the path (left or right) at a given node in the tree is determined by the value of a particular feature $x_j, j = 1, \ldots, d$ and where the output of the tree is taken to be a fixed value associated with each leaf, $v_{\ell n},\ \ell = 1, \ldots, L,\ n = 1, \ldots, N$, where L is the number of leaves and N is the number of trees. For a given task (in our case, ranking), given training and validation sets, the user-chosen parameters of the training algorithm are the number of trees N, a fixed learning rate η (that multiplies every $v_{\ell n}$ for every tree), and the number of leaves[2] L. The binary decision functions at each node of each tree and the $v_{\ell n}$ are learned during training; the decision functions are chosen to minimize a least-squares loss.

Clearly, since MART models derivatives, and LambdaRank works by specifying the derivatives at any point during training, the two algorithms are well suited to each other. LambdaMART is the marriage of the two, and we refer the reader to [3, 2] for details. The set of M scores (one for each training sample) is computed, and the λ-gradient $\lambda_m,\ m = 1, ..., M$, of the cost function with respect to each model score is computed. Thus a single number is associated to each training sample, namely, the gradient of the cost with respect to the score which the model assigns to that sample. Tree f_n is then just a least-squares regression tree that models this set of gradients (so each leaf models a single value of the gradient). The overall cost is then reduced by taking a step along the gradient. This is often done by computing a Newton step $v_{\ell n}$ for each leaf, where the $v_{\ell n}$ can be computed exactly for some costs. Every leaf value is then multiplied by a learning rate η. Taking a step that is smaller than the optimal step size (i.e., the step size that is estimated to maximally reduce the cost) acts as a form of regularization for the model that can significantly improve test accuracy. The LambdaMART algorithm is outlined in Algorithm 1, where we have added the notion that the first model trained can be any previously trained model (Step 3), which is useful for model adaptation tasks.

B Distributed LambdaMART

In this section, we give pseudocode for both feature-distributed (Algorithm 2) and data-distributed (Algorithm 3) LambdaMART.

C Cluster and Parameter Sweep Details

In this section, we give details about our cluster and the sweeps conducted for the experiments. We ran all of our experiments on a 40-node MPI cluster, running Microsoft HPC Server 2008 (for details, see Appendix C). Each node has two 4-core Intel Xeon 5550 processors running at 2.67GHz and 48 GB of RAM. Each node is connected to two 1Gb Ethernet networks: a private network dedicated to MPI traffic and a public network. Each network is provided by a Cisco 3750e Ethernet switch. The communication layer between nodes on our cluster was written using MPI.NET.

[2]One can also allow the number of leaves to vary at each iteration, but we do not consider such models here.

Algorithm 1 LambdaMART.

Input: Training Data: $\{\mathbf{x}_m, y_m\}$, $m = 1, \ldots, M$;
Number of Trees: N;
Number of Leaves: L;
Learning Rate: η;
Output: Model: $f(\mathbf{x}, N)$;
$f(\mathbf{x}, 0) = BaseModel(\mathbf{x})$ {$BaseModel$ may be empty.}
for $n = 1$ to N **do**
  for $m = 1$ to M **do**
    $\lambda_m = G(q, \mathbf{x}, y, m)$ {Calculate λ-gradient for sample m as a function of the query q and the documents and labels $\mathbf{x}$, y associated with q.}
    $w_m = \frac{\partial \lambda_m}{\partial f(\mathbf{x}_m)}$ {Calculate derivative of λ-gradient for sample m.}
  end for
  $\{R_{\ell n}\}_{\ell=1}^{L}$ {Create L-leaf regression tree on $\{\mathbf{x}_m, \lambda_m\}_{m=1}^{M}$.}
  for $\ell = 1$ to L **do**
    $v_{\ell n} = \frac{\sum_{\mathbf{x}_m \in R_{\ell n}} \lambda_m}{\sum_{x_m \in R_{\ell n}} w_m}$ {Find the leaf values based on approximate Newton step.}
  end for
  $f(\mathbf{x}_m, n) = f(\mathbf{x}_m, n-1) + \eta \sum_\ell v_{\ell n} 1(\mathbf{x}_m \in R_{\ell n})$ {Update model based on approximate Newton step and learning rate.}
end for

Algorithm 2 Feature-distributed LambdaMART.

