Exponential Family Graph Matching and Ranking

James Petterson, Tibério S. Caetano, Julian J. McAuley and Jin Yu
NICTA, Australian National University
Canberra, Australia

Abstract

We present a method for learning max-weight matching predictors in bipartite graphs. The method consists of performing maximum a posteriori estimation in exponential families with sufficient statistics that encode permutations and data features. Although inference is in general hard, we show that for one very relevant application–document ranking–exact inference is efficient. For general model instances, an appropriate sampler is readily available. Contrary to existing max-margin matching models, our approach is statistically consistent and, in addition, experiments with increasing sample sizes indicate superior improvement over such models. We apply the method to graph matching in computer vision as well as to a standard benchmark dataset for learning document ranking, in which we obtain state-of-the-art results, in particular improving on max-margin variants. The drawback of this method with respect to max-margin alternatives is its runtime for large graphs, which is comparatively high.

1   Introduction

The Maximum-Weight Bipartite Matching Problem (henceforth 'matching problem') is a fundamental problem in combinatorial optimization [22]. This is the problem of finding the 'heaviest' perfect match in a weighted bipartite graph. An exact optimal solution can be found in cubic time by standard methods such as the Hungarian algorithm.

This problem is of practical interest because it can nicely model real-world applications. For example, in computer vision the crucial problem of finding a correspondence between sets of image features is often modeled as a matching problem [2, 4]. Ranking algorithms can be based on a matching framework [13], as can clustering algorithms [8].

When modeling a problem as one of matching, one central question is the choice of the weight matrix. The problem is that in real applications we typically observe edge *feature vectors*, not edge weights. Consider a concrete example in computer vision: it is difficult to tell what the 'similarity score' is between two image feature points, but it is straightforward to extract feature vectors (e.g. SIFT) associated with those points.

In this setting, it is natural to ask whether we could parameterize the features, and use *labeled matches* in order to estimate the parameters such that, given graphs with 'similar' features, their resulting max-weight matches are also 'similar'. This idea of 'parameterizing algorithms' and then optimizing for agreement with data is called *structured estimation* [27, 29].

[27] and [4] describe max-margin structured estimation formalisms for this problem. Max-margin structured estimators are appealing in that they *try* to minimize the loss that one really cares about ('structured losses', of which the Hamming loss is an example). However structured losses are typically piecewise constant in the parameters, which eliminates any hope of using smooth optimization directly. Max-margin estimators instead minimize a surrogate loss which is easier to optimize, namely a convex upper bound on the structured loss [29]. In practice the results are often good, but known convex relaxations produce estimators which are statistically inconsistent [18], i.e., the algorithm in general fails to obtain the best attainable model in the limit of infinite training data. The inconsistency of multiclass support vector machines is a well-known issue in the literature that has received careful examination recently [16, 15].

1

Motivated by the inconsistency issues of max-margin structured estimators as well as by the well-known benefits of having a full probabilistic model, in this paper we present a maximum a posteriori (MAP) estimator for the matching problem. The observed data are the edge feature vectors and the labeled matches provided for training. We then maximize the conditional posterior probability of matches given the observed data. We build an exponential family model where the sufficient statistics are such that the mode of the distribution (the prediction) is the solution of a max-weight matching problem. The resulting partition function is ♯P-complete to compute exactly. However, we show that for *learning to rank* applications the model instance is tractable. We then compare the performance of our model instance against a large number of state-of-the-art ranking methods, including DORM [13], an approach that only differs from our model instance by using max-margin instead of a MAP formulation. We show very competitive results on standard document ranking datasets, and in particular we show that our model performs better than or on par with DORM. For intractable model instances, we show that the problem can be approximately solved using sampling and we provide experiments from the computer vision domain. However the fastest suitable sampler is still quite slow for large models, in which case max-margin matching estimators like those of [4] and [27] are likely to be preferable even in spite of their potential inferior accuracy.

