Summary: Efficient Learning of Sparse Ranking Functions

Juliana Louback - jl4354@columbia.edu

December 14, 2014

Introduction to Computational Learning Theory - Final Project

1.Introduction

In the article "Efficient Learning of Sparse Ranking Functions", Google researchers Mark Stevens, Samy Bengio and Yoram Singer (2013) describe a new algorithm for learning ranking functions, Stesinbio[1]. This algorithm was tailored to the application needs of Google's search engine and online advertisement system, both of which use ranking systems with high efficiency demands in addition to accuracy. The objective was to develop a less computationally expensive approach than the current benchmark, while still producing sufficiently accurate results. Stesinbio meets this objective primarily by producing sparse ranking models.

Stesinbio was tested on several datasets and its performance indices was compared to those of RankSVM[6], AdaRank[9], Rankboost[3] and PAMIR[5]. A high level description of these algorithms, presented in chronological order, is included in this summary, followed by a more comprehensive delineation and analysis of Stevens, Bengio and Singer's algorithm. The testing was performed on document datasets and image datasets of varying size, placing particular emphasis on learning speed and the ability to yield compact results.

2. Previous algorithms

2.1 RankSVM[6]

In 2002, Thorsten Joachims of Cornell University presented a Support Vector Machine algorithm for learning search engine retrieval functions. One of the highlights of Joachim's work on this topic was the use of logs of search engine queries and of the links users clicked on as training data, easily obtainable and greatly abundant. Clickthrough rates could not be considered absolutely accurate as it was observed that users typically only scan the first 10 results, approximately. As such, it could be possible that a user never sees a search engine result relative to the query made and it is not possible to claim that a clicked link is of superior rank to all the unseen links. However, inferences about the scanned results could be made relative to each other; if a user clicks on link $t+1$, the link t must've been deemed irrelevant to the user's query.

The objective of Joachim's algorithm is to efficiently find the ranking function $f \in F$ maximizing the empirical Kendall tau rank correlation coefficient[7], and that this function f generalizes beyond training data. An adaptation of Kendall's tau was used as the rank generated by the algorithm will likely not be identical to the target rank. This coefficient is based on the number of concordant and discordant pairs. Two different documents, d_i and d_j are concordant if two orderings r_a and r_b agree on the ordering of d_i and d_j. The idea is to minimize the number of discordant pairs. For $F =$ the class of linear functions, the problem is reduced to finding the weight vector $\vec{w}$ that

1. Minimizes $V(\vec{w}, \vec{\xi}) = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k}$
2. Fulfills the most of the following inequalities:

$$\forall (d_i, d_j) \in r_1 : \vec{w}(\phi(q_1, d_i) - \phi(q_1, d_j)) \geq 1 - \xi_{i,k,1}$$

$$\vdots$$

$$\forall (d_i, d_j) \in r_n : \vec{w}(\phi(q_n, d_i) - \phi(q_n, d_j)) \geq 1 - \xi_{i,k,n}$$

Where (d_i, d_j) are two different documents; $(r_1, ..., r_n)$ are target rank functions; $(q_1, ..., q_n)$ are queries and ξ a non-negative slack variable.

The slack variable is necessary to reach an approximate solution, making allowances for non linearly separable data. This is problem can be solved with a classification SVM where the support vectors are the pairwise difference vectors of the closest pairs. Additionally, the way the problem is formulated allows kernels can be used to apply the SVM on non linear data. One downfall to RankSVM is that it doesn't differentiate between ranking errors. If a ranking function errs and swaps the 3rd and 4th instance rankings, it is still better than one that swaps the 1st and 2nd ranking. It's more damaging to give an incorrect ranking for the top instances. RankSVM sees these two errors as equal.

The evaluation of the algorithm showed superior performance to Google's at the time of publication (2002).

