

Efficiently Learning Trends in User Preferences

Max Crane
17922192
maxc@berkeley.edu

December 12, 2006

1 Introduction

Many websites encounter the problem of recommending content to users based on what they have shown interest in in the past. They must find correlation in what the user in question and other users have purchased or looked at to make these recommendations. This is one common example of a collaborative filtering problem. The goal of this project is to develop a very efficient method for solving this sort of problem.

Researchers have applied many different techniques to the collaborative filtering problem and have interpreted the problem in different ways. It can be interpreted as a classification problem, a clustering problem, a regression problem, or a searching problem. It is a common problem for online stores, such as amazon.com, but it has also been solved for other purposes, such as in the categorization of Usenet news. The problem is not limited to user systems; it deals with general prediction of missing data scenarios. [7][5][4]

2 Problem Formulation

There is a system with M users and N items. Each user gives a response to a random collection of R of the N items. This response is a real number. The response is greater than zero if the user likes the item and negative if the user dislikes the item. Typically, M will be much greater than N and R will be much less than N . The algorithm must classify for each user whether each item would be liked or disliked.

The random collection of responses per user is assumed to be determined completely at random. That is, data is missing completely at random (MCAR) as opposed to missing at random (MAR) or worse. This may not be a valid assumption for many potential applications. For example, in a web-based store, a user is very unlikely to buy or even look at an item in a category that he or she is not interested in. So, the missingness of these items would be related to the user's preference for them. This algorithm makes no attempt to include the missingness of item responses in the predictions for the user preference toward them. However, this could be accomplished in some sort of post-processing step.

Another assumption made about the data is that every item has a suitable collection of responses. Consider the graph with a vertex for each item and an edge connecting a pair of items when and only when some user has provided responses for both items. For good results to be found, this graph must be made of only one connected component. For example, if there is a user who supplies responses to some items that no other users supply responses to, there is clearly no way to predict how that user would respond to other items or how other users would respond to the items this first user responded to. Beyond the requirement that the entire graph is connected, it is also important that the graph is fairly well connected. Consider the case where two well connected portions of the graph are connected to each other by only one edge (one user). If this single user changed his response to one of the items that resulted in this connecting edge, a large portion of the solution would be changed because that item is the only piece of information correlating the two portions of the solution. Here the problem is that the system is very chaotic, meaning a small change in the initial input has a great effect on the final solution. If that user is atypical, then large parts of solution will be incorrect.

It is not a strict requirement that each user provides exactly R responses. The algorithm will work with a variable number of responses per user. We will suppose instead that each user supplies an average of R responses, and that each user supplies at least one response.

The variance of the response data for each item is of some significance. For most applications, this variance can be expected to be relatively uniform. If the variance in responses for one item is significantly different from that of another item, an extra normalization step is required at some point in the algorithm. This can be done at the beginning, but if the number of users is much greater than the number of items, it is slightly more efficient to do

this at a later stage. There is also another restriction related to this and the connectedness of the data. The variance of the subset of responses for an item i all connected to an item j must be approximately equal to the variance of the complete set of responses to item i . The reasons for this restriction become apparent in the formulation of the algorithm.

3 Algorithm

Each item is modeled in terms of the other items in the system using a perceptron classifier.

$$C(x_{i,j}) = \begin{cases} \text{liked} & \mathbf{w}_j \cdot \mathbf{x}_i > b \\ \text{disliked} & \mathbf{w}_j \cdot \mathbf{x}_i \leq b \end{cases}$$

\mathbf{x}_i is the vector of response values for user i . $x_{i,j}$ is the unknown j th response value for user i that is being classified. \mathbf{w}_j is a parameter learned for item j . The elements of \mathbf{w}_j are the correlation values between item j and each other item. Note that \mathbf{w}_j and \mathbf{x}_i both contain values for the j th item. This value in \mathbf{x}_i will be set to zero so the computation isn't affected.

3.1 Computing \mathbf{w}_j

Computation of \mathbf{w}_j is the main learning component of this algorithm. This involves taking the weighted sum of the item preferences of each user. The basic intuition for the weights is that we want to give greater sway to those preferences of users that gave strong negative or positive responses to item j . This leads us to the following sum.

$$\mathbf{w}_j = \sum_{k=1}^M x_{k,j} \mathbf{x}_k$$

When this vector is dotted with \mathbf{x}_i in the classification of $x_{i,j}$, items with strong positive correlation in \mathbf{w}_j and with strong positive responses in \mathbf{x}_i will push the result in the positive direction. Similarly, items with strong negative correlation and negative responses in \mathbf{x}_i will push the result in the positive direction. Items with opposite correlation and response in \mathbf{x}_i will push the result of the dot product in the negative direction.

