Determining Influence in Social Media (Twitter): Preference Learning Approaches to Object Ranking

Darrell Aucoin, Xiaomei Yu, Peter Starszyk

August 10, 2018

Abstract

A very prominent area in machine learning is classification, where the challenge lies in learning how to classify instances into their true categories. However, a less popular but very important problem is learning how to rank a set of labels or objects given some input vector containing information regarding preferences. This paper focuses on a natural extension to classification, Preference Learning, where the problem lies in estimating a suitable ranker. We will work with a dataset consisting of Twitter users where each instance is a pairwise comparison between two users and an associated human judgement indicating which of them is more influential. Not surprisingly, this human judgement can be subjective and in some cases inconsistent.

Preference Learning attempts to account for such problems by taking a machine learning approach to learning a model which predicts the preferences of an underlying decision maker. In the process of solving this problem, the paper will give an overview of key ideas we learned about Preference Learning, then argue how and why it applies to the problem in this particular dataset. In the latter portion of the paper, we will discuss the modeling employed which turns out to be alterations of models we are already familiar with from Classification.

Keywords: Preference Learning, Object Ranking, Bootstrap, Cohen's Method, SVOR

1 Introduction

The dataset chosen for this project is entitled "Influencers in Social Networks" provided by Peerindex and offered through Kaggle, consisting of $1,172$ unique Twitter users (call this set $\mathcal{X}_U$) and is of dimensionality $5,500 \times 22$ of pairwise comparisons in its raw form with a response vector $\mathcal{T} \in \{0,1\}$. Each row is a comparison between the feature vectors of two users, user A (x_A) and user B (x_B), which comprises 11 pre-computed non-negative attributes for both users (follower count, retweets received, mentions received, listed count, etc) and an indication (generated by human judgment at Twitter) of "1" if $x_A \succ x_B$ (x_A is more influential than x_B), "0" otherwise. The goal is to learn a function which, given feature vectors of two users, predicts Twitter's internal judgment on which of the two users is more influential. This paper is written chronologically, outlining the scientific approach we used to learn functions suitable for accomplishing the aforementioned goal.

1.1 Initial Problems

Our initial approach to the dataset was naturally to treat it as a classification problem, but there were several flaws in this approach:

1. Classification of an object has an absolute, definitive meaning. Preferences, on the other-hand, only have meaning in comparing of two or more objects.
2. Classification has a fixed number of classes and the number of ranks for a given data set is variable with the number of objects being compared.
3. Even if we create a classifier to identify pairwise preferences and use this to create a ranking of objects, this ranking is possibly transitive. For instance, for objects A, B, C we can possibly find pairwise preferences such $x_A \succ x_B \succ x_C$ and $x_C \succ x_A$.

Upon investing solutions to these problems we found preference learning, a subfield of machine learning, that uses techniques similar to classifiers (with alterations) but tackles the above problems.

1.2 Preliminary Data Exploration and Processing

To get a clear understanding of the data, we converted every user's attributes to the logarithmic scale because the magnitude of some features were far larger than others (i.e.. number of followers went over 30M in some cases). We chose four user attributes we intuitively expect would have some positive relationship with influence: i) follower count, ii) posts, iii) mentions received, iv) retweets received. Figure 1 plots 40 randomly sampled users (represented by the glyphs) color coded by the ranges of follower counts that they have. The four axes on these glyphs describe the magnitude of the four selected attributes for that user, and they are plotted along the first two principal components.