2.2 Rankboost

This algorithm, developed by Yoav Freund, Raj Iyer, Robert E. Schapire and Yoram Singer, combines a group of ranking functions to improve the outcome, based on Freund and Schapire's Adaboost algorithm[2]. A practical application of this method is seen in a user based recommender system such as the one Netflix uses for recommending films: a user X is prompted to rank films he/she has previously seen, after which a recommendation is generated of films X hasn't seen, based on rankings made by other users who made similar rankings as X for films and as such may have similar tastes. Netflix creates a new ranking for X based on a set of rankings from other users. Following is a sketch of the Rankbookst algorithm. Here X denotes the instance space - in the film recommender system, a film is an instance $x \in X$.

```
Rankboost algorithm pseudocode
Given: initial distribution $D$ over $X \times X$.
Initialize: $D_1 = D$.
For $t = 1, ..., T$:
    • Train weak learner using distribution $D_t$.
    • Get weak ranking $h_t : X \to R$.
    • Choose $\alpha_t \in R$.
    • Update: $D_{t+1}(x0, x1) = \frac{Dt(x_0,x_1)\exp(\alpha_t(h_t(x_0)-h_t(x_1)))}{Z_t}$
```

where Z_t is a normalization factor (so D_{t+1} is a distribution).

Output the final ranking: $H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$

It's clear this algorithm follows the Adaboost pattern. In sum, the Adaboost algorithm runs a 'weak' learner multiple times; each iteration creates a new distribution for the algorithm to be run on, this new distribution 'forces' the algorithm to learn something new, incrementally increasing the algorithm's performance. The algorithm is run on all the pairs to be ranked, verifying if the relative rank of each pair is correct.

In this algorithm, greater weight is placed on pairs where previous hypotheses h_i erred; this distribution puts more importance on the current hypothesis learning that pair. The final hypothesis H is a sum of the generated hypotheses h_t multiplied by their respective weights, α_t.The hypothesis weight α_t will be negative if the weight of wrongly ranked pairs is greater than the weight of correctly ordered pairs, decreasing the influence of hypothesis h_t on the final hypothesis. The inverse is also true, a positive α_t indicates a greater weight of correctly ordered pairs and increases the influence of hypothesis h_t. It's not sufficient to correctly order a greater *amount* of pairs; pairs that previously were correctly ranked have lower weight, the hypothesis should focus on previously incorrectly ranked pairs with greater weight.

Rankboost was evaluated with respect to the fraction of disordered pairs, average precision, predicted rank of top (the precision of ranking one highly rated instance), and coverage (the precision of the last ranked instance). Rankboost outperformed Nearest Neighbors on all four measures.

2.3 AdaRank[9]

AdaRank builds on the success of RankSVM (2002) and Rankboost (2003). Developed by Microsoft researchers Jun Xu and Hang Li in 2007, the algorithm is also based on Freund and Schapire's Adaboost algorithm[ada]. Its differential lies in minimizing an exponential loss function directly defined on performance measures; the RankBoost algorithm focuses on minimizing loss functions only loosely correlated to the performance measures. Rankboost at the core seeks to minimize classification error on instance pairs which is equivalent to maximizing a lower bound of the Average Precision, a comparatively weak relation. Similarly to Joachim's RankSVM, AdaRank attempts to optimize the performance measures themselves. To better understand the pseudocode of the AdaRank algorithm, refer to the notation glossary in Table 1.

Table 1: Notations and explanations

Notations	Explanations
$q_i \in Q$	i^{th} query
$d_i = \{d_{i1}, d_{i2}, ..., d_{i,n(qi)}\}$	List of documents for q_i
$y_{ij} \in \{r_1, r_2, ..., r_\ell\}$	Rank of d_{ij} with respect to q_i
$y_i = \{y_{i1}, y_{i2}, ..., y_{i,n(qi)}\}$	List of ranks for q_i
$S = \{(q_i, d_i, y_i)\}_{i=1}^m$	Training set [of size m]
$\vec{x_{ij}} = \Psi(q_{ij}, d_{ij}) \in X$	Feature fector for (q_i, d_j)
$f(\vec{x_{ij}}) \in R$	Ranking model
$\pi(q_i, d_i, f)$	Permutation for q_{ij}, d_{ij}, and f
$h_t(\vec{x_{ij}}) \in R$	t^{th} weak learner
$E(\pi(q_i, d_i, f), y_i) \in [-1, +1]$	Performance measure function

