Hierarchical Local Clustering for Constraint Reduction in Rank-Optimizing Linear Programs

Kaan Ataman and W. Nick Street
Department of Management Sciences
The University of Iowa

Abstract

Many real-world problems, such as lead scoring in marketing and treatment planning in medicine, require predictive models that successfully order cases relative to each other. We developed a linear-programming-based learning method, similar to SVMs, that optimizes ranking problems with binary output by maximizing an approximation to area under the ROC curve (AUC). This method consistently outperforms SVMs and other classification methods in terms of ranking. However, our formulation requires a quadratic number of constraints, limiting its application to moderate and large problems. In this paper, we present a localized hierarchical clustering algorithm that reduces the size of the problem by clustering points based on both geometric similarity and class labels. This method dramatically reduces the number of constraints while maintaining high-quality ranking ability.

1 Introduction

Traditional machine learning algorithms are built to minimize classification error. Most of these algorithms also produce a numeric output, such as class membership probabilities or a distance measure to a decision surface. Relative values of such numeric outputs are generally ignored by classification algorithms when determining a class label. However, many real world applications require a ranking for evaluation of cases. For example, it is common in marketing to have a collection of potential customers, or leads, which need to be evaluated on their likelihood of becoming actual customers. Given a finite budget for contacting leads, a marketing manager needs a ranked list of the most likely customers. The traditional data-mining model is built on binary (buyer vs. non-buyer) results, but the problem requires a ranking of how positive each point actually is. In the light of such questions, ranking in binary classification has become a popular machine learning problem that has been addressed extensively in the literature [4, 5, 10].

Over the years, rank optimizing versions of several learning algorithms have been developed as well as methods that directly optimize some ranking metric, such as AUC or equivalently the Wilcoxon-Mann-Whitney (WMW) statistic [18, 7, 3, 15]. Among such methods, optimization-based algorithms are quite promising, especially large margin methods such as support vector machines (SVMs). One drawback of such algorithms is the computational cost, arising from the quadratic form of the optimization problem and formulations that require a quadratic number of either variables or constraints. of these problems are addressed in the literature, yielding promising yet mixed results.

Optimization-based learning algorithms are usually not robust to increasing number of data points. The solution times can increase dramatically with the increasing number of data points making it infeasible to apply such algorithms for many real-world problems. This issue has been addressed in the LP and SVM literature over the years, providing a wide array of speed-up approaches for optimization problems in the context of classification [2, 11, 6]. However, the nature of the ranking problem introduces a different challenge making traditional constraint reduction methods, typically used in classification problems, inappropriate. In this work we present a linear programming formulation that achieves high-quality ranking results, together with a new data reduction method specifically designed for the ranking problem.

2 Background

The most commonly used performance measure to evaluate a classifier's ability to rank instances in binary classification problems is the area under the ROC curve. ROC curves, which plot true positive rate vs. false positive rate, have the favorable property of being independent of class and cost distributions [14], which in the general case are not known in advance. The area under the ROC Curve (AUC) is a single scalar value for classifier comparison. Statistically speaking, the AUC of a classifier is the probability that a classifier will rank a randomly-chosen positive instance higher than a randomly-chosen negative instance. This quantity is equivalent to the Wilcoxon-Mann-Whitney (WMW) statistic [12, 16]. Assuming a dataset with binary class labels where positives need to be ranked higher then negatives, the WMW statistic simply measures the ratio of correctly ordered positive-negative pairs versus all possible pairs. Maximizing this statistic would yield an optimal ordering in data with binary output.

Our formulation of a mathematical program for ranking requires a quadratic number of constraints, limiting the size of solvable problems. Several methods have been developed in the literature to address the problem of a large number of constraints, the most significant being the chunking algorithm [2]. Chunking divides the data into manageable bins and optimizes separately so that the whole problem can be solved in reasonable time. After solving for each chunk, the points that are the most influential are added to the next bin as the algorithm iterates through the whole dataset. Chunking, in its original form, is not a good fit for the ranking problem because unlike in classification problems the influential points in each chunk are not necessarily relevant to other chunks. Unlike classification, constraint reduction in ranking problems where constraints represent pairwise orderings were not addressed consistently in the literature. Most of the methods developed are ad hoc and context-dependent. We believe that the problem of optimization of pairwise ordering has general characteristics that may be exploited to systematically reduce the number of constraints.