2 Background

2.1 Structured Prediction

In recent years, great attention has been devoted in Machine Learning to so-called *structured predictors*, which are predictors of the kind

$$g_\theta : \mathcal{X} \mapsto \mathcal{Y}, \tag{1}$$

where $\mathcal{X}$ is an arbitrary input space and $\mathcal{Y}$ is an *arbitrary discrete space, typically exponentially large*. $\mathcal{Y}$ may be, for example, a space of matrices, trees, graphs, sequences, strings, matches, etc. This structured nature of $\mathcal{Y}$ is what *structured prediction* refers to. In the setting of this paper, $\mathcal{X}$ is the set of vector-weighted bipartite graphs (i.e., each edge has a feature vector associated with it), and $\mathcal{Y}$ is the set of perfect matches induced by $\mathcal{X}$. If N graphs are available, along with corresponding annotated matches (i.e., a set $\{(x^n, y^n)\}_{n=1}^N$), our task will be to *estimate θ* such that when we apply the predictor g_θ to a new graph it produces a match that is similar to matches of similar graphs from the annotated set. *Structured learning* or *structured estimation* refers to the process of estimating a vector θ for predictor g_θ when data $\{(x^1, y^1), \ldots, (x^N, y^N)\} \in (\mathcal{X} \times \mathcal{Y})^N$ are available. *Structured prediction* for input x means computing $y = g(x; \theta)$ using the estimated θ.

Two generic estimation strategies have been popular in producing structured predictors. One is based on max-margin estimators [29, 27], and the other on maximum-likelihood (ML) or MAP estimators in exponential family models [12].

The first approach is a generalization of support vector machines to the case where the set $\mathcal{Y}$ is structured. However the resulting estimators are known to be *inconsistent* in general: in the limit of infinite training data the algorithm fails to recover the best model in the model class [18, 16, 15]. McAllester recently provided an interesting analysis on this issue, where he proposed new upper bounds whose minimization results in consistent estimators, but no such bounds are convex [18]. The other approach uses ML or MAP estimation in conditional exponential families with 'structured' sufficient statistics, such as in probabilistic graphical models, where they are decomposed over the cliques of the graph (in which case they are called Conditional Random Fields, or CRFs [12]). In the case of tractable graphical models, dynamic programming can be used to efficiently perform inference. ML and MAP estimators in exponential families not only amount to solving an unconstrained and convex optimization problem; in addition they are statistically consistent. The main problem with these types of models is that often the partition function is intractable. This has motivated the use of max-margin methods in many scenarios where such intractability arises.

2.2 The Matching Problem

Consider a weighted bipartite graph with m nodes in each part, $G = (V, E, w)$, where V is the set of vertices, E is the set of edges and $w : E \mapsto \mathbb{R}$ is a set of real-valued weights associated with the edges. G can be simply represented by a matrix (w_{ij}) where the entry w_{ij} is the weight of the edge ij. Consider also a bijection $y : \{1, 2, \ldots, m\} \mapsto \{1, 2, \ldots, m\}$, i.e., a permutation. Then the matching problem consists of computing

2
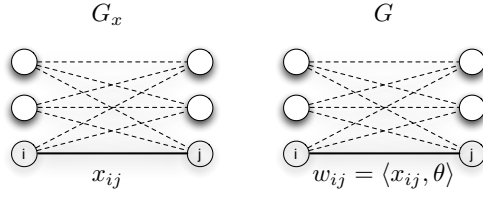
Figure 1: Left: Illustration of an input *vector-weighted* bipartite graph G_x with 3×3 edges. There is a vector x_e associated with each edge e (for clarity only x_{ij} is shown, corresponding to the solid edge). Right: weighted bipartite graph G obtained by evaluating G_x on the learned vector θ (again only edge ij is shown).

$$y^* = \underset{y}{\operatorname{argmax}} \sum_{i=1}^{m} w_{iy(i)}. \tag{2}$$

This is a well-studied problem; it is tractable and can be solved in $O(m^3)$ time [22]. This model can be used to match features in images [4], improve classification algorithms [8] and rank documents [13], to cite a few applications. The typical setting consists of engineering the score matrix w_{ij} according to domain knowledge and subsequently solving the combinatorial problem.

