The Love-Hate Square Counting Method for Recommender Systems

Joseph S. Kong joseph.kong@ngc.com
Kyle Teague kyteague@gmail.com
Justin Kessler
Northrop Grumman, 7575 Colshire Dr, McLean, VA 22102

Editor: G. Dror, Y. Koren and M. Weimer

Abstract

Recommender systems provide personalized suggestions to users and are critical to the success of many e-commerce sites, such as Netflix and Amazon. Outside of e-commerce, recommender systems can be deployed in fields such as intelligence analysis, for recommending high-quality information source to analysts for further examination. In this work, we present the *square counting method* for rating predictions in recommender systems. Our method is based on analyzing the bipartite rating network with score-labeled edges representing user nodes' ratings to item nodes. Edges are denoted as an *I-love-it* or *I-hate-it* edge based on whether the rating score on the edge is above or below a threshold. For a target user-item pair, we count the number for each configuration of love-hate squares that involve the target pair, where the sequence of *I-love-it* or *I-hate-it* edges determine the particular configuration. The counts are used as features in a supervised machine learning framework for training and rating prediction. The method is implemented and empirically evaluated on a large-scale Yahoo! music user-item rating dataset. Results show that the square counting method is fast, simple to parallelize, scalable to massive datasets and makes highly accurate predictions. Finally, we report an interesting empirical finding that configurations with consecutive *I-hate-it* edges seem to provide the most powerful signal in predicting a user's love for an item. [1]

Keywords: collaborative filtering, recommender systems, square counting method, complex network, rating network

1. Introduction

Recommender systems aim to provide a user with high-quality personalized recommendations by analyzing data on all users' preference for items. Recommender systems have greatly gained popularity over the years and are now deployed across the e-commerce domains, in websites such as Netflix, Amazon and Youtube. Besides suggesting items to purchase or movies/videos to view in e-commerce type of applications, recommender systems have the potential to be productive in other fields, such as in recommending information sources for further examination (e.g. documents, images, etc) in intelligence analysis (Bartee and Gertner, 2006).

1. Approved for public release by Northrop Grumman Information Systems, ISHQ-2011-0042. The work was entirely performed when Kyle Teague was at Northrop Grumman.

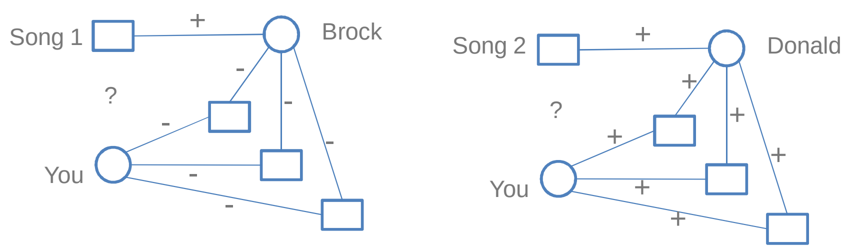


Figure 1: Hanging out with Brock and Donald ('+' denotes an *I-love-it* edge, '-' denotes an *I-hate-it* edge). You and Brock hate the same 3 songs, while you and Donald love the same 3 songs. Are you more likely to prefer Song 1 or Song 2?

Existing popular methods for recommender systems are based on collaborative filtering, and have the interesting and attractive property of being domain-independent: no expert knowledge is required for building a recommender system based on collaborative filtering. For example, a method that works well for recommending movies to watch is expected to work reasonably well in suggesting books to buy.

Imagine for a moment that you are hanging out with two friends called Brock and Donald at a coffee shop (see Fig. 1 for an illustration): after chatting with Brock for a few minutes, you discover that there are three songs that both you and Brock hate; after a few more exchanges with Donald, you find that there are three songs that both you and Donald love listening to. Now, Brock likes Song 1 and Donald likes Song 2. With the given information, is it more likely that you will love Song 1 or Song 2? Intuition may suggest that you are more likely to love Song 2, since you and Donald seem to have very similar tastes in what music to enjoy, but one is far from being sure. In fact, empirical data may suggest otherwise.