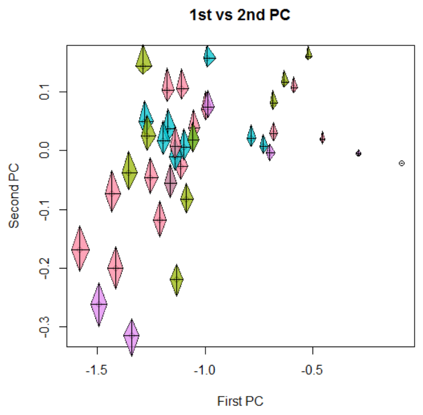

Figure 1: Radial Axes on Top 2 PC

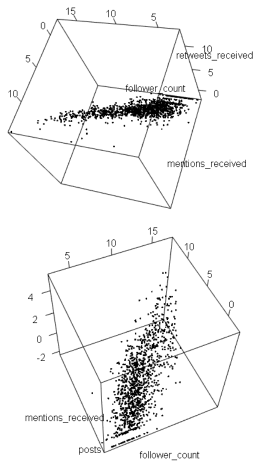

Figure 2: 3D Plot of Chosen Attributes

Figure 1 shows no clear pattern in color distribution along the PCs. Furthermore, as the glyphs move up along both PCs, the four attribute axes seem to shrink at the same rate for all of the users. This would suggest that the four attributes chosen share

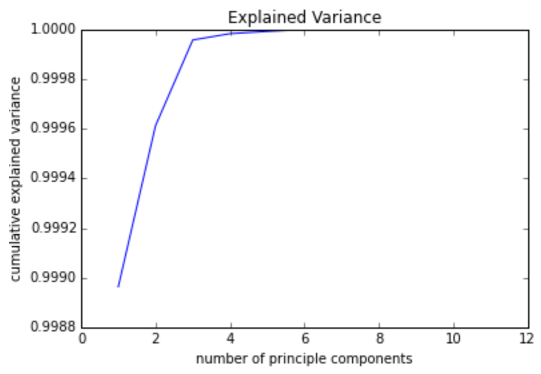

Figure 3: Explained Variance

some sort of dependencies. The 3-D plots of these attributes (Figure 2) further shows collinearity between the attributes on the log scale, which again suggests that the raw features in the dataset have some sort of dependencies. This in mind, we would have to consider penalizing our models appropriately in an attempt to remedy collinearity as much as possible. This linear nature is reinforced when we look at Figure 3 (Explained Variance), the first principle component of the raw data explains 99.9% of the variance in the user features.

Processing the data before proceeding further mainly consisted of

1. Removing contradictions (some instances indicated that $x_A \succ x_B$ and $x_A \prec x_B$ simultaneously)

2. Making the training data symmetric (for every pairwise comparison $x_A, x_B \implies \{1, 0\}$ we added the reversed pairwise comparison $x_A, x_B \implies \{0, 1\}$) so that our models are less likely to predict contradictions.

3. Removal of duplicates within the data. In the data set there were rows where $x_A \succ x_B$ and then those exact same rows are repeated.

4. Given separate attribute vectors for a pair of users, we needed a single vector that would serve as the basis of comparison between the two users. As such, given a pair of users $x_A, x_B \in \mathbb{R}^{11,+}$ we trained our models on the differences of their attributes $x_A - x_B \in \mathbb{R}^{11}$. The reason for this is that we assume their attribute differences contain information regarding their relative influences.

2 Preference Learning[5]

2.1 Preference Learning Types[5]

Preference Learning is a branch of Machine Learning, whereby the ultimate goal is to learn a function that ranks instances based on some criterion. Three main approaches to Preference Learning are:

Label Ranking: There exists an instance space $X = \{x_i\}_{i=1}^{N}$ (ex. Attributes of N movie watchers) and an associated set of finite labels $Y = \{y_j | j = 1, \ldots, m\}$ (ex. Set of movies) and the model is trained on a listed preference for each x_i. So $y_j \succ_{x_i} y_k$ means that watcher i prefers movie j over movie k.

Instance Ranking: There exists an instance space $X = \{x_i\}_{i=1}^{N}$ and label set Y. Each label $y_k \in Y$ is associated with an instance $x_k \in X$. There exists a total order $y_1 \succ \cdots \succ y_N$ which orders the instances $x_1, \ldots, x_N$ based on some criterion or quality (ex. All Football teams are ranked by their chances to enter the playoffs).