These correlation values computed for \mathbf{w}_j are not normalized, but they do not need to be for the classification to work properly in the case where the

different items in the system have uniform response variance. In the other case, the entire vector must be scaled by the inverse standard deviation of the response values for item j , and each element n of the vector must be multiplied by the inverse standard deviation of a subset of the response values for item n . This subset is those response values that were actually used in the computation of this n th element. These are the values for which the user has supplied a response value for both item j and item n . If these extra steps are taken, the \mathbf{w}_j values will be properly normalized and are the actual Pearson product-moment correlation values for the item with each other item.

3.2 Handling Missing Values

The equations above use vectors \mathbf{x}_i , the response values for each item from each user i . Not all of these values are defined because any given user i has only provided responses to, on average, R of the N items. We deal with this by *imputating* values of zero for missing responses, meaning we leave the missing response values in the computations, but define them as zero.

In these computations, the zero values essentially have no effect on the results. For \mathbf{w}_j , clearly each n th value is only affected by those \mathbf{x}_k vectors with nonzero values for both $\mathbf{x}_{k,j}$ and $\mathbf{x}_{k,n}$. So, a user who doesn't provide responses for both item j and item n will not affect the n th correlation value in \mathbf{w}_j . During classification, since the operation being performed is a dot product, zero values in \mathbf{x}_i will have no effect on the final result.

3.3 Formulation as Matrix Multiplication

These computations can be formulated as matrix multiplications. We will define the $M \times N$ matrix \mathbf{X} as the matrix with rows of response values for each user. As described above, missing values will be filled in with zeros.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_M \end{bmatrix}$$

This matrix is the input to the algorithm.

Reconsider the sum used to calculate \mathbf{w}_j .

$$\mathbf{w}_j = \sum_{k=1}^M x_{k,j} \mathbf{x}_k$$

This is equivalent to multiplying the transpose of X with the j th column of X.

$$\mathbf{w}_j^T = \mathbf{X}^T \mathbf{X}_{*,j}$$

So, the \mathbf{w} vectors can all be calculated at once by multiplying X with its transpose.

$$\mathbf{W} = \mathbf{X}^T \mathbf{X}$$

The \mathbf{w} vectors are the columns of this $N \times N$ matrix W. We notice that the matrix is symmetric due to the way it is computed. This knowledge can be used to accelerate the computations. We also observe that W is similar in structure to the item to item correlation matrix for the system. $W_{i,j}$ is a large positive value when the i th and the j th items are strongly correlated and a large negative value when they are strongly anticorrelated. The values are not normalized with respect to one another as they would be in a typical correlation calculation. This could be done with some additional computations, but if we can expect the data to have fairly even variance, this is unnecessary.

In the case that the variance in the response values was not normalized to begin with, we now need to modify the values of W. We first calculate a N -vector \mathbf{s} with elements equal to the inverse standard deviations of the supplied responses of each item. Each row and column of W is then multiplied element-wise with \mathbf{s} .

$$W'_{i,j} = W_{i,j} \mathbf{s}_i \mathbf{s}_j$$

This is subtly different from the normalization process described in the section on calculating \mathbf{w}_j . In that process, the variance of only a subset of the response values are used in some of the adjustments. That is the proper way to do the normalization, but would take much more time to compute than this method. The difference between the two should be negligible for most applications, however. This is the reason for the restriction mentioned in the formulation of the problem.

Classification can be simplified as well. The dot product between each \mathbf{x} vector and each \mathbf{w} vector is computed by multiplying X with W.

$$\mathbf{Q} = \mathbf{XW}$$

This $M \times N$ matrix Q has a similar arrangement to X and has each of the response values filled in with the dot products used for classification. All that remains is to compare these values to the bias parameter to get the classification for each user and each item.