In this work, we hypothesize that such rating network neighborhood information is very useful in determining a person's preference, and we will empirically investigate whether this hypothesis is true. To this end, we have developed the *square counting method* for recommender systems: first, we construct a bipartite rating network with two types of nodes (i.e. the users and the items); the edges carrying scores represent the users' ratings to items; all edges are categorized as an *I-love-it* or an *I-hate-it* edge based on whether the rating score for the edge is above or below a threshold; for a target user-item $u - i$ pair, all squares involving the given user-item pair (i.e. paths that start from user u and end at the item i) are identified; the counts for each particular love-hate square configuration are tabulated; the counts for each love-hate configuration are then used as features in a machine learning framework for training and prediction. Intuitively, the presence of a large number of certain love-hate square configuration on the rating network may indicate the target user-item pair as being an *I-love-it* or *I-hate-it* edge.

Many interesting questions arise here: for example, which love-hate square configuration is most powerful in predicting a user's like or dislike for an item? We have implemented the square counting method and applied it on the Yahoo! music user-item rating dataset as part of participating in the data mining competition KDD Cup 2011. Very encouraging results have been obtained.

2. Related Works

A popular approach to building recommender systems is to employ neighborhood models as in (Herlocker et al., 1999; Linden et al., 2003). Such an approach predicts the rating of a given item by using the ratings of similar users or alternatively, by using the ratings of similar items. The Pearson correlation coefficient is commonly used to measure the similarity between items or between users, although other distance measurements can be used as well. For example, borrowing from the notation in (Koren, 2008), to predict the rating $\hat{r}_{ui}$ of item i for user u we let $S^k(u;i)$ be the set of the k most similar items to item i rated by user u. A predicted rating could then be calculated in the following way:

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k(u;i)} s_{ij} r_{uj}}{\sum_{j \in S^k(u;i)} s_{ij}}$$

where s_{ij} is the similarity between item i and item j.

Neighborhood models of this type enjoy widespread popularity because they are intuitive and relatively easy to implement. However, as show in (Koren, 2008) they perform poorly in terms of reducing the RMSE (root-mean-squared-error) compared to more complex methods. Neighborhood models with learned parameters perform considerably better (Bell and Koren, 2007; Koren, 2008).

Another collaborative filtering approach, which was popularized by the Netflix competition in 2006, is the latent factors model. The basic form described by Koren attempts to parameterize each item i by a vector of factors $q_i \in \mathbb{R}^f$ and each user u by a vector $p_u \in \mathbb{R}^f$ representing the user's affinity for those factors. The additional parameters b_u and b_i are used to represent the bias for each user and item, respectively. A predicted rating is then given by $\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i$. Parameter estimation can be done by solving the regularized least squares problem with stochastic gradient descent or alternating least squares:

$$\min_{p*,q*,b*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2)$$

Newer techniques such as Factorization Machines (Rendle, 2010) attempt to provide a more general factorization algorithm, applicable to a wider range of problems. Other techniques that have produced relatively low RMSE scores include Restricted Boltzmann Machines (Salakhutdinov et al., 2007) and Kernel Ridge Regression (KRR) (Paterek, 2007). Further gain in accuracy can be achieved by combining multiple models using an ensemble method known as stacking, also commonly referred to as blending. Using this technique the predictions of several models on a held-out test set are provided as training examples to a final regression algorithm. Algorithms used for stacking have included Additive Regression with Stochastic Gradient Boosting, Neural Networks, KRR, and Binned Linear Regression (Jahrer et al., 2010).

