

CSCI E-280 Final Project, 2004

Movie Recommender

Clifford Lyon (clyon@fas.harvard.edu)

1 Introduction

The project objective is to determine movies a person may like based on knowledge of other movies the person liked. The modified problem for classification is to predict a person's rating on an unseen movie using knowledge from a database of movie ratings. The database stores ratings of various movies contributed by many different users. If a rating on a single movie can be predicted, by extension one can predict the ratings for all the movies in the database that a person has not seen. Movies the person would like can be chosen from among favorable predictions. Collaborative Filtering (CF) or recommender systems explore user preference data to predict interest on unseen items. There are fundamentally two types of CF algorithms: memory-based, and model-based. Memory-based algorithms store a matrix of ratings, and missing items are predicted using a weighted average. Weights represent similarity between vectors of user-ratings or item-ratings. Model-based CF uses a probabilistic approach to develop a model of ratings. In this project, algorithms of both types are evaluated. The project uses the Weka software framework, and so recommendation is treated as a classification problem. Though CF was not part of the syllabus, much of the material from the course is reviewed and explored, both in terms of particular algorithms and in terms of measuring performance.

2 Approach

The Weka software package was used to build a framework for evaluating the various recommendation algorithms. The input data was from the GroupLens Project, available at <http://www.grouplens.org/data>. The abstract *Recommender* class is the base class for all the recommenders. Because a probability distribution preserves all the information about the prediction, *Recommender* extends Weka's *DistributionClassifier*. Ratings are on a scale of 1 to 5 stars with 5 the best, so ratings used for training are all positive integers. However, the predictions are floating point numbers. The outputs may be rounded if a best guess is required, however, Weka's *distributionForInstance()* method allows the prediction to be delivered with minimum information loss which may be important in a model-based algorithm.

All the algorithms were tested on the GroupLens data using an averaged ten-fold cross-validation over ten trials on the 100K record dataset. The choice was made after reviewing Witten[1], pp. 127: “When going for an accurate error estimate, it is standard procedure to repeat the cross-validation process ten times—that is, ten tenfold cross-validations—and average the results. This involves invoking the learning algorithm one-hundred times, on datasets that are all nine-tenths the size of the original one.” All the figures presented in this paper for the ten ten-fold validations may be reproduced using the random seed 2337742489427657586. When comparing two algorithms using a t-test, the pairs compared are over one trial of ten-fold validation.

Stratifying the data for cross-validation is usually done on the class attribute. For this project, additional steps were taken to ensure a uniform item distribution across the folds – the extra steps ensure that unseen items were rated by other users 99.8% of the time. The metrics used to evaluate algorithm performance were:

- **Mean absolute error (MAE):** A standard metric for evaluating CF algorithms. The MAE is sensitive to the number of folds used for cross-validation, since the more items are used for training, the better the prediction (see chart below). For this reason, MAE and other metrics are meaningful only within the context of this project, or when fairly compared with measures from a framework using the same tests. It should be noted, the MAE is computed using the difference between the floating point prediction and the user rating. If rounded predictions are compared instead the MAE is significantly lower, with the best algorithms perform below 0.70.
- **Error Rate:** Percentage of incorrect rounded predictions.
- **Percent correct guesses:** *Decision support accuracy* metrics look at prediction as a binary problem; users either like something or they don't. The GroupLens rating system assigns the following descriptions to ratings: 1=Awful, 2=Fairly Bad, 3=It's

OK, 4=Will Enjoy, and 5=Must See. Based on this, it can be assumed that a 4 or a 5 means a user likes a movie. Percent correct guesses is the percentage of predictions that were a 4 or 5 when the actual prediction was a 4 or a 5, or a 1, 2 or 3 when the actual rating fell into that range. This metric did not appear in the material reviewed for the project. It was included because it directly addresses the problem of how well a predictor can determine movies a person would like.

- **Average time per prediction in milliseconds:** This is wall-clock time per prediction. It is useful for comparison. All the ratings in this paper were done on the same machine. (However some of the sample output was done on a slower machine.)

As mentioned, the number of folds used for testing does effect the performance of the algorithms. The chart below shows that as the number of folds and the amount of data used for training increase, the MAE goes down. This test was done using a single algorithm with the same random seed for each cross-validation. The tests performed in this project produce the rightmost and most favorable measure of Mean Absolute Error (MAE).