Input: Training Data: $\{x_m, y_m\}$, $m = 1, \cdots, M$;
Number of Trees: N;
Number of Leaves: L;
Learning Rate: η;
Number of Workers: K;
Output: Model: $f(\mathbf{x}, N)$;
for $k = 1$ to K **do**
  $f(\mathbf{x}, 0) = BaseModel(\mathbf{x})$ {$BaseModel$ may be empty.}
  for $n = 1$ to N **do**
    for $m = 1$ to M **do**
      $\lambda_m = G(q, \mathbf{x}, y, m)$ {Calculate λ-gradient for sample m as a function of the query q and the documents and labels $\mathbf{x}$, y associated with q.}
      $w_m = \frac{\partial \lambda_m}{\partial f(\mathbf{x}_m)}$ {Calculate derivative of λ-gradient for sample m.}
    end for
    for $\ell = 1$ to $L - 1$ **do**
      φ_k {Compute the optimal feature and split, φ_k, over features A_k on worker k.}
      $Broadcast(\varphi_k)$ {Broadcast φ_k to all other workers.}
      $\varphi_* = \{\arg\max_k(\varphi_k)\}_{k=1}^{K}$ {Find optimal φ_* across all φ_k's.}
      $R_{\ell n}$ {Create regression tree on φ_* and $\{\mathbf{x}_m, \lambda_m\}_{m=1}^{M}$.}
    end for
    for $\ell = 1$ to L **do**
      $v_{\ell n} = \frac{\sum_{x_m \in R_{\ell n}} \lambda_m}{\sum_{x_m \in R_{\ell n}} w_\ell}$ {Find the leaf values based on approximate Newton step.}
    end for
    $f(\mathbf{x}_m, n) = f(\mathbf{x}_m, n-1) + \eta \sum_\ell v_{\ell n} 1(\mathbf{x}_m \in R_{\ell n})$ {Update model based on approximate Newton step and learning rate.}
  end for
end for

Algorithm 3 Data-distributed LambdaMART.

Input: Training Data: $\{x_m, y_m\}$, $m = 1, \ldots, M$;
Number of Trees: N;
Number of Leaves: L;
Learning Rate: η;
Number of Workers: K;
Output: Model: $f(\mathbf{x}, N)$;
for $k = 1$ to K **do**
  $f(\mathbf{x}, 0) = BaseModel(\mathbf{x})$ {$BaseModel$ may be empty.}
  for $n = 1$ to N **do**
    for all $m \in S_k$ **do**
      $\lambda_m = G(q, \mathbf{x}, y, m)$ {Calculate λ-gradient for sample m as a function of the query q and the documents and labels $\mathbf{x}$, y associated with q, where m is in the fraction of training data S_k on worker k.}
      $w_m = \frac{\partial \lambda_m}{\partial f(\mathbf{x}_m)}$ {Calculate derivative of λ-gradient for sample m.}
    end for
    $\{R_{\ell n}\}_{\ell=1}^{L}$ {Create L-leaf regression tree $\{R_{\ell nk}\}_{\ell=1}^{L}$ on $\{\mathbf{x}_m, \lambda_m\},\ m \in S_k$.}
    for $\ell = 1$ to L **do**
      $v_{\ell n} = \frac{\sum_{\mathbf{x}_m \in R_{\ell n}} \lambda_m}{\sum_{\mathbf{x}_m \in R_{\ell n}} w_m}$ {Find the leaf values based on approximate Newton step.}
    end for
    $f_k(\mathbf{x}_m, n) = f(\mathbf{x}_m, n-1) + \eta \sum_\ell v_{\ell n} 1(\mathbf{x}_m \in R_{\ell nk})$ {Update model based on approximate Newton step and learning rate.}
    $\{C_k(f_k(\mathbf{x}, n))\}_{[K]\setminus\{k\}}$ {Compute candidate weak hypotheses cost values.}
    $C(f_k(\mathbf{x}, n)) = \sum_{i \in V} C_i(f_k(\mathbf{x}, n))$ {Evaluate candidate weak hypotheses from all other workers.}
    $f(\mathbf{x}, n) = \underset{f_k(\mathbf{x}, n)}{\operatorname{argmax}}\, C(f_k(\mathbf{x}, n))$ {Choose best weak hypothesis and update model.}
  end for
end for

Table 1 gives the parameters that gave the best accuracy on the validation set for each experiment.