AdaRank algorithm pseudocode

```
Input: $S = \{(q_i, d_i, y_i)\}_{i=1}^m$ and parameters $E$ and $T$
Initialize $P_1(i) = 1/m$
For $t = 1, ..., T$
• Create weak ranker $h_t$ with weighted distribution $P_t$ on training data $S$.
• Choose $alpha_t$
$alpha_t = \frac{1}{2} \cdot \ln \frac{\sum_{i=1}^m P_t(i)\{1+E(\pi(q_i,d_i,f),y_i)\}}{\sum_{i=1}^m P_t(i)\{1-E(\pi(q_i,d_i,f),y_i)\}}$
• Create $f_t$
$f_t(\vec{x}) = \sum_{k=1}^t \alpha_k h)k(\vec{x})$.
• Update $P_{t+1}$
$P_{t+1}(i) = \frac{\exp\{-E(\pi(q_i,d_i,f),y_i)\}}{\sum_{i=1}^m \exp\{1-E(\pi(q_i,d_i,f),y_i)\}}$.
End For
Output ranking model: $f(\vec{x}) = f_T(\vec{x})$.
```

In the pseudocode above, P_t represents a distribution (I assume) so as not to be confused with a document d_i. Analogously to Rankboost, each new distribution increases the weights of queries that previously were not ranked well, encouraging the next run of the weak ranker to learn those queries. Again, the weight α_t corresponds to the 'importance' of h_t, measuring its influence on the final hypothesis. AdaRank, however, differs from Rankboost by having α_t defined by the performance metric E and the distribution P_t (see step 'Choose α_t'), instead of defining α_t by the ratio of incorrectly ranked pairs of instances. Consequently, the aim is not to minimize the classification loss but instead maximize the performance measure E on the training data. Any performance measure can be incorporated for E so long as it is within the range [-1,+1]. In experiments run on main datasets, AdaRank outperformed RankSVM and Rankboost in acctuacy. With regard to efficiency, AdaRank is of order $O((k+T)\cdot m\cdot n\log n)$ for T as number of rounds, m the number of queries and n the number of documents, compared to RankBoost's $O(T\cdot m\cdot n^2)$. The efficiency of RankSVM depends on the inner product of the query and document feature vectors[6].

2.4 PAMIR[5]

The PAMIR algorithm diverges from the algorithms described in previous sections as it was specifically developed for query based image retrieval (hence the acronym PAMIR for Passive-Aggressive Model for Image Retrieval). The creators of the algorithm, Grangier and Bengio, do however cite Joachim's work on RankSVM[6] as part of PAMIR's foundation. At the time publication, the state-of-the-art algorithms needed a image annotation phase to define a label for a given image, followed by the actual text retrieval of the annotated image. These algorithms maximized the *annotation* performance whereas PAMIR seeks to maximize the image *retrieval* performance directly and does not rely on image annotation; it learns a model that *predicts the annotation.*

The query based image retrieval problem was reformulated as a ranking problem. The comparison is evident; the objective is to display results (images) in order of relevance to the query. The algorithm contains a scoring function F_w, that attributes a score correspondent to the image relevance. The higher the relevance, the higher the score. Additionally, F_w must be capable of generalization, ranking images previously unseen. With a defined F_w, ranking is simple. For a query q, a set of pictures P,

$R(q, P)$ the set of pictures relevant to q and $\hat{R}(q, P)$ the set of pictures not relevant to q, the output of F_w should be such that

$\forall p^+ \in (q, P), \forall p^- \in \hat{R}(q, P), F_w(q, p^+) > F_w(q, p^-)$.

What the scoring function F_w does is project the picture into text space with a sub function $f_w(p) = (w_1 \cdot p, ..., w_T \cdot p)$, then measure the match between $f_w(p)$ and the query q by multiplying q by the vector returned by $f_w(p)$. The $f_w(p)$ sub function 'predicts' the image's annotation. This is a linear function, but this algorithm can be applied to non-linear scenarios using kernels. Now we have

$F_w(q, p) = q \cdot f_w(p) = q \cdot (w_1 \cdot p, ..., w_T \cdot p)$. As a query $q = (q_1, ..., q_T)$, and for $\gamma(q, p) = (q_1 p, ..., q_T p)$, $F_w(q, p) = \sum_{t=1}^{T} w_t \cdot (q_t p) = w \cdot \gamma(q, p)$.