Researchers have developed network-based techniques for collaborative filtering in recommender systems. A method based on resource allocation dynamics over networks has been proposed (Zhou et al., 2007). A method based on a heat spreading algorithm for resolving the diversity-accuracy dilemma of recommender systems has been developed (Zhou et al., 2010). A related line of research involves the study of signed social networks, which

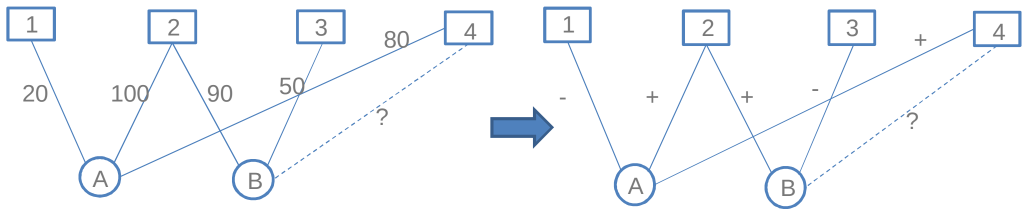


Figure 2: Lefthand figure: an example of a bipartite rating network extracted from ratings data. Here, user A rates items 1, 2 and 4 with scores 20, 100 and 80, respectively, while user B rates items 2 and 3 with scores 90 and 50. Righthand figure: an edge with a rating score ≥ 80 is denoted as *I-love-it* (by the '+' sign); an edge with a rating score < 80 is denoted as *I-hate-it* (by the '-' sign).

are found on websites such as Epinions and Slashdot. A method for predicting the sign of a link in online social networks by applying machine learning on the signed degrees of nodes and the triads in the immediate social network neighborhood has been proposed (Leskovec et al., 2010).

3. The Square Counting Method for Recommender Systems

3.1. Bipartite Rating Network

Rating information collected by recommender systems can be fully modeled as a bipartite rating network, where two types of nodes, users and items, are linked by edges with a rating score. The lefthand figure in Fig. 2 shows an example bipartite rating network extracted from ratings data: the rating scores given by users A and B to items 1, 2, 3 and 4 are shown. A typical task in a recommender system may be to predict what score user B would like to give to item 4 or to predict whether user B is likely to give a high rating to item 4.

3.2. The Love-Hate Squares

Formally, we set an arbitrary cutoff score r_c for all edges in the rating network: an edge with a rating score $r \geq r_c$ is considered an *I-love-it* edge, while an edge with a score $r < r_c$ is considered an *I-hate-it* edge (see the righthand figure in Fig. 2; '+' denotes an *I-love-it* edge, while '-' denotes an *I-hate-it* edge).

For a target user-item $u_{tg} - i_{tg}$ pair on the rating network to make prediction on, we examine a path of 3 hops that start from u_{tg} and end at i_{tg}, thus completing a cycle and forming a square with the target pair $u_{tg} - i_{tg}$. Fig. 3 enumerates all 8 possible *love-hate* square configurations for a target user-item pair. In this work, we will use the terms *target user* and *target item* to denote the user and item that make up the target pair in a square, labeled u_{tg} and i_{tg} in Fig. 3, respectively. We will use the terms *other user* and *other item* to denote the other two nodes in a square, labeled u_{ot} and i_{ot} in Fig. 3, respectively.