Table 1: The learning rate η and the number of leaves L for centralized LambdaMART, and full and sample data-distributed LambdaMART, respectively. The first set of columns are the parameters when training on 3500 queries per worker; in the central case, a single worker trains on $3500K$ queries. The second set of columns are the parameters when training on 7000 overlapping queries per worker; in the central case, a single worker trains on $7000K$ queries. The final columns contain the parameters when training on $\frac{|S|}{K}$ queries per worker; in the central case, a single worker trains on $\frac{|S|}{K}$ queries.

	3500		7000		All	
K	η	L	η	L	η	L
1	0.1, 0.1, 0.1	20, 20, 20	0.1, 0.1, 0.1	80, 80, 80	0.1, 0.1, 0.1	20, 20, 20
2	0.1, 0.05, 0.05	80, 80, 80	0.1, 0.1, 0.1	180, 180, 180	0.1, 0.1, 0.1	80, 190, 200
4	0.1, 0.1, 0.05	180, 80, 80	0.1, 0.05, 0.05	200, 200, 200	0.1, 0.05, 0.05	180, 170, 200
8	0.1, 0.05, 0.05	200, 120, 120	0.1, 0.05, 0.05	200, 200, 200	0.05, 0.05, 0.05	200, 180, 200
16	0.1, 0.05, 0.05	200, 140, 140	0.1, 0.05, 0.05	200, 200, 200	0.1, 0.05, 0.05	200, 170, 160
32	0.1, 0.05, 0.05	200, 140, 140	0.1, 0.05, 0.05	200, 140, 140	0.1, 0.05, 0.05	200, 100, 140

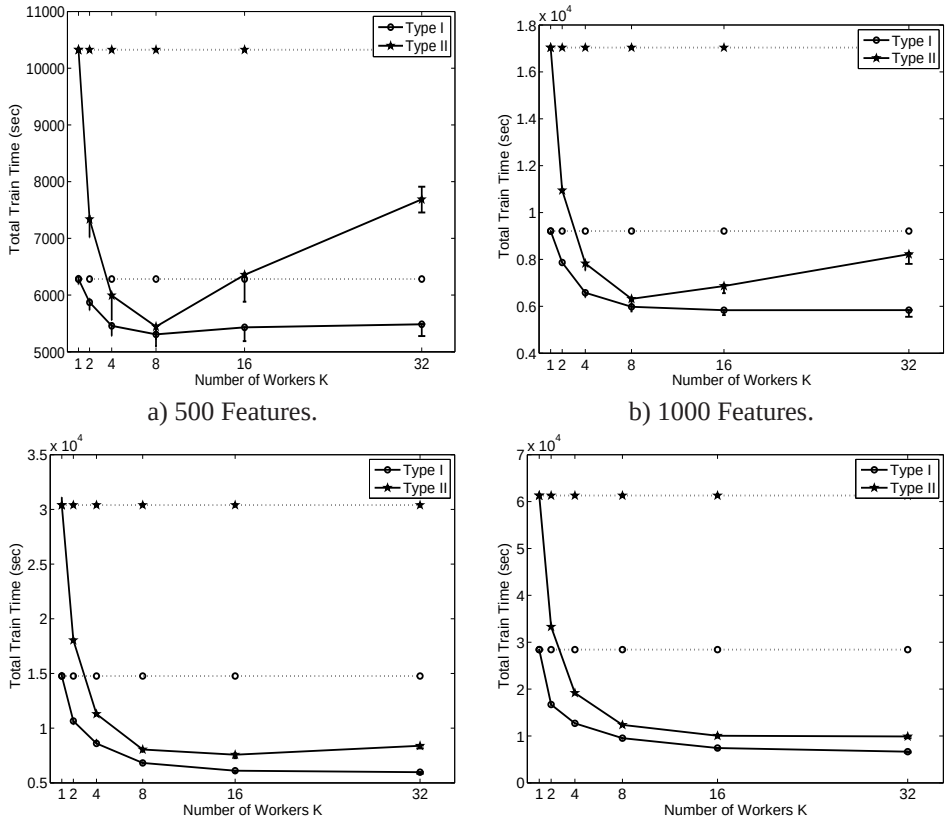
D Additional Experimental Results

D.1 Training Time Results

Figures 3(a)–(d) show the difference in total training time between centralized and feature-distributed LambdaMART. We vary the number of workers K from 1–32, and the number of features $|A|$ from 500–4000. For feature-distributed LambdaMART, $\frac{|A|}{K}$ features are assigned to each node. We employ the same set of parameters for each algorithm to provide fair training time comparisons; the parameters are set to $\eta = 0.1$, $N = 500$, and $L = 200$.