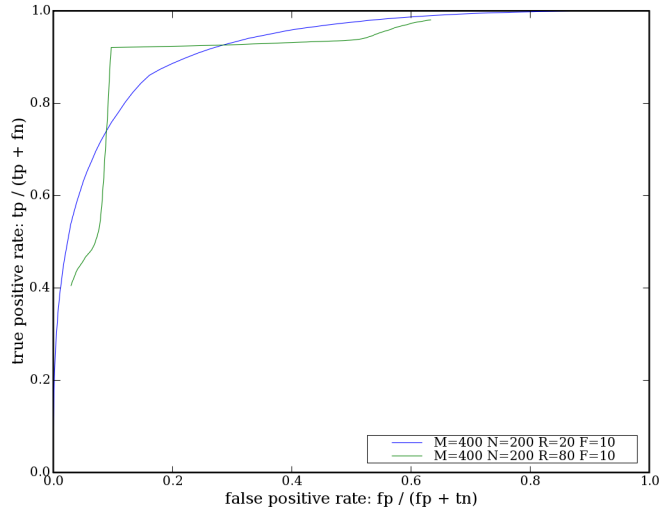
4 Efficiency

The algorithm can essentially be simplified to two matrix multiplications and the normalization step, which may or may not be necessary. This is a useful result because the algorithm can be very easily implemented using various math libraries. A simple implementation like this will instantly be able to leverage research being done to optimize matrix multiplication. An important consideration is that the matrices being operated on are fairly sparse. Depending on the application, there may be major performance benefits to be gained by taking advantage of this sparsity. The reader is referred to the literature on fast matrix multiplication techniques.[2][6]

5 Accuracy

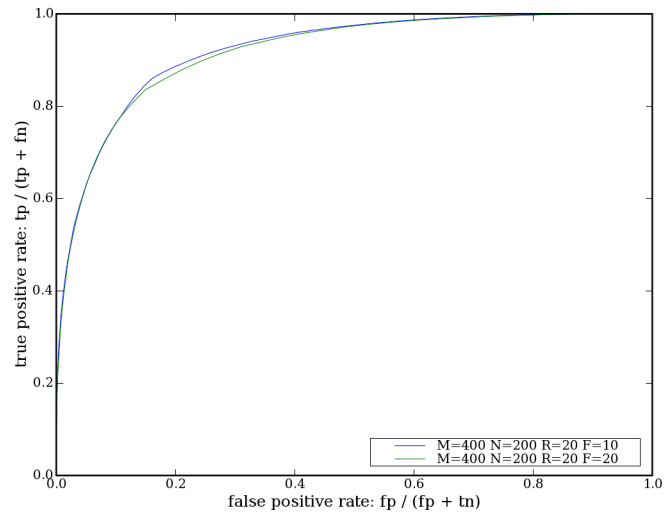
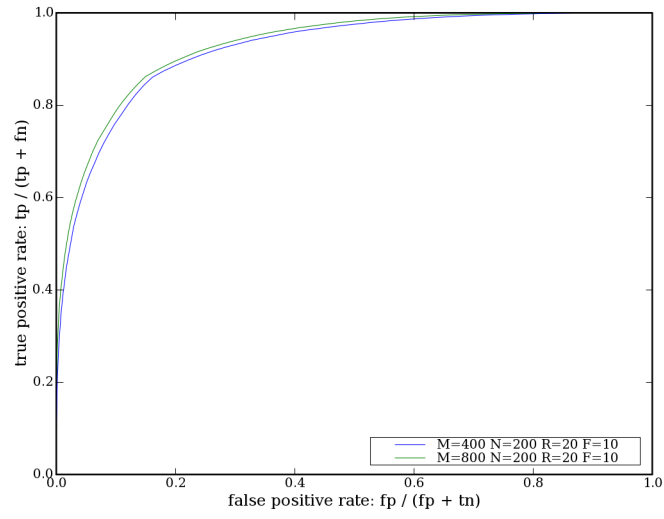
While accuracy is not the main focus of this project, it is still an important part of any learning system. Due to a lack of suitable real data, I have attempted to evaluate the accuracy of the algorithm on synthetic data. The danger of this approach is that the data the algorithm may encounter in the real-world is likely to be significantly different from the data generated for testing purposes. Since I developed this method with no specific application in mind, whatever data is used to test this would be inadequate for evaluation with respect to some other use. Researchers have used a variety of synthetic data generation techniques to test collaborative filtering algorithms, but the datasets used to test some algorithms are inappropriate for evaluation of others. [1][3]