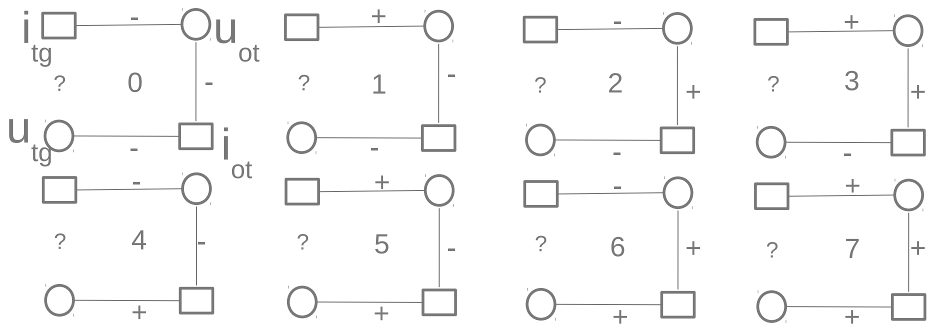


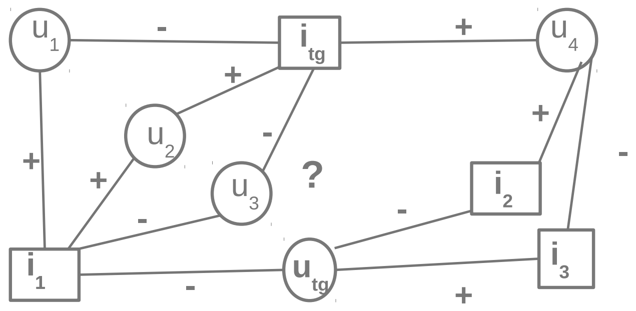
Figure 3: All possible love-hate squares: for a target user-item pair (labeled u_{tg} and i_{tg}, respectively) to be predicted, all 8 possible love-hate square configurations are displayed. The configuration number for each square configuration is shown in the middle: for example, configuration No. 5 describes the scenario where the target user u_{tg} loves an item i_{ot}, for which another user u_{ot} hates; but, user u_{ot} loves the target item i_{tg}.

3.3. Square Counting

Here we describe the algorithm for tabulating the different square configurations for a target user-item pair: for the target item, we assume a hash table is available, where the keys consist of the set of users that have rated the target item with the rating score as the value. Given a target user, we set the target user as the root node, from which we perform a breadth-first search on the rating network with a finite depth of 2 hops; For each of the user nodes discovered after 2 hops, we perform a lookup on the hash table to check if the user node is one of the users that has rated the target item; if yes, a 3-edge path has been found, and we determine the configuration number based on the signs on the 3 edges (i.e. either *I-love-it* or *I-hate-it*). We increment the count for the corresponding configuration number (see Fig. 4 for an illustration).

For example, in Fig. 4, the path $u_{tg} - i_1 - u_1 - i_{tg}$, which has a sign sequence of $\{-, +, -\}$, corresponds to configuration No. 2 (see Fig. 3); thus, the count for configuration No. 2 is 1. Note that the sign sequence discussed here will follow the convention of tracing the edges from the target user u_{tg} to the target item i_{tg}. Another example, the paths $u_{tg} - i_1 - u_2 - i_{tg}$ and $u_{tg} - i_2 - u_4 - i_{tg}$, which both have a sign sequence of $\{-, +, +\}$, correspond to configuration No. 3; thus, the count for configuration No. 3 is 2. The counts for each instance of target user-item pair will be written as a row in an instance-feature matrix.

For each target user-item pair, the time complexity for performing square counting is simply the time complexity for a finite breadth-first search (BFS) with a depth of 2 hops from the target user. Let j and k denote the user node degree (i.e. number of items a user has rated) and item node degree (i.e. number of users that have rated an item), respectively. The breadth-first search is performed on a bipartite rating network with heterogeneous degree distributions $P(j)$ and $Q(k)$, which denote the degree distributions of user and item



Config No.	0	1	2	3	4	5	6	7
Count	1	0	1	2	0	1	0	0

Figure 4: Counting squares: in this illustrated sample scenario, given a target user-item pair u_{tg}-i_{tg} to make prediction on, all 3-hop paths that start from user u_{tg} and end at the item i_{tg} are enumerated; for each 3-hop path, the corresponding square configuration number can be determined based on the signs of the 3 edges (see Fig. 3); we simply keep the count for each configuration number; note that the counts for each configuration number will be later used as features in the machine learning framework as described in Sec. 3.4.