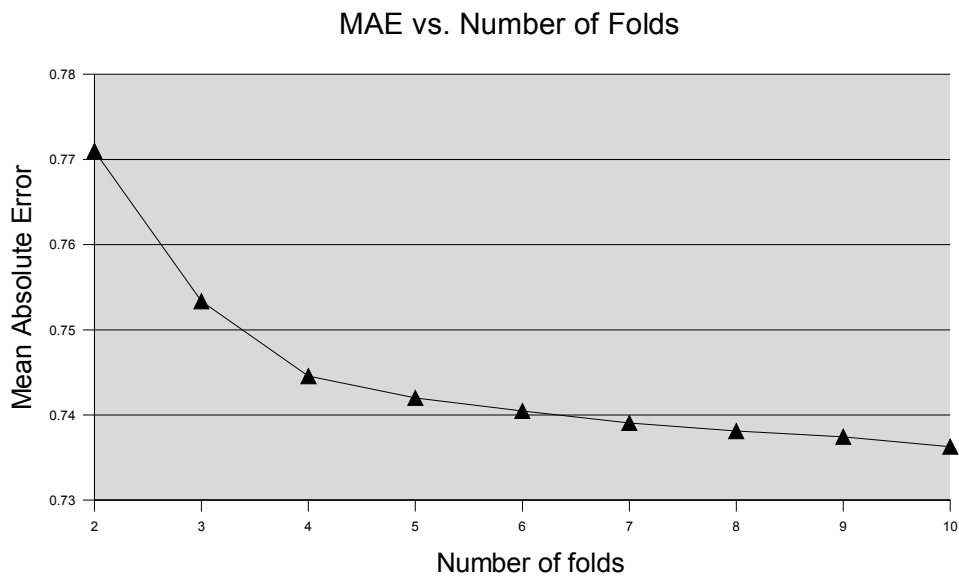


Figure 1. Effect of Number of folds on MAE

3 Algorithm Performance

Several algorithms and variations were reviewed for this project. Chosen for evaluation were: the Pearson correlation coefficient algorithm, vector similarity, k-nearest neighbor, naïve Bayes, and an associative analysis-based heuristic, which was named “association heuristic.” Random and average recommenders were created for baselines. Finally, there are meta-recommenders, which use clustering and partitioning to try and improve the performance of underlying recommenders. As mentioned previously, the algorithms themselves were implemented from the abstract class *Recommender*, which was extended from Weka's *DistributionClassifier* class. *Recommender* adds a prediction method to the classifier.

Pearson

The Pearson algorithm was adapted from Resnick[2]. It uses the Pearson correlation as a similarity metric between vectors of ratings. Typically, memory-based algorithms such as this one use some sort of ratings-based measure to describe the association between users. The measure acts as a weight when predicting a user's rating on an unseen item. This is described by Breese et al[3]. The similarity between two users x and y using the Pearson correlation coefficient is:

$$Corr(x, y) = \frac{\sum_i (v_{x,i} - \bar{v}_x)(v_{y,i} - \bar{v}_y)}{\sqrt{\sum (v_{x,i} - \bar{v}_x)^2} \sqrt{\sum (v_{y,i} - \bar{v}_y)^2}} \quad (1)$$

where v represents vectors of user ratings. The first step after reading the training data is to pre-compute this metric for all the user pairs in the input data set. This greatly speeds up the time it takes to make each prediction. During the test phase, the recommender is asked to predict a user's rating on unseen item. The correlation is used to weight the average rating of the user and bias the rating towards similar users, and away from dissimilar users. The predicted vote of user u_i on item m is:

$$Pred(u_i, m) = \bar{v}_u + \sum_j Corr(u_i, u_j)(v_{j,m} - \bar{v}_j) \quad (2)$$

where $Corr(u_i, u_j)$ is equation (1), and v are ratings vectors, and the sum is over the vectors of user ratings.

The Pearson Recommender performed best overall without tuning. Over ten ten-fold

validation trials, results were as follows:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.73751	0.58369	0.71016	0.06437
2	0.73724	0.58405	0.70952	0.07348
3	0.73763	0.58309	0.70964	0.06107
4	0.73775	0.58407	0.70952	0.06267
5	0.73843	0.58367	0.70916	0.06123
6	0.73764	0.58328	0.71000	0.06581
7	0.73773	0.58414	0.70943	0.06528
8	0.73793	0.58316	0.70998	0.06427
9	0.73746	0.58280	0.70976	0.06418
10	0.73777	0.58389	0.71011	0.06763
Avg	0.73771	0.58358	0.70973	0.06500

Table 1. Pearson Performance

Performance of this recommender can be quantified by comparing performance to a suitable baseline recommender. There are three to choose from: random, average user rating, and average item rating. Because the prediction is a weighted user average, it is interesting to compare performance with the average user rating recommender. A paired t-test establishes a level of confidence that the Pearson MAE is significantly better than the average alone. Using the same random seed as above, over a ten-fold validation, the value of t was 76.88, for 9 degrees of freedom. This is greater than t for 0.005 in a lookup table. Because this is a two-tailed test, 0.005 yields 99% confidence that the means are different and that the Pearson metric performs better than average. Computing the p value directly gives 5.385×10^{-14} . A t-test using error rate or percentage of correct guesses also gives 99% confidence the algorithm was better than a simple average. Of course, the average recommender was significantly faster because no computation was involved.