The next section explains a linear programming approach to optimize ranking for binary classification problems [1] and introduces a clustering approach to reduce the size of the constraint set in the optimization problem created by pairwise ordering. Section 4 shows results on some benchmark datasets and discussions on further issues. Section 5 concludes the paper.

3 The LP Ranker Algorithm

3.1 Formulation

Let (x, y) be an instance in the training set, X, where x is the data vector and $y \in \{-1, 1\}$ is the class label for that instance. We define the index set of all the points as Θ such that $\Theta = \{l | x_l \in X\}$. We refer to the set of positive points in the data set as P and negatives as N. To optimize the WMW statistic, we would like to have all positive points ranked higher than all negative points: $f(x_i) > f(x_j) \;\; \forall x_i \in P, \forall x_j \in N$, where f is some scoring function. A perfect separation of classes is impossible for most real-world datasets. Therefore an LP is constructed to minimize some error term corresponding to the number of incorrect orderings. We construct a linear program with soft constraints penalizing negatives that are ranked above positives. We define the index set of all positive/negative pairs as Ω such that $\Omega = \{(i, j) | x_i \in P, x_j \in N\}$. Our formulation avoids combinatorial complexity by minimizing the total error, $\sum_\Omega z$, which is the total difference in scores of incorrectly ordered pairs. We select a scoring function, $f(x) = \sum_{l \in \Theta} y_l \alpha_l k(x, x_l)$, such that it assigns a real-valued number as a score to each data point while making the optimization problem continuous. α_l represents a weight for each point in the training set. The function $k(.)$ is a kernel function through which we can induce nonlinearity in the linear constraints. In the objective function we would like to minimize the error, z's, along with the magnitude of the coefficients, α. We minimize α to maximize the separation margin similar to SVMs. The proposed objective function and LP can be obtained as follows:

$$\begin{array}{ll} \min\limits_{\alpha, z} & \sum\limits_{l \in \Theta} \alpha_l + C \sum\limits_{(i,j) \in \Omega} z_{i,j} \\ \text{s.t.} & \sum_{l \in \Theta} [k(x^+, x_l) - k(x^-, x_l)] \quad \geq 1 - z_{i,j}, \quad \forall x^+ \in P, x^- \in N \\ & \qquad \alpha, z \geq 0 \end{array} \tag{1}$$

where e is the unit vector, C is the tradeoff parameter between the two parts of the objective and $z_{i,j}$ represents an error term for each positive/negative pair. In our experiments we used RBF kernel in the form: $(k(x_i, x_j) =$

$e^{-\gamma\|x_i - x_j\|^2}$). The output of the LP gives the optimal set of weights, α^*. The instances with non-zero weights, which we call $ranking\ vectors$, represent the unique points that influence the ranking.