nodes, respectively. Starting BFS from a user, the expected degree of a user node is $\langle j \rangle$. Thus, $\langle j \rangle$ items are discovered after 1 hop of BFS on average. Now, the expected degree of an item node reached via a hop from a user node is given by $\frac{\sum_k kP(k)(k-1)}{\langle k \rangle} = \frac{\langle k(k-1) \rangle}{\langle k \rangle}$. The explanation is as follows: an item with degree k has k ways of getting discovered and thus the probability an item node getting discovered is proportional to $kP(k)$ (i.e. the probability is given by $\frac{kP(k)}{\langle k \rangle}$ where $\langle k \rangle$ is the normalizing constant); the discovered item node has a remaining degree of $(k-1)$ not counting the edge on which the item node is reached; thus, we arrive at our result by summing over all possible values of k. See (Newman et al., 2001) for a detailed explanation and derivation. On average, the number of users discovered after 2 hops of BFS is given by $\frac{\langle j \rangle \langle k(k-1) \rangle}{\langle k \rangle}$. Thus, the average time complexity for 2-hop BFS is $(1+\langle j \rangle+\frac{\langle j \rangle \langle k(k-1) \rangle}{\langle k \rangle})$, which is $O(\langle k^2 \rangle)$ since the second moment typically dominates the first moment in degree distributions of real-world networks. Assuming only a constant number of target items need to be trained with each target user, the average time complexity is thus $O(N\langle k^2 \rangle)$ for generating the instance-feature matrix as the number of system users N scales up. Since the degree distributions are not expected to change as the system size scales up, $\langle k^2 \rangle$ becomes a constant, which results in a time complexity of $O(N)$. Furthermore, it is simple to parallelize the task of populating the instance-feature matrix by performing square counts on multiple target user-item pairs in parallel.

3.4. Machine Learning Framework

The counts for different configurations will be used as *features* in our machine learning approach. We assume that rating data for training and a set of user-item pairs with known rating scores for validation have been made available. The machine learning framework consists of the following four steps:

1. First, we perform square counting, as described in Sec. 3.3, on the training network for each user-item pair in the validation set. A *validation* instance-feature matrix is generated, where each user-item pair in the validation set represents an instance, and the counts for the different configurations are the features. The truth label for each instance can either be a 0/1 binary value if the task is to make some binary prediction, or can be the rating score if the task is to predict the score.

2. Second, depending on whether the task is to predict a binary value or a continuous score, a classifier or a regression model is trained on the *validation* instance-feature matrix.

3. Third, repeat square counting (see Sec. 3.3) on the (training + validation) network for each user-item pair in the test set, and generate a *test* instance-feature matrix.

4. Fourth, apply the trained classifier or trained regression model from Step 2 to make a prediction for each instance in the *test* instance-feature matrix.

3.5. Possible Enhancements

There exist numerous ways to perform square counting in order to improve the accuracy in making predictions. Here we describe some possible methods for making enhancements, which provide additional features for the instance-feature matrix in the machine learning framework.

3.5.1. Normalizing Counts against the Random Network Model

Intuitively, for a target pair consisting of a prolific user and a popular item, the number of all possible squares in the rating network that involve the target user-item edge may be high simply because the user and the item are high-degree nodes in the rating network. In fact, it is straightforward to show that in a bipartite random network with identical degree sequence as the given bipartite rating network (Newman et al., 2001), the expected number of squares that involve a target user-item edge with user degree j_u and item degree k_i, respectively, is proportional to their degree product $j_u k_i$. Thus, we can normalize against the random network model by dividing the counts for the different square configurations by the degree product $j_u k_i$. Now, for example, a high normalized count represents a true presence of high number square configurations that involve the target user-item pair, but not an artifact from the degree of the user or the item.

3.5.2. Separate Counts Based on the Item Hierarchy

In recommender systems, an item hierarchy that describes the relationship among the items may be provided. For example, two songs may belong to the same artist or two movies may belong to the same genre. Each square contains two items. When enumerating the all possible squares for a target user-item pair, we can *separately* tabulate the square configuration counts based on whether the two items in the square belong to the same artist, for example.