As shown in Figure 3, feature-distributed LambdaMART (solid lines) achieves significantly faster training times than centralized LambdaMART (dotted lines) on both clusters. When trained on Type II with 500 features, feature-distributed LambdaMART with 8 workers achieves almost a two-fold speed-up over centralized LambdaMART (Fig. 3(a)). When the number of features is small, as the number of workers increases, the cost of communication among the workers outweighs the speed-ups due to feature distribution, as seen by the increase in time when $K \geq 8$ for Type II (Fig. 3(a)–(b)). However, as the number of features increases, communication occupies a smaller percentage of the training time, resulting in decreasing training times. For example, feature-distributed LambdaMART on Type II with 4000 features (Fig. 3(d)) exhibits decreasing training times as the number of workers increases and achieves a factor of 6 speed-up over centralized LambdaMART when trained on 32 workers. When trained on Type I, feature-distributed LambdaMART exhibits decreasing training times as the number of workers grows; with 32 workers training on 4000 features, roughly a 3-fold speed-up is obtained.

We evaluate the time required to train on $|S|$ queries, where the queries are split among K workers. For the centralized algorithm, a single worker trains on $|S|$ queries. We set $\eta = 0.1$, $L = \{100, 200\}$, and $N = 1000$. Figure 4 plots the number of workers K versus the total train time in seconds; every point represents a model trained on all $|S|$ queries. For the data-distributed algorithms, the training data S is split among K workers: as K increases, the number of queries on a single worker ($\frac{|S|}{K}$) decreases, but the total number of queries across all nodes remains constant ($|S|$). The two points at $K = 1$ represent the training times of centralized LambdaMART trained on fourteen million URLs. The central model at $K = 1$ is plotted at all K values for reference (shown by the dotted lines). When $K > 1$, the train times of full and sample data-distributed LambdaMART are significantly less than the centralized algorithm. Particularly notable is the reduction in train time obtained by the sample data-distributed LambdaMART algorithm.



a) 500 Features. b) 1000 Features.

c) 2000 Features. d) 4000 Features.

Figure 3: Number of Workers K versus Total Training Time in seconds for centralized (dotted) and feature-distributed (solid) LambdaMART, for 500–4000 features and two cluster types. Centralized was trained on the full dataset for all K. Each experimental setting was run three times; times are shown by the bars around each point. Invisible bars indicate times are roughly equivalent.

D.2 Accuracy Comparison

The first experiment evaluates the change in accuracy of our data-distributed algorithms as the number of workers increases. We simulate memory constraints by assuming one worker can store at most 3500 queries — in order to exploit more training data, the data must reside on separate workers. As the number of workers increases, it simulates the case where more and more training data is available, but the memory capacity of a single worker remains the same. The training set S is randomly partitioned into 32 disjoint subsets and each subset resides on one of the 32 nodes in our cluster. Each partition contains roughly 3500 queries and corresponding URLs. When $K = 1$, a single worker trains on 3500 queries, when $K = 2$, two workers train on 3500 queries each, and so on.

Figure 5 plots the number of workers K versus NDCG for full and sample data-distributed LambdaMART The training data distributed among the workers in the cluster acts as additional validation data since it is used for the evaluation and selection of the weak hypothesis. Full and sample selection strategies result, for each K, in similar NDCG scores, and exhibit NDCG accuracy increases as K increases. Having $3500K$ queries in the cluster, for $K = \{8, 16, 32\}$, yields significant gains in NDCG@3 and 10 over training on 3500 queries ($K = 1$). Thus, additional data, although mostly used for validation, significantly increases NDCG accuracy.