Object Ranking: Similar to instance ranking but there is no total order in the training data. Preference information is given by pairwise preferences $x_A \succ x_B$. (ex. Given a pair of Twitter users, which is more influential). Since the training data does not show the true order for its' instances, the data is considered semi-supervised.

2.1.1 Further Information on Object Ranking[1,5]

Since our data is in the form of pairwise preferences, indicating that we should use object ranking on our twitter data.

Object ranking into orders has some similarities to ordinal regression on ordered categories (i.e.. "good", "fair", "poor"). However, there as significant differences between the two:

1. Ordered categories have an absolute meaning while orders or ranks have only relative meaning ($x_i \succ x_j$)

2. Ordered categories have an limited number of grades while orders or ranks represent a difference in quality of some difficult to measure but comparable quality (the quality is difficult to put as a real number value but comparisons between two objects with that quality is possible). Thus, for object ranking is only dependent on the number of objects to be ranked.

2.2 Learning Techniques[5]

To perform these three Preference Learning methods, two main techniques are used:

Utility Function: Given a set of instances $\{x_A, x_B, \ldots, x_Z\}$ their preference relation can be determined by some function that maps $\{x_A, x_B, \ldots, x_Z\} \mapsto A, B, \ldots, Z \in \mathbb{R}$ such that $A > B \implies x_A \succ x_B$.

Preference Relation: Given a pair of instances $\{x_A, x_B\}$ their preference relation can be determined by a binary indicator. The issue here is that there may not exist a unique order for a large instance space.

This in mind, it is clear that the main challenge associated with our dataset is that of ranking, as opposed to classification and can be reduced to object ranking specifically. When considering modeling strategies we focused on employing utility functions as much of the literature argued that many preference relation techniques prove to be computationally NP-hard. For the majority of the remaining paper, we will discuss several object rankings methods we employed, some of which were a combination of methods proposed in Preference Learning literature and altered classification methods from the course.

2.3 Loss functions[5]

There are several Error Loss functions that can be minimized to learn a ranker in Preference Learning. We mainly focused on two methods which are more suitable to our problem: **Kendall's Distance** and **Gamma coefficient**.

Kendall's distance counts the total number of incorrectly ranked pairs. For example, if the true ranking is A, B, C but we estimate B, A, C then our estimate still shows the correct ordering of A, C and B, C but not A, B so the Kendall's distance is 1.

The gamma coefficient is the normalized distance between the number of correctly ranked pairs, denoted by d, and the number of incorrectly ranked pairs, which is $\overline{d}$. Then the gamma coefficient is:

$$\gamma = \frac{d - \overline{d}}{d + \overline{d}} \in [-1, 1]$$

This in mind, it is clear that the main challenge associated with our dataset is that of ranking, as opposed to classification and can be reduced to object ranking specifically. When considering modeling strategies we focused on employing utility functions as much of the literature argued that many preference relation techniques prove to be computationally NP-hard. For the majority of the remaining paper, we will discuss several object rankings methods we employed, some of which were a combination of methods proposed in Preference Learning literature and altered classification methods from the course.

3 Bootstrap[2,3,4]

Our first idea was to treat the sample data as if it were the population of all Twitter users and proceed using a Bootstrap approach. The algorithm would sample n pairwise users with replacement K times. For each of the K times, a scoring function would be fit to the sample and produce a total order all of the users in the sample based on these scores. After K samples were taken, we would sum up each user's scores and divide by the total number of times they were sampled. Finally, a total order of all users based on these aggregate scores would be generated.

The justification for using a Bootstrap approach is that if a true estimator (in this case rank estimator) exists and is highly complex, it offers a reliable and fast approach to inference on the properties of this estimator given the data.