3.5.3. Further Edge Categorization

In Sec. 3.3, each edge falls into one of two categories of *I-love-it* or *I-hate-it* based on whether the rating score is above or below a cutoff score. A natural extension is to have two cutoff scores such that each edge can fall into one of three categories, and so forth. For an edge categorization scheme of separating each edge into c categories, the number of possible square configurations (i.e. the number of features) grows as c^3.

3.5.4. Removing Very Popular Items

Extremely popular items that are rated by a significant fraction of the users in a recommender system may provide very little or no signal regarding a user's preference. We can preprocess the rating network by first removing the items with the highest number of ratings. A side benefit from this technique is that the run time for square counting may be significantly reduced.

3.5.5. Using Bias-Removed Scores

In recommender systems, it is well known that inherent user and item biases exist. Stochastic gradient descent based techniques for estimating user and item biases are well established and are straightforward to implement (see Eq. (3) in (Koren et al., 2009)). Thus, we can estimate the bias for every rating in the dataset and simply subtract off the bias from each rating to obtain the bias-removed scores. Now, in addition to using the raw rating scores for edge categorization in Sec. 3.3 and 3.5.3, we can use the bias-removed scores.

4. Results on Yahoo! Music Dataset

4.1. Dataset Description and Some Statistics

The KDD Cup 2011 competition provides a large-scale user-music-item rating dataset from Yahoo! music. See (Dror et al., 2012) for a detail description of the dataset. Although the square counting method can be applied on both tracks of KDD Cup 2011, we focused on track 2 of the competition, where the task is to distinguish a track that has been highly rated (defined as a rating score greater than 80) by a user from a track that has been sampled with a probability proportional to the number of high ratings for the track. The rating network extracted from the track 2 dataset has 249,012 user nodes, 296,111 item nodes, 61,944,406 edges for training, and 607,032 for test.

4.2. Generating the Validation Set

Since no validation set is provided for track 2, the first challenge is to generate a validation set that mimics the test set. For each user that appears in the test set and has 6 or more high ratings, first, we randomly select 3 tracks from all tracks in the dataset with a probability proportional to the number of high ratings received by the track, and insert these 3 tracks

Table 1: Summary of Results (Test1 Set)

Enhancement Methods Used	**ErrRate**
simple and deg product normalized counts	6.9969
4-category edges	6.1215
popularity normalized counts	5.6284
use GBM for blending	5.4072
hi-deg-items-removed .1%	4.9484
hi-deg-items-removed .5%,bias-remov scores	4.6336

into the validation set with a label of “0”; second, we randomly select 3 highly rated tracks from the list of highly rated tracks by the user, and insert them into the validation set with a label of “1” (note that these 3 ratings are deleted from the training network). Note that the validation instance-feature matrix used to train the machine learned classifier will be balanced, since there is an equal number of “0” and “1” instances.

4.3. Results

4.3.1. Baseline

We perform square counting (see Sec. 3.3) on the rating network constructed from the track 2 dataset, where the cutoff score r_c is set to $r_c = 80$ (see Sec. 3.2). We have applied the enhancement method detailed in Sec. 3.5.1, which normalizes square counts by the degree product of the target user-item edge. We have also separated counts into categories based on the item hierarchy as described in Sec. 3.5.2: we tabulate separate counts for *shared-album*, *shared-artist*, *shared-genre*, and *shared-nothing* categories for the cases where the target track and the other item in the square belong to the same album, same artist, same genre and share nothing, respectively. Since each track can have multiple genres, the category *shared-genre-Jaccard* is introduced, where the count for each square configuration is weighted by the Jaccard Coefficient of the target item’s genre set and the other item’s genre set.