In Figure 5(d), we analyze the effect of lifting the memory constraint and plot the centralized algorithm accuracy trained on $3500K$ queries (dotted line) on a single worker, for increasing values of K. For $K = \{4, 8, 16, 32\}$, the resulting model is significantly better than the corresponding

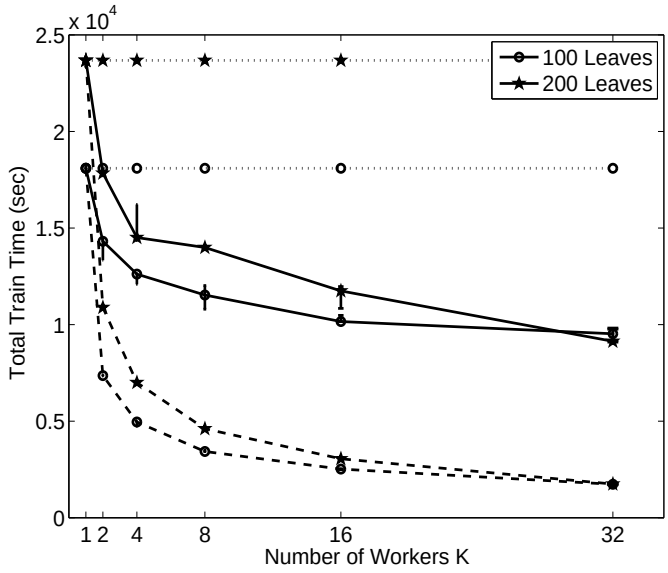


Figure 4: Number of Workers K versus Training Time in seconds for centralized (dotted), full data-distributed (solid), and sample data-distributed (dashed) LambdaMART on fourteen million samples (query-URL pairs). Each experimental setting was run three times; times are shown by the bars around each point.

data-distributed models trained on $3500K$ queries, indicating that when able to use additional data directly for training, it is preferable to using it for cross-validation.

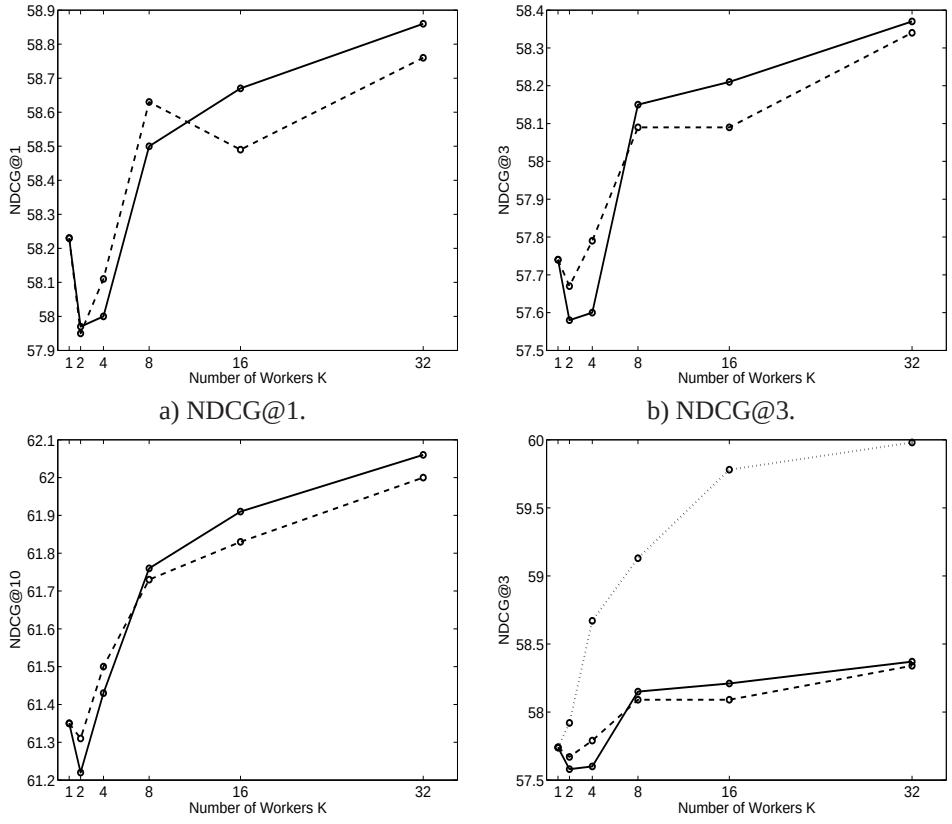
Somewhat surprisingly, as the amount of data increases, even though the data is highly distributed, the optimal values of the learning rate η and number of leaves L change dramatically for data-distributed LambdaMART (see Table 1). Even though it is only the amount of validation data that increases as K increases, the parameters behave similarly to increasing the amount of centralized training data.