3 The Model

3.1 Basic Goal

In this paper we assume that the weights w_{ij} are instead to be *estimated* from training data. More precisely, the weight w_{ij} associated with the edge ij in a graph will be the result of an appropriate composition of a *feature vector* x_{ij} (observed) and a *parameter vector* θ (estimated from training data). Therefore, in practice, our input is a *vector-weighted* bipartite graph $G_x = (V, E, x)$ ($x : E \mapsto \mathbb{R}^n$), which is 'evaluated' at a particular θ (obtained from previous training) so as to attain the graph $G = (V, E, w)$. See Figure 1 for an illustration. More formally, assume that a training set $\{X, Y\} = \{(x^n, y^n)\}_{n=1}^{N}$ is available, where $x^n := (x_{11}^n, x_{12}^n \ldots, x_{M(n)M(n)}^n)$. Here $M(n)$ is the number of nodes in each part of the vector-weighted bipartite graph x^n. We then parameterize x_{ij} as $w_{iy(i)} = f(x_{iy(i)}; \theta)$, and the goal is to find the θ which maximizes the posterior probability of the observed data. We will assume f to be bilinear, i.e., $f(x_{iy(i)}; \theta) = \langle x_{iy(i)}, \theta \rangle$.

3.2 Exponential Family Model

We assume an exponential family model, where the probability model is

$$p(y|x; \theta) = \exp\left(\langle \phi(x, y), \theta \rangle - g(x; \theta)\right), \text{ where} \tag{3}$$

$$g(x; \theta) = \log \sum_{y} \exp \langle \phi(x, y), \theta \rangle \tag{4}$$

is the log-partition function, which is a convex and differentiable function of θ [31].

The prediction in this model is the most likely y, i.e.,

$$y^* = \underset{y}{\operatorname{argmax}}\, p(y|x; \theta) = \underset{y}{\operatorname{argmax}}\, \langle \phi(x, y), \theta \rangle \tag{5}$$

and ML estimation amounts to maximizing the conditional likelihood of the training set $\{X, Y\}$, i.e., computing $\operatorname{argmax}_\theta p(Y|X; \theta)$. In practice we will in general introduce a prior on θ and perform MAP estimation:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(Y|X; \theta)p(\theta) = \underset{\theta}{\operatorname{argmax}}\, p(\theta|Y, X). \tag{6}$$

Assuming iid sampling, we have $p(Y|X; \theta) = \prod_{n=1}^{N} p(y^n|x^n; \theta)$. Therefore,

$$p(\theta|Y, X) \propto \exp\left(\log p(\theta) + \sum_{n=1}^{N} \left(\langle \phi(x^n, y^n), \theta \rangle - g(x^n; \theta)\right)\right). \tag{7}$$

3

We impose a Gaussian prior on θ. Instead of maximizing the posterior we can instead minimize the negative log-posterior $\ell(Y|X;\theta)$, which becomes our loss function (we suppress the constant term):

$$\ell(Y|X;\theta) = \frac{\lambda}{2}\|\theta\|^2 + \frac{1}{N}\sum_{n=1}^{N}(g(x^n;\theta) - \langle\phi(x^n,y^n),\theta\rangle) \tag{8}$$

where λ is a regularization constant. $\ell(Y|X;\theta)$ is a convex function of θ since the log-partition function $g(\theta)$ is a convex function of θ [31] and the other terms are clearly convex in θ.

3.3 Feature Parameterization

The critical observation now is that we equate the solution of the matching problem (2) to the prediction of the exponential family model (5), i.e., $\sum_i w_{iy(i)} = \langle\phi(x,y),\theta\rangle$. Since our goal is to parameterize features of individual pairs of nodes (so as to produce the weight of an edge), the most natural model is

$$\phi(x,y) = \sum_{i=1}^{M}x_{iy(i)}, \text{ which gives} \tag{9}$$

$$w_{iy(i)} = \langle x_{iy(i)},\theta\rangle, \tag{10}$$

i.e., linear in both x and θ (see Figure 1, right). The specific form for x_{ij} will be discussed in the experimental section. In light of (10), (2) now clearly means a *prediction* of the best match for G_x under the model θ.

4 Learning the Model

4.1 Basics

We need to solve $\theta^* = \operatorname{argmin}_\theta \ell(Y|X;\theta)$. $\ell(Y|X;\theta)$ is a convex and differentiable function of θ [31], therefore gradient descent will find the global optimum. In order to compute $\nabla_\theta\ell(Y|X;\theta)$, we need to compute $\nabla_\theta g(\theta)$. It is a standard result of exponential families that the gradient of the log-partition function is the expectation of the sufficient statistics:

$$\nabla_\theta g(x;\theta) = \mathbf{E}_{y\sim p(y|x;\theta)}[\phi(x,y)]. \tag{11}$$