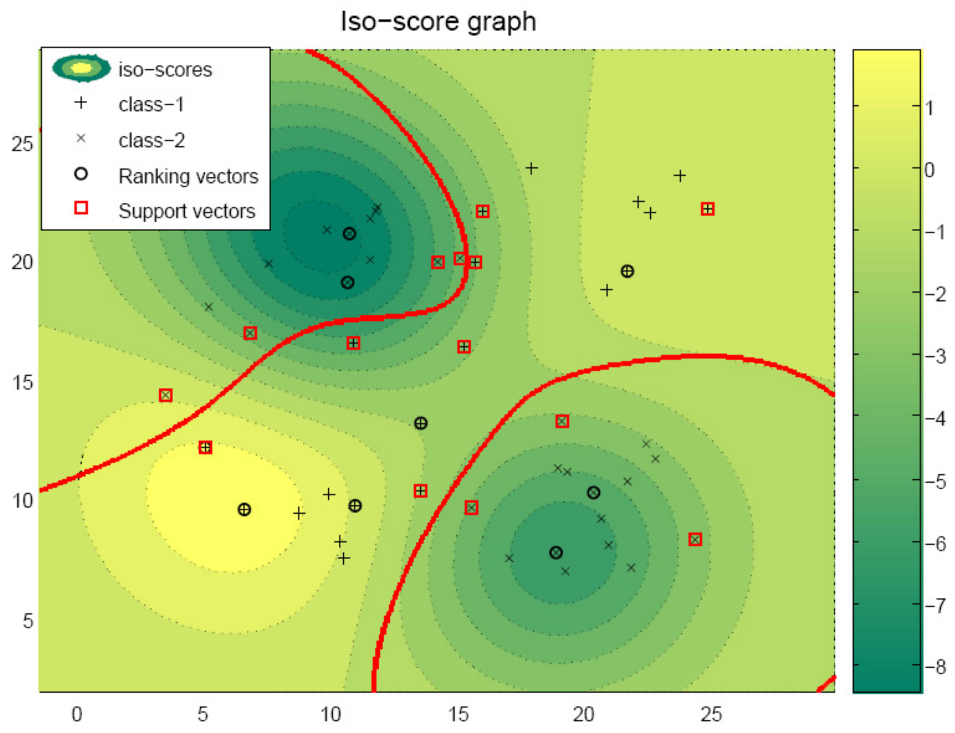

Figure 1: Ranking vectors vs. support vectors

Figure 1 shows a two-dimensional toy problem to illustrate where the ranking vectors appear in the original feature space. We also added regular 2-norm SVM decision boundary and the related support vectors for visual comparison. The figure shows an iso-score graph that displays equal score regions with a colormap, where two points having the same color are ranked equally. Unlike support vectors, the figure shows that the ranking vectors are generally the points from characteristic regions of each class. They may also appear at certain regions near a local positive/negative boundary to bend the iso-score lines just enough such that more correct pairs can be obtained. All local class transition regions are important to the optimization problem. The combination of these two properties provide us the necessary characteristics to construct a clustering approach.

3.2 Hierarchical local clustering for constraint reduction

It can be clearly seen from the LP formulation that the number of constraints generated by the algorithm is equal to number of positives times negatives. This becomes an issue as the dataset sizes reach around 1000 points, especially with balanced class distributions. In this section we propose a clustering approach to reduce the number of data points which in turn will reduce the number of constraints.

As mentioned in the previous section, to obtain the best possible ranking vectors the class transition regions should be kept from being clustered for size reduction as well as retaining the characteristic regions of each class. This enables data points to be clustered in the regions where the data is strictly from one class. Also, a critical decision when constructing a clustering algorithm is the selection of an effective stopping criteria. Excessive clustering would yield a too general algorithm while a conservative approach may not reduce the data to desired sizes. Here we will use class transitions as the stopping criteria for the agglomerative clustering, in order to leave local class transitions intact for the model to detect.

```
initialize distance matrix using all the data points
set each point $x_i$ as a cluster center and initialize $w_i = 1$
while all centers are not flagged
    find closest centers $x_i$ , $x_j$
    if $x_i$ and $x_j$ are from the same class
        merge to the geometric center: $x_k = \frac{w_i x_i + w_j x_j}{w_i + w_j}$
        set $w_k = w_i + w_j$
        remove $x_i$ and $x_j$ from data
        modify distance matrix
    else flag $x_i$ and $x_j$
    end if
end while
```