When obtaining the scores for each of the N users, we employed both a logistic regression that models the measure $P\left[x_A \succ x_B \mid x_A, x_B\right]$ and a and a Neural Network. The Logistic model was regularized by penalizing the Binomial Likelihood using the L_1 Lasso penalty.

$$\hat{\beta}_\lambda = \arg\max_{\beta} \sum_{i=1}^{n} \log\left(P\left[x_i \mid \beta\right]\right) - \lambda \sum_{j=1}^{p} \left|\beta_j\right|$$

The optimal β was by running 10-Fold Cross Validation on the training set at a regularization rate $\lambda = 0.0039$ (Figure 4), which forced 5 coefficients to zero (Figure 5).

With regards to the Neural Network, we made two modifications to the plain model. First we included a penalization for the weights in an attempt to remedy collinearity. Second, we replaced the original error loss function proposed in class $|y-\hat{y}|^2$ with the Kendall's Distance function $K\left(y,\hat{y}\right)$. The Kendall's distance loss is more appropriate to learn a ranker than the the original loss because it reflects the inaccuracies in ranking estimates. The above changes implied the modified algorithm:

$$u_{i,j}^{\text{new}} \leftarrow u_{i,j}^{\text{old}} - \rho\left[K\left(y,\hat{y}\right) + \lambda u_{i,j}^{\text{old}}\right]$$

Having estimated the appropriate Logistic and Neural Network parameters to model the scores of sampled Twitter users, we then incorporated them into the Bootstrap algorithm which would estimate a total ranked order of the users. This total order would then be queried to ultimately estimate the particular pairwise preferences in our data. The following table summarizes results of Bootstrap with an underlying Logistic scoring model and an underlying Neural Network scoring model.

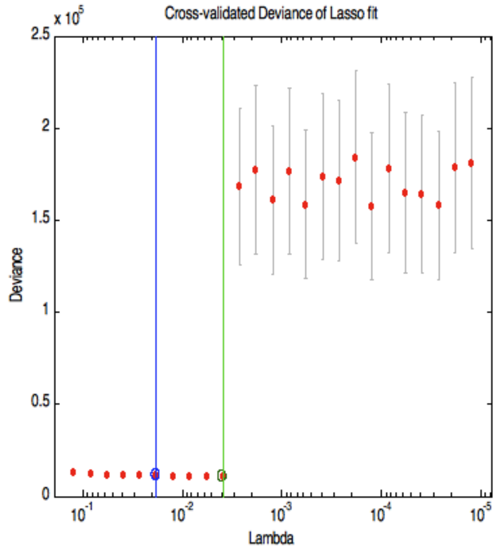

Figure 4: 10-fold CV using 25 penalty rates

Bootstrap Algorithm	Train Accuracy
Logistic Scorer	0.7292
Neural Network Scorer	0.7504

Naturally we looked for alternative algorithms for two reasons: 1) The above Accuracy rates still suggest that we can do better and 2) the Bootstrap technique is generally thought of as asymptotically consistent for an estimator, however may not reliable under finite samples as we have here.

4 Cohen's Greedy Method[2,3,4,5]

Given a set of users X_U (suppose $|\mathcal{X}_U| = N$), the greedy algorithm is applied to estimate the ranked order of these users $\hat{O}_U$ by computing the pairwise preference confidence between all users by some utility function mapping; given two users $\{x_A, x_B\}$, their pairwise preference confidence is computed as $P\left[x_A \succ x_B \mid x_A, x_B\right]$. The algorithm proceeds as follows:

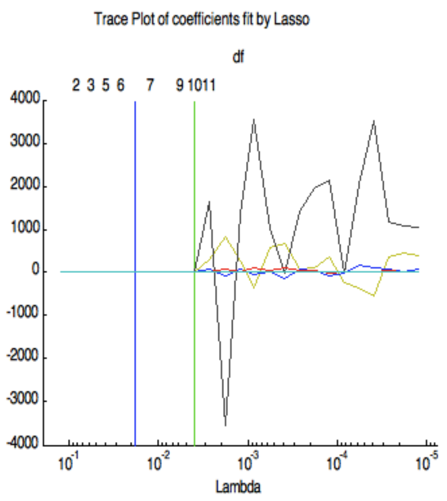

Figure 5: Lasso coefficient shrinkages

Algorithm 1 Cohen's Greedy Algorithm

```
$\hat{O}_U \leftarrow \emptyset$ % Begin by defining an empty ordered set
for all $x \in \mathcal{X}_U$ do:
    score[x] $\leftarrow \sum_{x' \in X_U} P\left[x \succ x' \middle| x, x'\right]$
    % For all N users, sum up their preference relations with the N-1 users
end for
while $\mathcal{X}_U \neq \emptyset$:
    $x_{top} \leftarrow \arg\max_x$ (score) % Find top scorer from current score[x] vector (score[x] $\in \mathbb{R}^{N-i+1}$ for $i^{th}$ iteration)
    $\hat{O}_U$[bottom] $\leftarrow x_{\text{top}}$ % Append this iteration's top scorer to the bottom of the current ordered set
    $X_U \leftarrow X_U \backslash \left\{X_{\text{top}}\right\}$ % Update the current user set by eliminating the previous top scorer
    % Before reiterating, the scores are adjusted by eliminating the effect of the previously extracted top scorer
    for all $x \in \mathcal{X}_U$ do:
        score[x] $\leftarrow$ score[x] - $P\left[x \succ x_{\text{top}} \middle| x, x_{\text{top}}\right]$
% Adjusting the remaining scores by removing $x_{\text{top}}$
    end for
end while
return $\hat{O}_U$
```

The justification for employing a greedy algorithm was firstly due to their proven optimality. Two proof approaches found in literature are the Greedy Algorithm stays ahead and the Exchange Argument[6]. Secondly, greedy algorithms offer a much faster running time at usually an infinitesimally incremental cost which is far more attractive than potentially NP-hard Preference Relation methods.

Cohen's Greedy Algorithm	Train Accuracy
Logistic Scorer	0.7302
Neural Network Scorer	0.7465

We can see that the above accuracy rates are very comparable between the greedy algorithm and the Bootstrap. This did not surprise us at this point because regardless of the sorting algorithm, we would intuitively expect the accuracy of the overall model to be dominated by the accuracy of the scoring functions rather than the algorithms applied to these scores. Finally, we looked to completely new alternatives to improve our accuracies.

5 Ranking SVM (SVOR)[1]

5.1 Overview[1]

To understand how SVOR works, it is best to assume that the training data is in the form $(x, y) \in X \times Y$, that is data applicable to instance ranking. This can then later be decomposed into pairwise comparisons allowing object ranking to be done.

The basic premise of Ranking SVM (a.k.a. Support Vector Ordinal Regression or SVOR) is to find a utility function that best fits the preferences induced by an unknown distribution $p(x, y)$. This is done by assuming that there are some scoring functions ($f : x \mapsto \mathbb{R}$) linear to the space of X, or space transformed by some kernel, which maps an instance pair $x_i, x_j \in X$ such that $x_i \succ x_j \iff f(x_i) > f(x_j)$.

We can break this scoring function into q intervals corresponding to the ranks $\theta(r_i)$, with $\theta(r_0) = -\infty, \theta(r_q) = \infty$, each interval corresponding to a rank. Using the theorem of **A Margin Bound for Ordinal Regression** given below, we can find the maximum margin between these ranks to find our scoring function $f(x)$.

Theorem 1. ***A Margin Bound for Ordinal Regression****:* *Let p be a probability measure on $\mathcal{X} \times \{r_1, \ldots, r_q\}$, let (X, Y) be a sample of size m drawn iid from p. Let σ be an permutation of the numbers $1, \ldots, m$. For each function $g : \mathcal{X} \mapsto \{r_1, \ldots, r_q\}$ there exists a function $f \in F$ and a vector θ such that*

$$g(x) = r_i \iff f(x) \in [\theta(r_{i-1}), \theta(r_i)]$$

Let the fat-shattering dimension of the set of functions F be bounded above by the function $afat_F : \mathbb{R} \mapsto$