A validation and a test instance-feature matrix are generated after tabulating the square counts (features) for the target user-item pairs (i.e. instances) in the validation set and test set. In applying the machine learning framework, a logistic regression classifier is trained on the validation instance-feature matrix, and the trained classifier is used to make prediction on the test instance-feature matrix. The probability prediction made by the logistic regression classifier for each instance is interpreted as the probability that the track has been highly rated by the user. For each user in the test set, 6 probabilities are generated by the classifier. For the purpose of submitting the prediction file to the KDD Cup website, the 3 items with the highest probabilities are labeled as “1” with the rest labeled as “0” . The error rate on the Test1 Set is 6.9969%.

Our C++/OpenMP implementation of the square counting method as described in Sec. 3.3 takes about 5 hours to run on a Desktop with 8 computing cores. Training the logistic regression classifier on the validation instance-feature matrix and using the classifier to make prediction on the test instance-feature matrix take minutes to complete with the GLM package in the statistical package R.

Table 2: GBM Parameters

GBM Parameter	**Value Used**
distribution	bernoulli
n.trees	2000
shrinkage	0.01
interaction.depth	1
bag.fraction	0.5
train.fraction	1.0
n.minobsinnode	1
cv.folds	0
method	OOB

4.3.2. Various Enhancements

In this section, when we incorporate an additional enhancement method, we always blend the existing results that have been obtained so far with the new predictions via ensemble learning (Jahrer et al., 2010). Results for the various enhancement methods are summarized in Table 1.

First, we try categorizing each edge into 4 labels as discussed in Sec. 3.5.3, where the 3 cutoff scores used are 30, 50, and 90, respectively. We repeat the procedure described above on the 4-label edges and lowered the error rate further to 6.1215%.

Here we describe an enhancement method that may be specific for track 2 of the KDD Cup 2011 competition. For track 2, recall that a track is randomly selected with a probability proportional to the number of high ratings it has received. Similarly, we can define the popularity of an album (or an artist) by summing the number of high ratings that the tracks within the album (or by the artist) have received. An album with high popularity is proportionally more likely to be randomly selected; hence, if a square count falls under the *shared-album* or *shared-artist* category, we normalize each square count by the inverse of the popularity of the album or artist, respectively. This enhancement method has further pushed our error rate down to 5.6284%.

At this point, we experiment with boosting methods by applying the GBM package in the statistical package R (see Table 2 for parameters used) for the classification task and for blending existing results via ensemble learning. The error rate is further reduced slightly to 5.4072% as a result.

Next, we remove the top 0.1% of items with the highest degree (i.e. highest number of ratings received) from the rating network (as described in Sec. 3.5.4), which further pushes the error rate down to 4.9484%. Finally, we experiment with removing the top 0.5% of items with the highest degree from the rating network and also try using bias-removed scores (see Sec. 3.5.5). The final error rate obtained is 4.6336%.

4.4. Hate is a Powerful Signal in Predicting Love

A very interesting question arises: which love-hate configuration provides the most powerful signal in predicting a user's preference for an item? For an empirical answer to this question, we turn to the results from training the logistic regression classifier as discussed in Sec. 4.3.

The coefficients of the logistic regression classifier quantify how much each feature (i.e. counts of each square configuration) contributes to or against the prediction that the target user has highly rated the target item. The logistic regression coefficients corresponding to the simple counts of each square configuration in the shared-album category are displayed in Fig. 5.

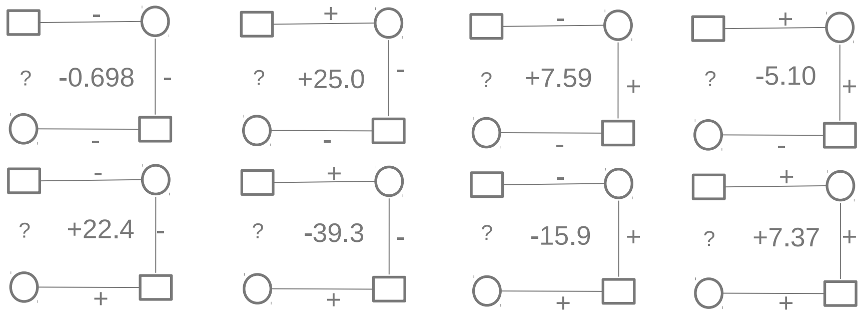


Figure 5: Logistic regression coefficients (in 10^{-3}) for each love-hate square configuration in predicting a user's highly rated items.