The rank is given by the order of projections of $\gamma(q, p)$ along the w vector. Figure 1 shows an example where $p2$ has higher rank than $p3$ that has higher rank than $p1$.

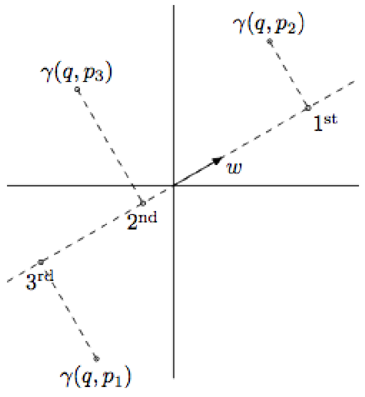


Figure 1: PAMIR rank[5]

The algorithm runs for n iterations, construction a new weight vector w^i at each iteration i. The initial weight vector, w^0 is set to zero. At each iteration, w^i is defined by the previous weight w^{i-1} and the current training example:

$w^i = \arg\min_w \frac{1}{2}||w - w^{i-1}||^2 + cl(w; (q^i, p^{i+}, p^{i-}))$.

In the stage, c is the aggressiveness parameter, influencing how w_i is updated. The parameter l is the loss function:

$l(w; (q^i, p^{i+}, p^{i-})) = max(0, 1 - w \cdot \gamma(q, p+) + w \cdot \gamma(q, p-))$.

This form of update is a trade off between minimizing the training loss and minimizing $||w||^2$. Minimizing $||w||^2$ increases the margin and consequently results in better generalization. However, a decrease in error increases $||w^2||$; this balance is determined by the number of iterations n. The algorithm stops running when there is no significant improvement to performance.

3. Stesinbio

3.1 Problem setting

Google's search engine produces personalized search results for its users. Its targeted online marketing system also requires customized rankings for advertisements. Considering the billions of searches per day[4], it's clear efficiency, in addition to high accuracy, is of the essence. This was a strong incentive to develop the Stesinbio algorithm; existing algorithms such as RankSVM, RankBoost and PAMIR were based on learning over pairs of instances. This is very computationally expensive for large ranking tasks such as those performed by Google. Another important consideration is that in these particular settings, there are very few positive examples and an exorbitant amount of negative examples, which requires algorithms designed for learning sparse ranking functions.

3.2 Algorithm

One of the key differentials of the algorithm is the use of a *domination loss function* as it is expressly suited to learning a sparse ranking. The loss function is based on the number of instances that should be ranked below a certain instance i but aren't. For $\mathscr{D}(i)$ as the set of all instances ranked below instance i by the target ranking function τ and f the ranking function produced by the algorithm, the loss function is

$\ell_{\mathscr{D}}(\mathbf{x}_i; f) = log(1 + \sum_{j \in \mathscr{D}(i)} e^{f(\mathbf{x}_j) - f(\mathbf{x}_i + \Delta(i,j))})$.

Every instance $\mathbf{x}_i$ is represented as a vector in $\mathbb{R}^n$. The parameter $\Delta(i,j) \in \mathbb{R}_+$ is used to mitigate the loss imposed by an instance j that isn't entirely unrelated to i. The article uses the example of a result for the query 'dog' containing a relevant image with a dog, another without a dog and one with many animals including a dog. The image that contains a dog among other animals shouldn't be considered as unrelated as an image with no dog. *Note: All vectors are marked in bold, non-bold represents an element of the vector.*

Initially, the algorithm is presented in a restricted format, limited to the class of linear functions and using only two possible labels for each instance, positive (relevant) or negative (irrelevant), {-1,+1} . In this case, all the instances in $\mathscr{D}(i)$ are labeled -1, and i is labeled i. They are not ranked relative to i, they are simply all irrelevant to the query. Additionally, in this case the parameter $\Delta(i,j)$ is disconsidered.