$\mathbb{N}$. *Then for each function g with zero training error, i.e.*

$$\sum_{i=1}^{m-1} C_g\left(\sigma\left(i\right), \sigma\left(i+1\right)\right) = 0$$

and

$$p_f = \min \Omega\left(y_{\sigma(i)}, y_{\sigma(i+1)}\right) \left|f\left(x_{\sigma(i)}\right) - f\left(x_{\sigma(i+1)}\right)\right|$$

with probability $1-\delta$

$$R_{pref}(g) \leq \frac{2}{m-1}\left(a+b\right)$$

$$a = k \log_2\left(\frac{8e\left(m-1\right)}{k}\right)\log_2\left(32\left(m-1\right)\right)$$

$$b = \log_2\left(\frac{8\left(m-1\right)}{\delta}\right)$$

where $k = afat_F\left(\frac{\rho_f}{8}\right) \leq e\left(m-1\right)$.

(Ralf Herbrich, Thore Graepel, and Klaus Oberma 2000)

5.2 SVOR Breakdown[1]

Let us assume that we have an outcome space of $\mathcal{Y} = \{r_1, \ldots, r_q\}$ such that $r_q \succ_{\mathcal{Y}} r_{q-1} \succ_{\mathcal{Y}} \cdots \succ_{\mathcal{Y}} r_1$. Because $\mathcal{Y}$ contains a finite number of stochastic orderings corresponding to the vector space $\mathcal{X}$, then for vector pair $x_1, x_2 \in \mathcal{X}$, then one of following must hold $\forall x_i, x_j$:

$$\begin{array}{c} P\left[y \leq r_k \mid x_i\right] \geq P\left[y \leq r_k \mid x_j\right] \quad \forall r_k \in \mathcal{Y} \\ \text{or} \\ P\left[y \leq r_k \mid x_i\right] \leq P\left[y \leq r_k \mid x_j\right] \quad \forall r_k \in \mathcal{Y} \end{array}$$

If assume a utility function $U\left(x\right) = w^\top x + \epsilon$ ($y = r_i \iff U\left(x\right) \in \left[\theta\left(r_{i-1}\right), \theta\left(r_i\right)\right]$, $E\left[\epsilon\right] = 0, \epsilon \sim P_\epsilon$), then if apply a monotonic inverse link quartile function of ϵ, $l^{-1} : [0,1] \mapsto (-\infty, \infty)$ and $\theta : \mathcal{Y} \mapsto \mathbb{R}$ for increasing ranks, it can be proved that stochastic ordering is still preserved by:

$$l^{-1}\left(P\left[y \leq r_i \mid x\right]\right) = \theta\left(r_i\right) - \left(w^\top x\right) \implies$$

$$\begin{array}{rcl} P\left[y \leq r_k \mid x_i\right] & \geq & P\left[y \leq r_k \mid x_j\right] \\ l^{-1}\left(P\left[y \leq r_k \mid x_i\right]\right) & \geq & l^{-1}\left(P\left[y \leq r_k \mid x_j\right]\right) \end{array}$$

$$\begin{array}{rcl} l^{-1}\left(P\left[y \leq r_k \mid x_i\right]\right) - l^{-1}\left(P\left[y \leq r_k \mid x_j\right]\right) & \geq & 0 \\ w^\top\left(x_i - x_j\right) & \geq & 0 \end{array}$$

If we denote $x^{(1)}, x^{(2)} \in \mathcal{X}$ as the first and second object in a comparison, then we can denote a set (X', Y') such that $\forall i, 0 < \left|y_i^{(1)} - y_i^{(2)}\right|$:

$$(X', Y') = \left\{\left(\left(x_i^{(1)}, x_i^{(2)}\right), t_i\right)\right\}_{i=1}^m$$

Where $t_i = \Omega\left(y_i^{(1)}, y_i^{(2)}\right) := \text{sign}\left(y_i^{(1)} \ominus y_i^{(2)}\right)$

Generalizing the equation for both probability instances:

$$t_i w^\top\left(x_i^{(1)} - x_i^{(2)}\right) \geq 0 \tag{1}$$

implying $w^\top x_i^{(1)} > w^\top x_i^{(2)}$ when $x_i^{(1)} \succ x_i^{(2)}$ and vice versa.