From examining Fig. 5, we discover a very surprising result: the most powerful signal in predicting love (i.e. user's high rating of an item) comes from the *I-hate-it* edges. Let's interpret the configuration with the highest logistic regression coefficient (i.e. configuration No. 1, refer to Fig. 3): both users hate a common item and the other user loves the target item; for the configurations with the second highest coefficient, configuration No. 4, both items are hated by a common user and the target user loves the other item. Such common hatred toward the same item by two users or hatred by the same user to two items seems to provide a very strong signal in predicting target user's love toward the target item, provided that preference has been expressed either by the other user toward the target item or by the target user toward the other item.

The finding that configuration No. 1 provides a strong signal in predicting a high rating may have interesting ramifications: for example, a real-world recommender system based on square counting may find itself explaining to a user that item X is recommended because the user hates items Y and Z! Also, next time you meet up with friends, besides talking about which of the latest songs and movies you and your friends like, perhaps a discussion on which songs or movies you and your friends hate may lead to you to your next favorite song or movie!

Another interesting observation is that the configuration with the most negative logistic regression coefficient is configuration No. 5, which corresponds to the situation where the two users disagree on the other item and the other user loves the target item. Such situation is the most likely to predict that the target user will NOT like the target item.

5. Future Work

For future work, we want to evaluate the performance of the square counting method for the task of predicting user-item rating scores, which is the goal of track 1 of the KDD Cup 2011 competition. We plan to benchmark the square counting method in terms of accuracy and computational complexity against other popular methods from literature (e.g. k-Nearest-Neighbor, Matrix Factorization based methods, etc). In addition, we intend to evaluate the overall performance when blending the results of the square counting method with results from popular methods from the literature. We want to benchmark the accuracy of the square counting methods against other large-scale rating datasets, such as the Netflix prize dataset. Finally, we want to explore other enhancement techniques to further improve the square counting method.

6. Acknowledgments

This paper is dedicated to our friend and colleague, Justin Kessler, who dedicated the very last minutes of his life to the advancement of data mining research. His passion for life and mathematics has inspired us all.

References

Thomas Q. Bartee and Abigail Gertner. A multi-source recommender system for intelligence analysis. In *MITRE's Annual Technology Symposium*. MITRE Corporation, 2006.

Robert M. Bell and Yehuda Koren. Improved neighborhood-based collaborative filtering. In *KDD-Cup and Workshop*, San Jose, California, 2007.

Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The yahoo! music dataset and kdd-cup'11. *Journal Of Machine Learning Research W& CP*, 18:1–12, 2012.

Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. 22nd SIGIR*, pages 230–237, 1999.

Michael Jahrer, Andreas Töscher, and Robert Legenstein. Combining predictions for accurate recommender systems. In *Proc. KDD*, pages 693–702, 2010.

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. KDD*, pages 426–434, 2008.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proc. WWW*, pages 641–650, 2010.

Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, 2003.

M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64(2):026118, Jul 2001. doi: 10.1103/PhysRevE.64.026118.

Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*, 2007.

Steffen Rendle. Factorization machines. In *Proc. 10th ICDM*. IEEE Computer Society, 2010.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proc. 24th ICML*, pages 791–798, 2007.

Tao Zhou, Jie Ren, Matú š Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Phys. Rev. E*, 76(4):046115, 2007.

Tao Zhou, Zoltn Kuscsik, Jian-Guo Liu, Mat Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.