As f is a linear function of the kind $\mathbf{w} \cdot \mathbf{x}_i$, the loss function is equivalent to

$\ell_{\mathscr{D}}(\mathbf{x}_i; f) = \log(1 + \sum_{j \in \mathscr{D}(i)} e^{w \cdot (\mathbf{x}_j - \mathbf{x}_i)})$

There is no closed form solution for minimizing $\ell_{\mathscr{D}}$; instead a quadratic upper bound is set for the domination loss, a bound optimization technique developed by Roweis and Salakhutdinov. This technique involves the calculation of the gradient and the of Hessian of the domination loss and results in a update method for the r^{th} component of the weight vector $\mathbf{w}$, denoted w_r, with the aim of minimizing the loss upper bound.

$\delta = \frac{1}{m\mathbf{B}r} \sum_{i \notin \mathscr{D}(i)} \frac{\sum_{j \in \mathscr{D}(i)} e^{w \cdot \mathbf{x}_j} x_{i,r} - \sum_{j \in \mathscr{D}(i)} e^{w \cdot \mathbf{x}_j} x_{j,r}}{\sum_{j \in \mathscr{D}(i)} e^{w \cdot \mathbf{x}_j} + e^{w \cdot \mathbf{x}_i}}$.

$w_r \leftarrow w_r + \delta$

For $m =$ the number of relevant instances for the current query; $\mathbf{B}r = \max_{j \in \hat{\mathscr{D}}} x_{j,r}^2$ (the maximum value of the square of the r^{th} feature over instances retrieved). This can be calculated in time linear to the number of instances of each query. Using $Z = \sum_{j \in \mathscr{D}(i)} e^{w \cdot \mathbf{x}_j}$, $\rho_i = w \cdot \mathbf{x}_i$ and $\mu_r = \sum_{j \in \mathscr{D}(i)} e^{\rho_j} x_{j,r}$, the update looks simpler: $\delta = \frac{1}{m\mathbf{B}r} \sum_{i \notin \mathscr{D}(i)} \frac{Z x_{i,r} - \mu_r}{Z + e^{\rho_i}}$.

Following is the pseudocode of the Stesinbio algorithm:

```
initialize: $Z = m, \forall i : \rho_i = 0; \forall r : B_r = max_j x_{j,r}^j$
while not converged do
for $r \in \{1, ..., n\}$ do
        $v_r = 0; \mu_r = 0$ [$v_r$ is the gradient with respect to $r$]
        [1]for $j \in \mathscr{D}(i)$ do
                $\mu_r \leftarrow \mu_r + e^{\rho_j} x_{j,r}$
        end for
        [2]for each $i \notin \mathscr{D}(i)$ do
                $v_r \leftarrow v_r + (Z x_{i,r} - \mu_r)/(Z + e^{\rho_i})$
        end for
                $\delta_r \leftarrow v_r/(mB_r)$ ; $w_r \leftarrow w_r + \delta_r$
        [3]for each $j \in \mathscr{D}(i)$ do
                $Z \leftarrow Z - e^{\rho_j}$ ; $\rho_j \leftarrow \rho_j + \delta_r x_{j,r}$ ; $Z \leftarrow Z + e^{\rho_j}$
        end for
        [4]for each $i \notin \mathscr{D}(i)$ do
                $\rho_j \leftarrow \rho_j + \delta_r x_{j,r}$
        end for
end for
end while
```

Recall the current scenario of limiting the labeling to only positive or negative. A generalization to the algorithm is presented that allows labels to be any value {1,...,K}. The set $G(K)$ represents the top ranked instances and $G(1)$ the lowest. The update rule for δ - and, subsequently, for $\mathbf{w}$ - is slightly modified as the calculation of the gradient has changed:
$\delta = \frac{1}{m\mathbf{B}r} \sum_{k>1} \sum_{i \in G(k)} \frac{Zx_{i,r} - \mu_{k,r}}{Z + e^{\mathbf{w} \cdot \mathbf{x}_i}}$.