5.2.1 Optimization Function[1]

According to the theorem of **A Margin Bound for Ordinal Regression** to find a function f such that p_f is maximal, we need to find a w such that the margin of $\left\{\left(x_i^{(1)} - x_i^{(2)}\right)\right\}_{i=1}^m$ is as large as possible. When we translate this into the principals of SVM, get we get the optimization function for hard margin:

$$\begin{array}{l} \min \tau\left(w\right) = \frac{1}{2}\left\|w\right\|^2 \\ \text{s.t. } t_i\left(w^\top\left(x_i^{(1)} - x_i^{(2)}\right)\right) \geq 1 \quad, \forall i = 1, \ldots, m \end{array}$$

However, this is assuming zero training error (there is no pairwise comparisons that are misclassified, and the scoring function is linear). If we allow some misclassification and that our score function is only approximately linear, then we have to use soft margin. It is highly recommended to use soft margin SVM in this case as these assumptions are must more robust.

5.3 SVOR Algorithm[1]

Using KKT conditions, we can find the dual of the optimization problem giving α such that

$$w = \sum_{i=1}^m \alpha_i \cdot t_i \cdot \left(x_i^{(1)} - x_i^{(2)}\right)$$

With α being the (soft margin) solution to

$$\boxed{\begin{array}{l} \max\limits_\alpha \left[1^\top \alpha - \frac{1}{2}\left(\alpha \otimes t\right)^\top K\left(\alpha \otimes t\right)\right] \\ \text{s.t. } \mathbf{0} \leq \alpha \leq \mathbf{1} \\ \qquad \alpha^\top t = 0 \end{array}} \tag{2}$$

$$t = \left(\Omega\left(y_1^{(1)}, y_1^{(2)}\right), \ldots, \Omega\left(y_{m'}^{(1)}, y_{m'}^{(2)}\right)\right)$$

$$K = \left(X^{(1)} - X^{(2)}\right)\left(X^{(1)} - X^{(2)}\right)^{\top}$$

Where $\otimes$ denotes element-wise multiplication.

After finding our w vector, the ranking of our objects $x_i \in X$ is performed by finding the score for each object x_i by $w^{\top} x_i, \forall i$ and sorting the scores.

5.3.1 Using Mercer Kernels[1]

In Large Margin Rank Boundaries for Ordinal Regression, the authors argue for the use of Mercer Kernels in transforming X into non-linear space. In such a case, our K matrix is populated such that:

$$K_{i,j} = \begin{array}{c} k\left(x_i^{(1)}, x_j^{(1)}\right) - k\left(x_i^{(1)}, x_j^{(2)}\right) + \\ -k\left(x_i^{(2)}, x_j^{(1)}\right) + k\left(x_i^{(2)}, x_j^{(2)}\right) \end{array}$$

where $k(x, x')$ is a Mercer Kernel.

Linear Kernel: This is for the simple case where the scoring function

$$k(x, x') = x^{\top} x'$$

Polynomial Kernel: Corresponds to mapping into the space spanned by all products of exactly d dimensions of $\mathbb{R}^N$

$$k(x, x') = \left(x^{\top} x'\right)^d$$

If we wish to take into account all products up to order d, then we can use:

$$k(x, x') = \left(x^{\top} x' + c\right)^d \quad c > 0$$

RBF Kernel: Mercer RBF kernels is:

$$k(x, x') = \exp\left\{-\frac{\|x - x'\|^2}{2\sigma^2}\right\}$$

Sigmoid Kernel: For suitable values of gain κ and threshold Θ, sigmoid kernels can be used:

$$k(x, x') = \tanh\left(\kappa\left(x^{\top} x'\right) + \Theta\right)$$

5.4 Summary

After running our implementation of Ranking SVM, we found a 0.7561 accuracy between the returned list and the pairwise preferences given, the best of the models tested. We tried this on both data sets: the original dataset given and the symmetric data we constructed. Both gave the same results: same vector w and the same returned order. The reason for this can be easily seen by noting that:

$$t_i\left(w^{\top}\left(x_i^{(1)} - x_i^{(2)}\right)\right) = (-t_i)\left(w^{\top}\left(x_i^{(2)} - x_i^{(1)}\right)\right)$$