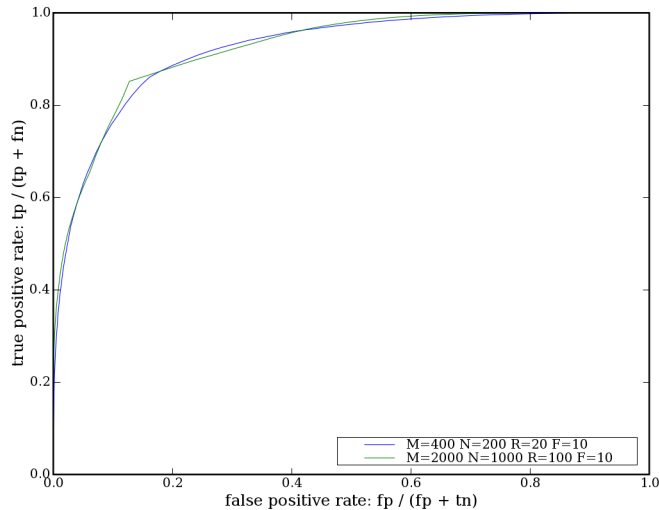
The data generated to evaluate this algorithm is based on simple features. Some number of features F is chosen. Each item is assigned random values in range $[0, 1]$ for each feature. Users are then assigned random preference values in range $[-1, 1]$ for each feature. A user's preference for a given item is then determined by dotting the user's feature preference vector with the item's feature vector. This method is intended to generate a dataset with



patterns that the learning algorithm should be able to deal with.

Test results are shown here as ROC plots with a varying bias classification parameter. Results on two different datasets are shown in each plot. In all but the last case, only one parameter was changed in the generation of the two different datasets. The ROC plots show several things about the algorithm. The first plot shows an interesting result about the effect of response density on the classification. The pointy curve yielded in the run against data with many supplied responses per user shows that the classification is very accurate with a neutral bias parameter, and adjusting it will not result in more or fewer true positive classifications. The concave sections are hard to explain, however. In the second plot, the number of users is varied with respect to the number of items. The algorithm is slightly more successful with more users to learn trends from. The third plot has a varied feature count parameter, which is only used for generation of the data. The hope is that varying the means of data generation do not wildly change the classification results. The two curves are acceptably similar. The final plot shows a larger dataset. The curve for this one is also slightly concave as in the first example. The classifier does seem to achieve better accuracy with bias values close to zero for the larger dataset.





6 Incremental Updating

Thus far, the assumption has been that the application requires the entire solution to be solved at once. However, most recommendation systems actually have users entering the system and providing responses to items very gradually. The formulation of the solution as a matrix product makes it simple to express methods to incrementally update the system as new users and item responses enter.

One obvious improvement when data is gradually entering the system is to store the computed W matrix. When a user or item response is added, the W matrix is immediately made invalid. However, it will not be far from correct, assuming the system is not chaotic as discussed in the problem formulation. A website making recommendations could simply use the same W for the day, then recompute everything at night when there is extra processing time available due to the decreased number of users. This is one good method, and very simple to implement.

A useful property of the matrices the algorithm works with is that a single row, column, or element of the overall product can be computed in proportionally smaller time. Also observe that updating the solution matrix a row or even a single element at a time never leaves the solution in an

invalid state. This is because none of the previously learned data is used by the algorithm; it is limited to the response values supplied by the users. A website could therefore update needed values just-in-time as people are using the site. This solves the problem of time being wasted computing recommendations for old users that no longer visit the website.

7 Conclusion

I developed an algorithm to solve a general collaborative filtering problem. The main focus of this project was producing a very efficient process. I feel that in this aspect, the algorithm is a success. Its accuracy is not especially impressive, but this is to be expected for an algorithm that strives for good efficiency. Future work will involve an actual real-world application of this algorithm on real data.

References

- [1] Charu C. Aggarwal, Joel L. Wolf, Kun-Lung Wu, and Philip S. Yu. Horting hatches an egg: a new graph-theoretic approach to collaborative filtering. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 201–212, New York, NY, USA, 1999. ACM Press.
- [2] Fred G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw.*, 4(3):250–269, 1978.
- [3] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.
- [4] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.
- [5] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

- [6] Victor Pan. *How to multiply matrices faster*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.
- [7] L. Ungar and D. Foster. Clustering methods for collaborative filtering, 1998.