Using $Z_k = \sum_{j \in G(k-1)} e^{w \cdot \mathbf{x}_j} + Z_{k-1}$ and $\mu_{k,r} = \sum_{j \in G(k-1)} e^{w \cdot \mathbf{x}_j} x_{j,r}$.

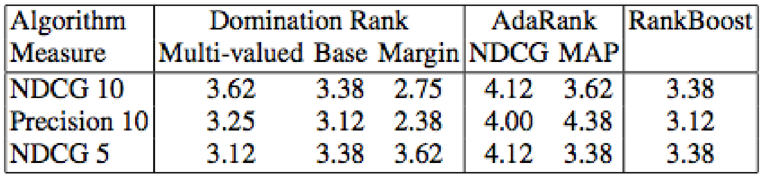
The next extension brings back the concept of using margin requirements $\Delta(i,j)$ to mitigate the loss caused by instances that aren't entirely unrelated. If the margin requirements can be expressed as a sum of two functions, $\Delta(i,j) = s(i) - s(j)$, the algorithm can be successfully extended simply by replacing $e^{w \cdot \mathbf{x}_j}$ by $e^{w \cdot \mathbf{x}_j \pm s(j)}$. After recalculating the gradient, the update to δ is similar to the generalization of the algorithm to allow multiple labels:
$\delta = \frac{1}{m\mathbf{B}r} \sum_{k>1} \sum_{i \in G(k)} \frac{Zx_{i,r} - \mu_{k,r}}{Z + e^{w \cdot \mathbf{x}_j \pm s(j)}}$.

Using $Z = \sum_{j \in \mathscr{D}(k)} e^{w \cdot \mathbf{x}_j - s(j)}$ and $\mu_r = \sum_{j \in \mathscr{D}(k)} e^{w \cdot \mathbf{x}_j - s(j)} x_{j,r}$.

The final extension is the use of a regularization term for feature selection. Although other options are valid, the ℓ_1 norm (sum of the absolute values) of the weight vector is used for regularizing the weights as it is better suited for sparse functions. The ℓ_1 norm is added to the quadratic bound as a penalty, and a new optimization problem and solution is obtained. The update is done according to the following procedure:
$\delta \leftarrow -wiff|g_r - B_r w_r| \leq \lambda$.
Here λ is the penalty added to the bound; $B_r = B_r \sum_{k>1} m(k)$; $g_r = K + \sum_{k>1} \sum_{i \in G(k)} (\sum_{j \in \mathscr{D}(i)} p_i x_{j,r} - x_{i,r})$ The last two values, B_r and g_r correspond to the the sum over all dominating x_i in the multi-labeled case. If the condition is not met:
$\delta = -(g_r + \lambda)/B_r if B_r w_r > 0$ and $\delta = (-g_r + \lambda)/B_r if B_r w_r < 0$.
Adding this regularization step to the algorithm results in compact models with only $\frac{1}{7}$ of the available features for representing images and even better results for documents, which are not as complex.

3.3 Evaluation

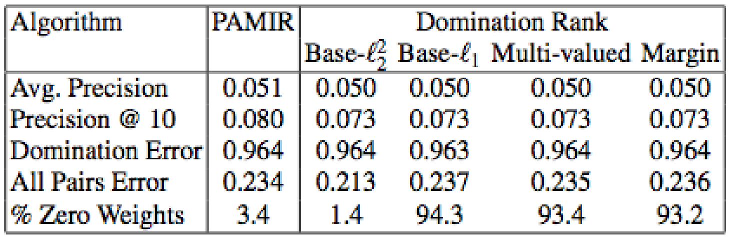
Three major experiments were performed on Stesinbio. Experiment 1 ran Stesinbio on Microsoft's LETOR document dataset, comparing the performance to published results of AdaRank, RankBoost and RankSVM on the same dataset. The comparison to RankSVM was not included as the published results were limited. The results of experiment 1 were not conclusive; it was suggested that this could be due to the modest size of the dataset and overfitting, as the LETOR dataset has long been used in multiple experiments. (See Figure 2 - Domination Rank refers to the Stesinbio algorithm)