Therefore in order to perform gradient descent we need to compute the above expectation. Opening the above expression gives

$$\mathbf{E}_{y\sim p(y|x;\theta)}[\phi(x,y)] = \sum_{y}\phi(x,y)p(y|x;\theta) \tag{12}$$

$$= \frac{1}{Z(x;\theta)}\sum_{y}\phi(x,y)\prod_{i=1}^{M}\exp(\langle x_{iy(i)},\theta\rangle), \tag{13}$$

which reveals that the partition function $Z(x;\theta)$ needs to be computed. The partition function is:

$$Z(x;\theta) = \sum_{y}\prod_{i=1}^{M}\underbrace{\exp(\langle x_{iy(i)},\theta\rangle)}_{=:B_{iy(i)}}. \tag{14}$$

Note that the above is the expression for the *permanent* of matrix B [19]. The permanent is similar in definition to the determinant, the difference being that for the latter $\operatorname{sgn}(y)$ comes before the product. However, unlike the determinant, which is computable efficiently and exactly by standard linear algebra manipulations, computing the permanent is a $\sharp$P-complete problem [30]. Therefore we have no realistic hope of computing (11) exactly for general problems.

4.2 Exact Expectation

The exact partition function itself can be efficiently computed for up to about $M = 30$ using the $O(M2^M)$ algorithm by Ryser [25]. However for arbitrary expectations we are not aware of any exact algorithm which is more efficient than full enumeration (which would constrain tractability to very small graphs). However we will see that even in the case of very small graphs we find a very important application: learning to rank. In our experiments, we successfully apply a tractable instance of our model to benchmark document ranking datasets, obtaining very competitive results. For larger graphs, we have alternative options as indicated below.

4

4.3 Approximate Expectation

If we have a situation in which the set of feasible permutations is too large to be fully enumerated efficiently, we need to resort to some approximation for the expectation of the sufficient statistics. The best solution we are aware of is one by Huber and Law, who recently presented an algorithm to approximate the permanent of dense non-negative matrices [10]. The algorithm works by producing *exact samples* from the distribution of perfect matches on weighted bipartite graphs. This is in precisely the same form as the distribution we have here, $p(y|x; \theta)$ [10]. We can use this algorithm for applications that involve larger graphs. We generate K samples from the distribution $p(y|x; \theta)$, and directly approximate (12) with a Monte Carlo estimate

$$\mathbf{E}_{y \sim p(y|x;\theta)}[\phi(x, y)] \approx \frac{1}{K} \sum_{i=1}^{K} \phi(x, y_i). \tag{15}$$

In our experiments, we apply this algorithm to an image matching application.

5 Experiments

5.1 Ranking

Here we apply the general matching model introduced in previous sections to the task of *learning to rank*. Ranking is a fundamental problem with applications in diverse areas such as document retrieval, recommender systems, product rating and others. Early learning to rank methods applied a *pairwise* approach, where pairs of documents were used as instances in learning [7, 6, 3]. Recently there has been interest in *listwise* approaches, where document lists are used as instances, as in our method. In this paper we focus, without loss of generality, on document ranking.

We are given a set of queries $\{q_k\}$ and, for each query q_k, a list of $D(k)$ documents $\{d_1^k, \ldots, d_{D(k)}^k\}$ with corresponding ratings $\{r_1^k, \ldots, r_{D(k)}^k\}$ (assigned by a human editor), measuring the relevance degree of each document with respect to query q_k. A rating or relevance degree is usually a nominal value in the list $\{1, \ldots, R\}$, where R is typically between 2 and 5. We are also given, for every retrieved document d_i^k, a joint feature vector ψ_i^k for that document and the query q_k.

Training At training time, we model each query q_k as a vector-weighted bipartite graph (Figure 1) where the nodes on one side correspond to *a subset* of cardinality M of all $D(k)$ documents retrieved by the query, and the nodes on the other side correspond to all possible ranking positions for these documents $(1, \ldots, M)$. The subset itself is chosen randomly, provided at least one exemplar document of every rating is present. Therefore M must be such that $M \geq R$.

The process is then repeated in a bootstrap manner: we resample (with replacement) from the set of documents $\{d_1^k, \ldots, d_{D(k)}^k\}$, M documents at a time (conditioned on the fact that at least one exemplar of every rating is present, but otherwise randomly). This effectively boosts the number of training examples since each query q_k ends up being selected many times, each time with a different subset of M documents from the original set of $D(k)$ documents.