For our next experiment, we investigate how training on overlapping sets of data affects NDCG accuracy. Assume that a single worker can store at most 7000 queries and let the amount of training data available be $3500K$ queries. We construct our overlapping sets as follows: the training data S is divided into K sets, $S_1, \ldots, S_K$. Worker k stores set S_k and set S_{k+1}, resulting in 7000 queries. For example, when $K = 4$, $S_1 + S_2, S_2 + S_3, S_3 + S_4, S_4 + S_1$ reside on workers $1, 2, 3, 4$, respectively. The total number of unique queries in the cluster remains $3500K$. This approach can easily scale to larger datasets.

Figure 6 plots the number of workers K versus NDCG, where each worker contains an overlapping set of 7000 queries, compared to 3500 queries. The accuracy gains from training on 7000 queries per worker instead of 3500 are significant for all K at NDCG@3 and 10, further indicating that training on more data is better than validating over more data, and also indicating that the samples need not be unique across the workers. In particular, training $K = 8$ workers on overlapping 7000-query sets results in similar accuracy to training $K = 32$ workers on 3500-query sets. In all cases, full and sample selection strategies result in similar accuracies.

In Figure 6(d), we again lift the memory constraint, and plot the NDCG@3 accuracy of the central model on $3500K$ queries (dotted line). The results highlight the benefit of increasing the amount of training data per worker over using additional validation data, as seen by the significant gap between the central and data-distributed models.

Even though the central model is superior in accuracy to our data-distributed models (assuming memory of a single worker is not constrained), our data-distributed algorithms exhibit significant gains when the memory of a single worker is exhausted. In this scenario, a benefit of our data-distributed algorithm is not only parallelized training, but also that the amount of information com-



a) NDCG@1. b) NDCG@3.

c) NDCG@10. d) NDCG@3 with Centralized.

Figure 5: Number of Workers K versus NDCG@1, 3, 10 for full (solid) and sample (dashed) data-distributed LambdaMART. Each worker trains on 3500 queries. Figure (d) includes centralized LambdaMART (dotted) trained on $3500K$ queries at each x-axis point. Significant differences are stated in the text.