Algorithm	Domination Rank			AdaRank		RankBoost
Measure	Multi-valued	Base	Margin	NDCG	MAP	
NDCG 10	3.62	3.38	2.75	4.12	3.62	3.38
Precision 10	3.25	3.12	2.38	4.00	4.38	3.12
NDCG 5	3.12	3.38	3.62	4.12	3.38	3.38

Figure 2: Results for LETOR dataset[8]

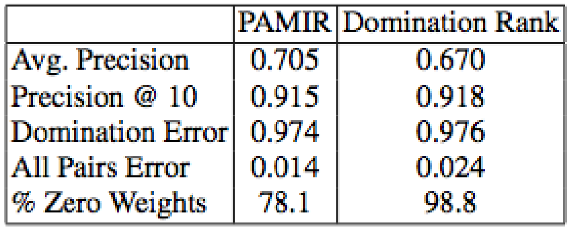
Experiment 2 was run on an a subset of Google's image search database, a dataset of 2.7 million images, 0.4 million of which were used for testing. Stesinbio was compared to the PAMIR algorithm. This time regularization was used to produce compact models and determine whether this truly brings an advantage. The 'label' for the data is based on users click through rates on results returned by queries. It was observed that the multiple layer generalization and the use of margin requirements did not improve the performance on the image dataset. Although the accuracy indices are basically the same (See Figure 3), Stesinbio produces much sparser models while remaining accurate, improving the computational complexity tremendously.



Algorithm	PAMIR	Domination Rank			
		Base-ℓ_2^2	Base-ℓ_1	Multi-valued	Margin
Avg. Precision	0.051	0.050	0.050	0.050	0.050
Precision @ 10	0.080	0.073	0.073	0.073	0.073
Domination Error	0.964	0.964	0.963	0.964	0.964
All Pairs Error	0.234	0.213	0.237	0.235	0.236
% Zero Weights	3.4	1.4	94.3	93.4	93.2

Figure 3: Results for the Google Image dataset [8]

Experiment 3 focused once again on document ranking, using the Reuters RCV1 datasets, composed of 800,000 news articles. As there are 103 topics associated to the articles. these topics were used as the rankings. A few of the topics were removed for being associated with too few articles. The data was randomly divided between test and training. Once again, Stesinbio produced similar accuracy results to PAMIR, but presented the advantage of a significantly sparser model.



	PAMIR	Domination Rank
Avg. Precision	0.705	0.670
Precision @ 10	0.915	0.918
Domination Error	0.974	0.976
All Pairs Error	0.014	0.024
% Zero Weights	78.1	98.8

Figure 4: Results for RCV1 dataset[8]

3.4 Conclusion

The Stesinbio algorithm offers a meaningful improvement of particular value to ranking systems that have high efficiency requirements while handling large amounts of data. If differs from previous algorithms in several ways: it avoids using preference learning over pairs which could incur significant overhead; it can be used for both document and image retrieval; the models produced are incredibly sparse while maintaining competitive accuracy levels.

References

[1] Singer, Yoram. Personal communication. 2014

[2] Freund, Yoav, and Robert E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting." Computational learning theory. Springer Berlin Heidelberg, 1995.

[3] Freund, Yoav, et al. "An efficient boosting algorithm for combining preferences." The Journal of machine learning research 4 (2003): 933-969.

[4] Google Zeitgeist 2012

[5] Grangier, David, and Samy Bengio. "A discriminative kernel-based approach to rank images from text queries." Pattern Analysis and Machine Intelligence, IEEE Transactions on 30.8 (2008): 1371-1384.

[6] Joachims, T. "Optimizing search engines using clickthrough data." In: Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD) (2002)

[7] Kendall, Maurice G. "A new measure of rank correlation." Biometrika (1938): 81-93.

[8] Stevens, Mark, Samy Bengio, and Yoram Singer. "Efficient learning of sparse ranking functions." Empirical Inference. Springer Berlin Heidelberg, 2013. 261-271.

[9] Xu, Jun, and Hang Li. "Adarank: a boosting algorithm for information retrieval." Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2007.