In the following we drop the query index k to examine a single query. Here we follow the construction used in [13] to map matching problems to ranking problems (indeed the only difference between our ranking model and that of [13] is that they use a max-margin estimator and we use MAP in an exponential family.) Our edge feature vector x_{ij} will be the product of the feature vector ψ_i associated with document i, and a scalar c_j (the choice of which will be explained below) associated with ranking position j

$$x_{ij} = \psi_i c_j. \tag{16}$$

ψ_i is dataset specific (see details below). From (10) and (16), we have $w_{ij} = c_j \langle \psi_i, \theta \rangle$, and training proceeds as explained in Section 4.

Testing At test time, we are given a query q and its corresponding list of D associated documents. We then have to solve the prediction problem, i.e.,

$$y^* = \underset{y}{\operatorname{argmax}} \sum_{i=1}^{D} \langle x_{iy(i)}, \theta \rangle = \underset{y}{\operatorname{argmax}} \sum_{i=1}^{D} c_{y(i)} \langle \psi_i, \theta \rangle. \tag{17}$$

5
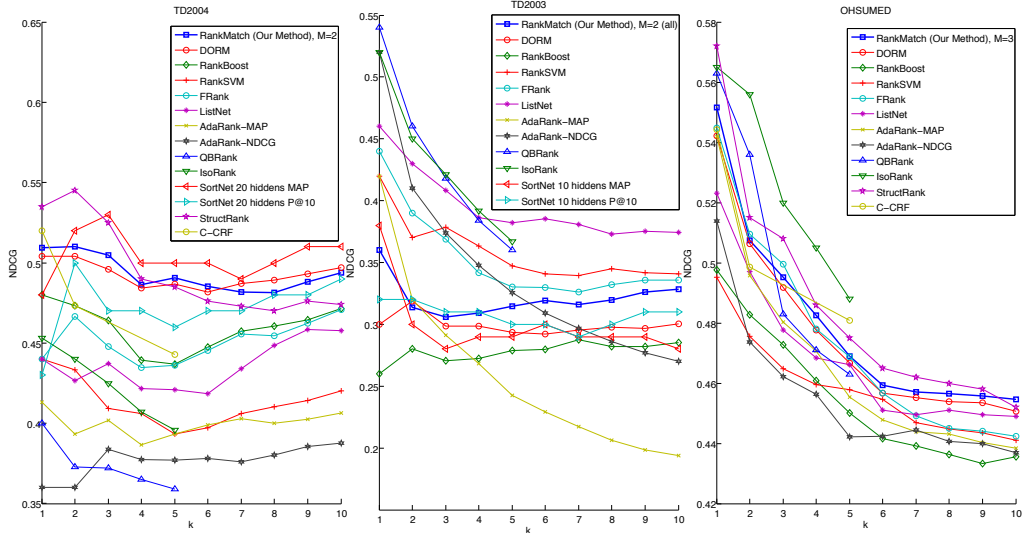
Figure 2: Results of NDCG@k for state-of-the-art methods on TD2004 (left), TD2003 (middle) and OHSUMED (right). This is best viewed in color.

We now notice that if the scalar $c_j = c(j)$, where c is a non-increasing function of rank position j, then (17) can be solved simply by sorting the values of $\langle \psi_i, \theta \rangle$ in decreasing order.[1] In other words, the matching problem becomes one of *ranking the values* $\langle \psi_i, \theta \rangle$. Inference in our model is therefore very fast (linear time).[2] In this setting it makes sense to interpret the quantity $\langle \psi_i, \theta \rangle$ as a *score* of document d_i for query q. This leaves open the question of which non-increasing function c should be used. We do not solve this problem in this paper, and instead choose a fixed c. In theory it is possible to optimize over c during learning, but in that case the optimization problem would no longer be convex. We describe the results of our method on LETOR 2.0 [14], a publicly available benchmark data collection for comparing learning to rank algorithms. It is comprised of three data sets: OHSUMED, TD2003 and TD2004.