Figure 2: Agglomerative local clustering pseudocode

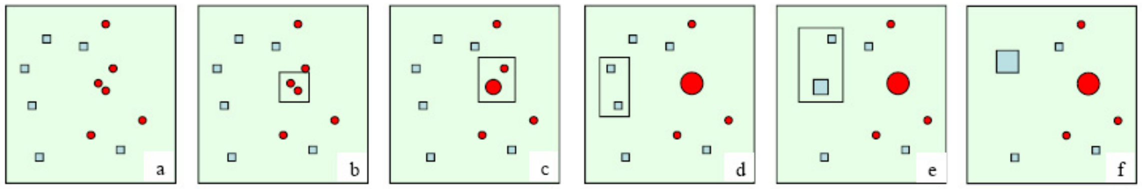

Figure 3: Illustration of data reduction iterations

Our algorithm clusters same-class regions into a single cluster center as long as no point of the opposite class exists in those regions, and then assigns a weight to that center proportional to the number of points clustered. The clustering algorithm is set up to build agglomeratively, that is, the two nearest cluster neighbors are merged if they are from the same class. If they are from the opposite class, those clusters are flagged and cannot be used again. The clustering algorithm stops when all neighboring data points are from opposite classes. Once we have the reduced number of data points, we create the regular constraints to rank positives higher then the negatives. Each constraint will have a significance weight based the weights obtained from the clustering algorithm. Therefore a violated constraint which is enforcing a big chunk of positives over big chunk of negatives will get a much higher penalty in the objective function. The outline of the clustering heuristic is given in Figure 2 and a graphical illustration of the clustering iterations are given in Figure 3 on a simple two dimensional toy problem.

Once the clustering is complete, the objective function is modified to add weights to the error terms. The weight on error $z_{i,j}$ is the product of computed point weights w_i and w_j, representing the fact that the new constraint replaces $w_i w_j$ old ones, one for each positive/negative pair in the two clusters. The new objective is given as:

$$\min_{\alpha,z} \quad \sum_{l\in\Theta} \alpha_l + C \sum_{(i,j)\in\Omega} w_i w_j z_{i,j} \tag{2}$$

4 Results and Discussions

In our experiments we used 11 data sets from the UCI repository. Multi-class data sets are converted to binary classification problems by using one class vs. others scheme. We implemented our algorithm in Matlab [9] and used CPLEX [8] as a solver in the Matlab environment. For performance comparisons with SVM we used the Sequential Minimal Optimization (SMO) algorithm [13] implemented in WEKA [17]. We constructed a grid search to find the best RBF parameter settings for both our algorithm and SVM. We used $C = 1$ and $\gamma = 0.01$ for both algorithms since they tend to do well for both algorithms for each dataset. For all the datasets, we averaged 5$\times$10-fold cross validation results. In our previous work using a similar set up we showed that our ranking algorithm performs favorably versus 2-norm regular SVM when RBF kernels are used without any constraint reduction approach [1].

We ran experiments using the same datasets from previous work to compare reduction in the number of data points and constraints using our clustering heuristic. The results are given in Table 1. According to the results

Table 1: Reduction in number of points and constraints using agglomerative local clustering

Dataset	#points	reduc.#pts	%reduc.	#con.	reduc.#con.	%reduc.
boston	457	81	82.28	18172	1298	92.86
ecoli	305	68	77.70	12336	931	92.45
glass	194	26	86.60	4509	120	97.34
heart	245	119	51.43	14824	3540	76.12
sonar	189	126	33.30	8888	3965	55.39
spectf	318	170	46.54	20240	5376	73.44
wpbc	176	117	33.52	5628	2916	48.19
ionosphere	317	141	55.52	23142	3978	82.81
haberman	277	153	44.77	15022	5642	62.44
liver	312	188	39.74	23711	8755	63.08
wbc	617	80	87.03	86616	1375	98.41
Average			**57.98**			**76.57**

shown the reduction in the size of data is on average 50% causing an average reduction of 75% in the number of constraints. We can see a substantial speed-up on the LP solving times after adopting the clustering heuristic (Table 2). The times logged for clustering algorithm includes the calculation and updates of the distance matrix, time to cluster the data set and finally solve the LP with reduced data. With this approach we were able to solve larger datasets in reasonable time. Also we have compared the AUC performance of our algorithm with constraint reduction vs. SVM (Table 3). Although there is a slight performance loss due to data reduction from our results in the previous work [1], we still obtain better or comparable results to SVM.