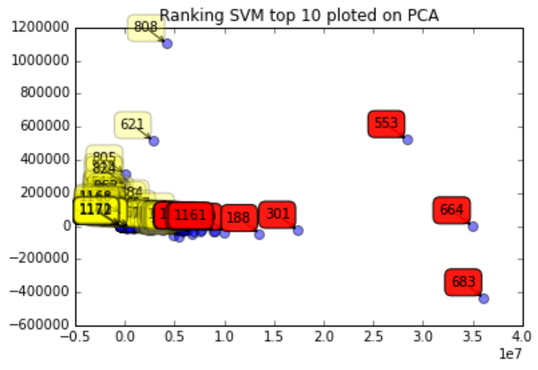

Figure 6: Top 10 Ranked Users According to SVOR

Thus by using our symmetric data, we were simply adding redundant constraints and not helping to achieve a better solution for SVOR.

If we plot the top 10 ranks from our SVOR model (outlined in red in Figure 6) on our first 2 principal components, we see that SVOR ranking follows the linear nature of the data fairly well, indicating that this is model for the data. To improve on the model feature engineering would be recommended.

As can be noted by Equation (2), SVOR is very similar to regular SVM with three important differences: points are the pairwise differences between users $\left\{\left(x_i^{(1)} - x_i^{(2)}\right)\right\}_{i=1}^{m}$, the primal has no intercept term, and $\alpha \in [0, 1]$.

6 Conclusion

Since the data is semi-supervised, to measure the accuracy of the model we compared the returned order with the pairwise preferences given. All models gave very similar results (between 0.72 and 0.75), however this metric is similar to the classification and does not measure how far off a rank is from its true position. This metric is even less reliable when we take into account some users are only ever compared once and others several times. This means that if a user that is rarely compared is very far from it's true rank, then this would only slightly affect the accuracy score while another ,more compared, user out of it's true position will greatly affect the score.

To get a rough judgement of the performance of the models overall, we investigated the attributes of the ordered users for each model and see if they followed a reasonable pattern (i.e. the users with largest follower count is near the top). Using this guideline, we found SVOR as the most reliable model of the three presented for this data.

	ID				
Rank	SVOR	Bootstrap		Cohen	
		NN	log	NN	log
1	683	197	246	1090	1090
2	664	236	455	1096	514
3	553	283	490	1146	904
4	301	421	585	1161	301
5	188	487	617	30	683
6	30	636	751	188	545
7	1161	721	938	301	30
8	1146	784	1073	553	607
9	1096	830	1112	664	553
10	1090	1114	1139	683	1

Table 1: Top 10 Most Influential Users for Each Model

References

1. Ralf Herbrich, Thore Graepel, and Klaus Oberma. 2000. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classiers

2. T. Kamishima, "A Survey and Empirical Comparison of Object Ranking Methods" `http://www.kamishima.net/archive/2009-b-pl2.pdf`

3. T. Kamishima, "Supervised Ordering — An Empirical Survey" `http://www.kamishima.net/archive/2005-p-icdm-unofficial.pdf`

4. T. Kamishima, "Learning from Order Examples" `http://pdf.aminer.org/000/302/633/learning_from_order_examples.pdf`

5. Johannes Fürnkranz, Eyke Hüllermeier, "Preference Learning". Springer

6. Greedy Algorithm Proofs:
`https://courses.cs.washington.edu/courses/cse421/08au/Greedy.pdf`
`http://www.idi.ntnu.no/~mlh/algkon/greedy.pdf`