Data sets OHSUMED contains features extracted from query-document pairs in the OHSUMED collection, a subset of MEDLINE, a database of medical publications. It contains 106 queries. For each query there are a number of associated documents, with relevance degrees judged by humans on three levels: *definitely*, *possibly* or *not relevant*. Each query-document pair is associated with a 25 dimensional feature vector, ψ_i. The total number of query-document pairs is 16,140. TD2003 and TD2004 contain features extracted from the topic distillation tasks of TREC 2003 and TREC 2004, with 50 and 75 queries, respectively. Again, for each query there are a number of associated documents, with relevance degrees judged by humans, but in this case only two levels are provided: *relevant* or *not relevant*. Each query-document pair is associated with a 44 dimensional feature vector, ψ_i. The total number of query-document pairs is 49,171 for TD2003 and 74,170 for TD2004. All datasets are already partitioned for 5-fold cross-validation. See [14] for more details.

Evaluation Metrics In order to measure the effectiveness of our method we use the *normalized discount cumulative gain* (NDCG) measure [11] at rank position k, which is defined as

$$NDCG@k = \frac{1}{Z} \sum_{j=1}^{k} \frac{2^{r(j)} - 1}{\log(1 + j)}, \qquad (18)$$

where $r(j)$ is the relevance of the jth document in the list, and Z is a normalization constant so that a perfect ranking yields an NDCG score of 1.

---

[1]If $r(v)$ denotes the vector of ranks of entries of vector v, then $\langle a, \pi(b) \rangle$ is maximized by the permutation π^* such that $r(a) = r(\pi^*(b))$, a theorem due to Polya, Littlewood, Hardy and Blackwell [26].

[2]Sorting the top k items of a list of D items takes $O(k \log k + D)$ time [17].

6

Table 1: Training times (per observation, in seconds, Intel Core2 2.4GHz) for the exponential model and max-margin. Runtimes for $M = 3, 4, 5$ are from the ranking experiments, computed by full enumeration; $M = 20$ corresponds to the image matching experiments, which use the sampler from [10]. A problem of size 20 cannot be practically solved by full enumeration.

M	exponential model	max margin
3	0.0006661	0.0008965
4	0.0011277	0.0016086
5	0.0030187	0.0015328
20	36.0300000	0.9334556

External Parameters The regularization constant λ is chosen by 5-fold cross-validation, with the partition provided by the LETOR package. All experiments are repeated 5 times to account for the randomness of the sampling of the training data. We use $c(j) = M - j$ on all experiments.

Optimization To optimize (8) we use a standard BFGS Quasi-Newton method with a backtracking line search, as described in [21].

Results For the first experiment training was done on subsets sampled as described above, where for each query q_k we sampled $0.4 \cdot D(k) \cdot M$ subsets, therefore increasing the number of samples linearly with M. For TD2003 we also trained with all possible subsets ($M = 2(all)$ in the plots). In Figure 2 we plot the results of our method (named RankMatch), for $M = R$, compared to those achieved by a number of state-of-the-art methods which have published NDCG scores in at least two of the datasets: RankBoost [6], RankSVM [7], FRank [28], ListNet [5], AdaRank [32], QBRank [34], IsoRank [33], SortNet [24], StructRank [9] and C-CRF [23]. We also included a plot of our implementation of DORM [13], using *precisely* the same resampling methodology and data for a fair comparison. RankMatch performs among the best methods on both TD2004 and OHSUMED, while on TD2003 it performs poorly (for low k) or fairly well (for high k).

We notice that there are four methods which only report results in two of the three datasets: the two SortNet versions are only reported on TD2003 and TD2004, while StructRank and C-CRF are only reported on TD2004 and OHSUMED. RankMatch compares similarly with SortNet and StructRank on TD2004, similarly to C-CRF and StructRank on OHSUMED and similarly to the two versions of SortNet on TD2003. This exhausts all the comparisons against the methods which have results reported in only two datasets. A fairer comparison could be made if these methods had their performance published for the respective missing dataset.

When compared to the methods which report results in all datasets, RankMatch entirely dominates their performance on TD2004 and is second only to IsoRank on OHSUMED.

These results should be interpreted cautiously; [20] presents an interesting discussion about issues with these datasets. Also, benchmarking of ranking algorithms is still in its infancy and we don't yet have publicly available code for all of the competitive methods. We expect this situation to change in the near future so that we are able to compare them on a fair and transparent basis.

Consistency In a second experiment we trained RankMatch with different training subset sizes, starting with $0.03 \cdot D(k) \cdot M$ and going up to $1.0 \cdot D(k) \cdot M$. Once again, we repeated the experiments with DORM using *precisely* the same training subsets. The purpose here is to see whether we observe a practical advantage of our method with increasing sample size, since statistical consistency only provides an asymptotic indication. The results are plotted in Figure 3-right, where we can see that, as more training data is available, RankMatch improves more saliently than DORM.