Dataset	All(s)	Reduced(s)
boston	26.11	6.91
ecoli	17.24	2.26
glass	1.12	0.65
heart	27.05	2.95
sonar	12.69	3.72
spectf	68.51	9.70
wpbc	9.39	1.97
ionosphere	33.00	4.41
haberman	175.01	14.47
liver	330.58	45.26
wbc	>10 min	14.99

Table 2: Comparison of algorithm run times with all vs. reduced constraints

$\gamma = 0.01, C = 1$	**AUC**	
Dataset	**LP Ranker**	**SVM**
boston	0.9732(0.0034)	0.9535(0.0085)
ecoli	0.9632(0.0051)	0.9387(0.0069)
glass	0.9753(0.0032)	0.9590(0.0035)
heart	0.9112(0.0046)	0.9012(0.0039)
sonar	0.8735(0.0079)	0.7950(0.0092)
spectf	0.9127(0.0006)	0.8835(0.0055)
wpbc	0.7624(0.0299)	0.7649(0.0064)
ion	0.9554(0.0079)	0.9196(0.0034)
haberman	0.5387(0.0304)	0.7028(0.0164)
liver	0.5647(0.0267)	0.6912(0.0082)
wbc	0.9955(0.0006)	0.9949(0.0004)
(W,L,T)	(7-2-2)	

Table 3: LP Ranker with clustering vs. SVM

5 Conclusions

In this paper we introduced a hierarchical local clustering algorithm to tackle the issue of quadratic expansion of constraints in rank optimization. Our rank optimization algorithm uses linear programming to maximize an approximation to WMW statistic to correctly rank positive/negative pairs. The results presented show that the rank optimization algorithm with the clustering approach performs better in general against regular 2-norm SVMs while the percent reduction of the number of constraints was around 75% on average for all the datasets. However, the size of the constraint set is still prohibitive for larger problems with balanced class distributions. In future work we will look into modifications of the clustering approach proposed in this work to further reduce the problem size.

References

[1] K. Ataman, N. Street, and Y. Zhang. Learning to rank by maximizing AUC with linear programming. In *International Joint Conference on Neural Networks (IJCNN)*, pages 123–129, 2006.

[2] P.S. Bradley, O. Mangasarian, and D. Musicant. Optimization methods in massive datasets. In J. Abello, P.M. Pardalos, and M.G.C. Resende, editors, *Handbook of Massive Datasets*, pages 439–471. Kluwer Academic, 2001.

[3] U. Brefeld and T. Scheffer. AUC maximizing support vector learning. In *Preceedings of ICML 2005 workshop on ROC Analysis in Machine Learning*, 2005.

[4] R. Caruana, S. Baluja, and T. Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 959–965. The MIT Press, 1996.

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

[6] G. Fung and O.L. Mangasarian. Data selection for support vector machine classifiers. In R. Ramakrishnan and S. Stolfo, editors, *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 64–70. ACM, 2000.

[7] A. Herschtal and B. Raskutti. Optimising area under the ROC curve using gradient descent. In *ICML '04: Twenty-First International Conference on Machine Learning*. ACM Press, 2004.

[8] ILOG. *CPLEX User Manual*. ILOG, 2002.

[9] The Mathworks Inc. *Matlab: The Language of Technical Computing*. The Mathworks Inc., 1997.

[10] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142. ACM Press, 2002.

[11] O.L. Mangasarian and D.R. Musicant. Large scale kernel regression via linear programming. *Machine Learning*, 46(1/3):255–269, 2002.

[12] H.B. Mann and D.R. Whitney. On a test whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statististics*, 18:50–60, 1947.

[13] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schlkopff, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning.* MIT Press, 1998.

[14] F.J. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. *Knowledge Discovery and Data Mining*, pages 43–48, 1997.

[15] A. Rakotomamonjy. Optimizing area under ROC curve with SVMs. In *Proceedings of the ROC Analysis in Artificial Intelligence*, pages 71–80, 2004.

[16] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[17] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.

[18] L. Yan, R.H. Dodier, M. Mozer, and R.H. Wolniewicz. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In *International Conference on Machine Learning*, pages 848–855, 2003.