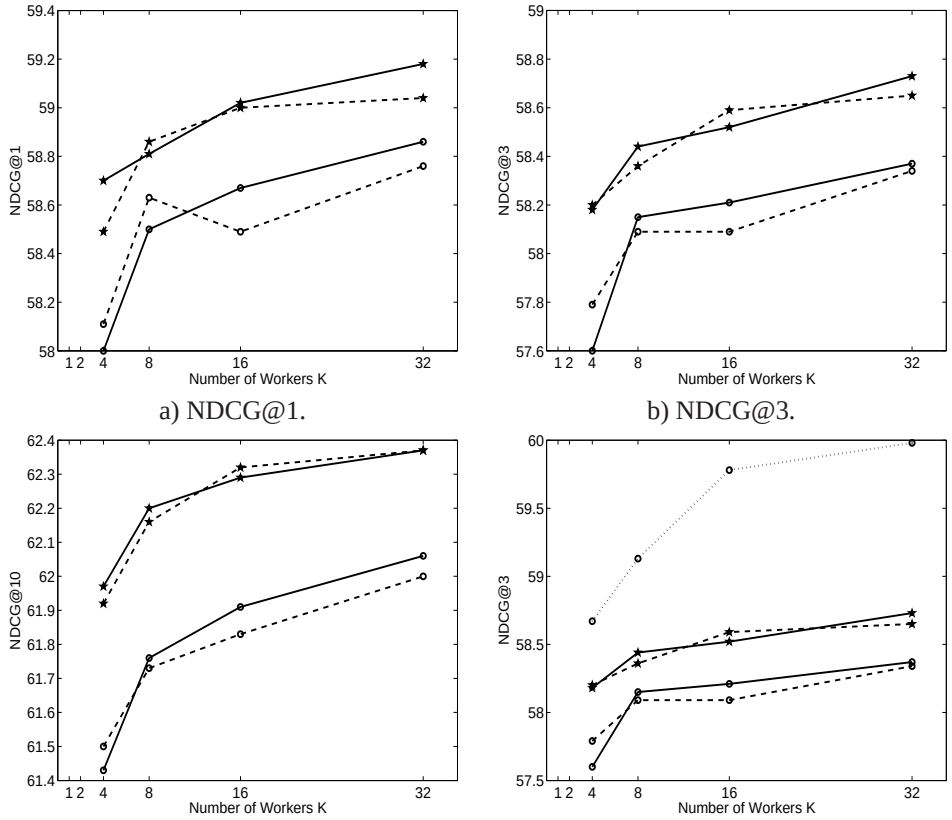


Figure 6: Number of Workers K versus NDCG@1, 3, 10 for full (solid) and sample (dashed) data-distributed LambdaMART. Each worker trains on 7000 overlapping queries (stars). Results from training on 3500 queries per worker (circles) are plotted for comparison. Figure (d) includes centralized LambdaMART (dotted) trained on $3500K$ queries at each x-axis point. Signficant differences are stated in the text.

municated between the master and the workers is independent of the amount of training data; it is dependent on the number of workers and the size of a weak hypothesis. Our full data-distributed algorithm relies on the diversity of each weak hypothesis, yet upon examination of the NDCG scores of the weak hypotheses, we found that during early rounds of boosting the weak hypotheses exhibited diversity, but after only a few rounds of boosting, the weak hypotheses achieved almost identical NDCG scores on the large validation data, indicating that we may be able to eliminate the evaluation step entirely and select a worker at random to produce the weak hypothesis at each iteration.

By eliminating the evaluation step at each iteration, the training time decreases dramatically, since the cost of evaluation is linear in the size of the largest split S_k, and the accuracies are equivalent to choosing the best weak hypothesis based on NDCG evaluation. Thus, our sample selection algorithm can be efficiently applied to billions of samples and achieve comparable accuracy to the full selection strategy. The sample selection algorithm also points to the advantages that an asynchronous distributed approach may have over a synchronous one. Since each worker k trains on a random subset S_k of the training data, then an asynchronous algorithm could assign idle workers different tasks, such as evaluating or training a regression tree for a future iteration. Such an approach could possibly yield improvements in speed or accuracy by taking advantage of the large number of workers available at any given time.

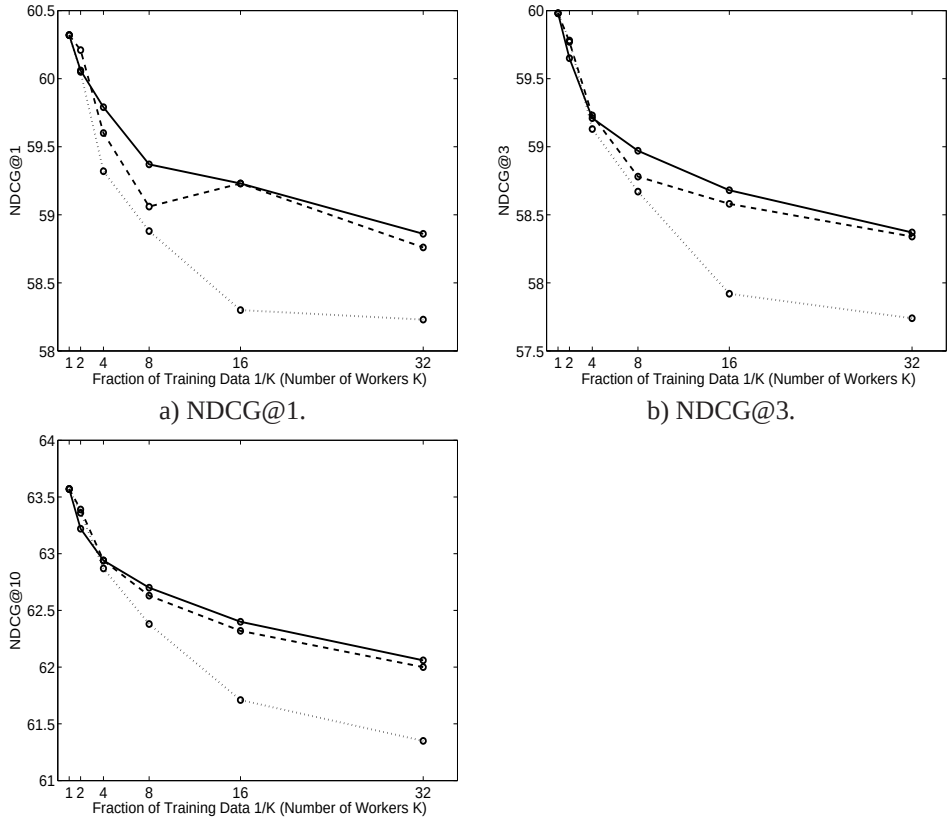
Our sample approach can also be applied to centralized training: at each round of boosting, sample the training data and train a weak hypothesis on that sample. If the complete training dataset fits in memory on a single machine, then the training time will decrease by training on a sample of the data during each boosting iteration. However, if the training data must reside on separate machines, then to train on a single machine, at each round of boosting, the sample must be sent to the machine and then loaded into memory on the machine. The sample must be sampled across all of the machines. The process of communicating the data samples from the many nodes that store the data will be costly and prohibit the use of the algorithm on very large datasets.