Runtime The runtime of our algorithm is competitive with that of max-margin for small graphs, such as those that arise from the ranking application. For larger graphs, the use of the sampling algorithm will result in much slower runtimes than those typically obtained in the max-margin framework. This is certainly the benefit of the max-margin matching formulations of [4, 13]: it is much faster for large graphs. Table 1 shows the runtimes for graphs of different sizes, for both estimators.

5.2 Image Matching

For our computer vision application we used a silhouette image from the Mythological Creatures 2D database[3]. We randomly selected 20 points on the silhouette as our interest points and applied

---

[3]http://tosca.cs.technion.ac.il

7

shear to the image creating 200 different images. We then randomly selected N pairs of images for training, N for validation and 500 for testing, and trained our model to match the interest points in the pairs – that is, given two images with corresponding points, we computed descriptors for each pair i, j of points (one from each image) and learned θ such that the solution to the matching problem (2) with the weights set to $w_{ij} = \langle x_{ij}, \theta \rangle$ best matches the expected solution that a human would manually provide. In this setup,

$$x_{ij} = |\psi_i - \psi_j|^2, \text{ where } |\cdot| \text{ denotes the elementwise difference} \tag{19}$$

and ψ_i is the Shape Context feature vector [1] for point i.

For a graph of this size computing the exact expectation is not feasible, so we used the sampling method described in Section 4.3. Once again, the regularization constant λ was chosen by cross-validation. Given the fact that the MAP estimator is consistent while the max-margin estimator is not, one is tempted to investigate the practical performance of both estimators as the sample size grows. However, since consistency is only an *asymptotic* property, and also since the Hamming loss is *not* the criterion optimized by either estimator, this does not imply a better large-sample performance of MAP in real experiments. In any case, we present results with varying training set sizes in Figure 3-left. The max-margin method is that of [4]. After a sufficiently large training set size, our model seems to enjoy a slight advantage.
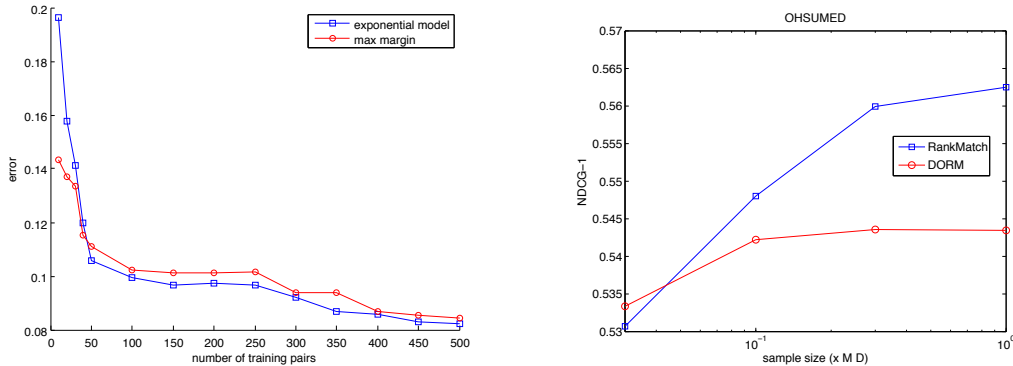
Figure 3: Performance with increasing sample size. Left: Hamming loss for different numbers of training pairs in the image matching problem (test set size fixed to 500 pairs). Right: results of NDCG@1 on the ranking dataset OHSUMED. This evidence is in agreement with the fact that our estimator is consistent, while max-margin is not.

6 Conclusion and Discussion

We presented a method for learning max-weight bipartite matching predictors, and applied it extensively to well-known document ranking datasets, obtaining state-of-the-art results. We also illustrated–with an image matching application–that larger problems can also be solved, albeit slowly, with a recently developed sampler. The method has a number of convenient features. First, it consists of performing maximum-a-posteriori estimation in an exponential family model, which results in a simple unconstrained convex optimization problem solvable by standard algorithms such as BFGS. Second, the estimator is not only statistically consistent but also in practice it seems to benefit more from increasing sample sizes than its max-margin alternative. Finally, being fully probabilistic, the model can be easily integrated as a module in a Bayesian framework, for example. The main direction for future research consists of finding more efficient ways to solve large problems. This will most likely arise from appropriate exploitation of data sparsity in the permutation group.

References

[1] Belongie, S., & Malik, J (2000). Matching with shape contexts. *CBAIVL00*.

[2] Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Trans. on PAMI*, *24*, 509–521.

[3] Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N. & Hulldender, G. (2005). Learning to rank using gradient descent. *ICML*.

8

[4] Caetano, T. S., Cheng, L., Le, Q. V., & Smola, A. J. (2009). Learning graph matching. *IEEE Trans. on PAMI*, *31*, 1048–1058.

[5] Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. *ICML*

[6] Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, *4*, 933–969.

[7] Herbrich, A., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*.

[8] Huang, B., & Jebara, T. (2007). Loopy belief propagation for bipartite maximum weight b-matching. *AISTATS*.

[9] Huang, J. C., & Frey, B. J. (2008). Structured ranking learning using cumulative distribution networks. In *NIPS*.

[10] Huber, M., & Law, J. (2008). Fast approximation of the permanent for very dense problems. *SODA*.

[11] Jarvelin, K., & Kekalainen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, *20*, 2002.

[12] Lafferty, J. D., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic modeling for segmenting and labeling sequence data. *ICML*.

[13] Le, Q., & Smola, A. (2007). Direct optimization of ranking measures. http://arxiv.org/abs/0704.3359.

[14] Liu, T.-Y., Xu, J., Qin, T., Xiong, W., & Li, H. (2007). Letor: Benchmark dataset for research on learning to rank for information retrieval. *LR4IR*.

[15] Liu, Y. & Shen, X. (2005) Multicategory ψ-learning and support vector machine: Computational tools. *J. Computational and Graphical Statistics*, *14*, 219–236.

[16] Liu, Y. & Shen, X. (2006) Multicategory ψ-learning. *JASA*, *101*, 500–509.

[17] Martinez, C. (2004). Partial quicksort. *SIAM*.

[18] McAllester, D. (2007). Generalization bounds and consistency for structured labeling. *Predicting Structured Data*.

[19] Minc, H. (1978). *Permanents*. Addison-Wesley.

[20] Minka, T., & Robertson, S. (2008). Selection bias in the letor datasets. *LR4IR*.

[21] Nocedal, J., & Wright, S. J. (1999). *Numerical optimization*. Springer Series in Operations Research. Springer.

[22] Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial optimization: Algorithms and complexity*. New Jersey: Prentice-Hall.

[23] Qin, T., Liu, T.-Y., Zhang, X.-D., Wang, D.-S., & Li, H. (2009). Global ranking using continuous conditional random fields. *NIPS*.

[24] Rigutini, L., Papini, T., Maggini, M., & Scarselli, F. (2008). Sortnet: Learning to rank by a neural-based sorting algorithm. *LR4IR*.

[25] Ryser, H. J. (1963). *Combinatorial mathematics*. The Carus Mathematical Monographs, No. 14, Mathematical Association of America.

[26] Sherman, S. (1951). On a Theorem of Hardy, Littlewood, Polya, and Blackwell. *Proceedings of the National Academy of Sciences*, *37*, 826–831.

[27] Taskar, B. (2004). *Learning structured prediction models: a large-margin approach*. Doctoral dissertation, Stanford University.

[28] Tsai, M., Liu, T., Qin, T., Chen, H., & Ma, W. (2007). Frank: A ranking method with fidelity loss. *SIGIR*.

[29] Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *JMLR*, *6*, 1453–1484.

[30] Valiant, L. G. (1979). The complexity of computing the permanent. *Theor. Comput. Sci.* (pp. 189–201).

[31] Wainwright, M. J., & Jordan, M. I. (2003). *Graphical models, exponential families, and variational inference* (Technical Report 649). UC Berkeley, Department of Statistics.

[32] Xu, J., & Li, H. (2007). Adarank: a boosting algorithm for information retrieval. *SIGIR*.

[33] Zheng, Z., Zha, H., & Sun, G. (2008a). Query-level learning to rank using isotonic regression. *LR4IR*.

[34] Zheng, Z., Zha, H., Zhang, T., Chapelle, O., Chen, K., & Sun, G. (2008b). A general boosting method and its application to learning ranking functions for web search. *NIPS*.

9