E Additional Remarks on Data-distributed LambdaMART

We have shown that our data-distributed approach is a viable method for exploiting additional training data when the main memory of a single machine is exceeded. In this section, we consider the case where the main memory of the workers is not exhausted and we have a fixed amount of training data. One goal of a distributed learning algorithm is to achieve comparable or better accuracy compared to the centralized algorithm, but with much shorter training times. We conduct a series of experiments to determine if our data-distributed approach achieves comparable accuracy with shorter training times compared to the centralized algorithm.

We first determine the effect of decreasing the training data size on the centralized algorithm's accuracy. Let the size of the training set residing on the central machine decrease as $\frac{|S|}{K}$, with increasing values of K. Figure 7 plots the training set size versus NDCG for the centralized model (dotted line). When training on 50% of the training data, the NDCG@1, 3, 10 accuracy compared to training on 100% of the data is statistically similar. It is also noteworthy that as the training set size decreases, the optimal number of leaves decreases, while the optimal learning rate stays constant across the training data sizes (Table 1).

We next determine the accuracy of full and sample data-distributed LambdaMART, where the training data S is split across K workers and each worker contains $\frac{|S|}{K}$ queries. Figure 7 contains the centralized and full and sample data-distributed accuracy results. In the central case, the x-axis indicates the size of the training set on the single node. In the data-distributed cases, the x-axis indicates the number of workers K and correspondingly the amount of training data $\frac{|S|}{K}$ on a given worker. The results indicate that choosing a weak hypothesis among the K nodes, either by full or sample selection, is better than choosing the same weak hypothesis from the same node at each iteration. This is seen by looking at a given value of K: the data-distributed NDCG scores are consistently higher than the centralized NDCG scores and statistically significantly higher for $K \geq 16$. However, there is not a single point on the data-distributed curves that outperforms training on the full data set using the centralized algorithm (the point at $K = 1$). Splitting the data across an increasing number of workers K causes a gradual and continual drop in accuracy, with significant losses compared to the point at $K = 1$ when $K \geq 4$.



a) NDCG@1. b) NDCG@3.

c) NDCG@10.

Figure 7: Number of Workers K vs. NDCG@1, 3, 10 for centralized (dotted) and full (solid) and sample (dashed) data-distributed LambdaMART. Each worker trains on $\frac{|S|}{K}$ queries. The central model was trained on $\frac{|S|}{K}$ queries on a single worker. Significant differences are stated in the text.

The experiment additionally shows that choosing a single weak hypothesis from a worker at random (sample selection) performs similarly to choosing the best weak hypothesis among the K workers based on the evaluation step.

Finally, we determine if training on larger overlapping sets of data achieves comparable accuracy to the central model, but with less training time. We consider $K = 4$ workers and divide the training data S into 4 sets S_1, S_2, S_3, S_4. Each set contains 25% of the full training set. Worker k is assigned sets $S_k + S_{k+1} + S_{k+2}$, and thus produces a weak hypothesis based on 75% of the full training set. At each iteration, we use sample selection to produce the next weak hypothesis in the ensemble. We find that training on 75% of the training queries per node yields equivalent NDCG scores to the central model trained on 100% of the training data, but trains in less than half of the time.