There are several improvements for memory-based algorithms suggested by Breese[3], including default voting, inverse user frequency, and case amplification. For the Pearson algorithm, default voting and case amplification were evaluated. Default voting did not perform as well as the original method. The value of t for a t-test comparing default voting and the original method using MAE was 9.625, which yields 99% confidence that the performance of default voting and the original algorithm were different. For the error rate, t was 2.07, which gives a 90% confidence. For the percentage of correct guesses, t was 4.011, giving 99% confidence.

Case amplification was better but very close in performance to the original. A paired t-test using MAE of original vs. case amplification gives a $t=8.747$. Though the performances were quite close, confidence is 99% that case amplification is a

significant improvement over the original algorithm with respect to MAE. For error rate, the value of t was 3.26, which is also 99% confidence for 9 degrees of freedom. For correct guesses, the value of t was 2.95, which allows a 98% confidence the means were different. The difference in time/prediction was insignificant.

For case amplification, the following adjustment is made to the correlation $Corr$:

$$Corr' = \begin{cases} Corr^p & \text{if } Corr \geq 0, \\ -(-Corr^p) & \text{otherwise} \end{cases} \quad (3)$$

The default value for p used for this evaluation was 1.5. The results for the *PearsonCaseAmplification* recommender over ten trials of ten-fold validation averaged each trial:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.73635	0.58186	0.71118	0.06343
2	0.73603	0.58173	0.71092	0.06939
3	0.73637	0.58251	0.71040	0.06235
4	0.73659	0.58282	0.71007	0.06655
5	0.73720	0.58208	0.71035	0.06297
6	0.73658	0.58235	0.71004	0.06637
7	0.73639	0.58319	0.71025	0.06670
8	0.73690	0.58267	0.71026	0.06391
9	0.73640	0.58226	0.71018	0.06470
10	0.73660	0.58210	0.71149	0.06734
Avg	0.73654	0.58236	0.71051	0.06537

Table 2. Pearson Case Amplification Performance

For default voting, the correlation becomes more complex:

$$Corr(x, y) = \frac{(n+k)(\sum_i v_{x,i} v_{y,i} + kd^2) - (\sum_i v_{x,i} + kd)(\sum_i v_{y,i} + kd)}{\sqrt{((n+k)(\sum_i v_{x,i}^2 + kd^2) - (\sum_i v_{x,i} + kd)^2)((n+k)(\sum_i v_{y,i}^2 + kd^2) - (\sum_i v_{y,i} + kd)^2)}} \quad (4)$$

Where d is the value of a default vote, neutral or biased towards a negative rating, and k is a count of votes to add for items unseen by both the user and the user's neighbors. Typical values used for d were 2.5 or 3, and 50 for k . The results of default voting are not quite as good as the original Pearson correlation coefficient algorithm:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.74197	0.58570	0.70718	0.06862
2	0.74170	0.58581	0.70683	0.06719
3	0.74210	0.58564	0.70672	0.06391
4	0.74239	0.58462	0.70821	0.06265
5	0.74320	0.58512	0.70730	0.05988
6	0.74246	0.58534	0.70770	0.06502
7	0.74236	0.58480	0.70856	0.06703
8	0.74231	0.58591	0.70694	0.06467
9	0.74173	0.58616	0.70650	0.06500
10	0.74219	0.58586	0.70706	0.06969
Avg	0.74224	0.58550	0.70730	0.06537

Table 3. Pearson Default Voting Performance

Results show case amplification was better than the original algorithm on this data, and that default voting was worse.

Vector Similarity

The vector similarity algorithm looks at the arrays of user ratings as vectors, and uses the cosine of the angle between the vectors as an index of similarity. The equation for similarity between users in the vector similarity algorithm becomes the normalized dot product of the two vectors, or the cosine of the angle between them:

$$Corr(x, y) = \frac{v_x \cdot v_y}{|v_x| |v_y|} \quad (5)$$

The equation for calculating the prediction is the same as the Pearson recommender (2). A paired t-test between vector similarity and the Pearson correlation algorithm with default voting using MAE gave $t=38.0371$, so the differences in performance are significant. Values of t for other metrics also produced a 99% confidence, including milliseconds per prediction. Here are the results of ten ten-fold validation of vector similarity:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.76027	0.59545	0.69314	0.08264
2	0.76014	0.59571	0.69251	0.07737
3	0.76074	0.59576	0.69254	0.07782
4	0.76114	0.59610	0.69223	0.07392
5	0.76162	0.59497	0.69277	0.07279
6	0.76102	0.59608	0.69323	0.07859
7	0.76111	0.59505	0.69339	0.07642
8	0.76108	0.59574	0.69286	0.07686
9	0.76040	0.59578	0.69339	0.07971
10	0.76049	0.59590	0.69290	0.08531
Avg	0.76080	0.59565	0.69290	0.07814

Table 4. Vector Similarity Performance

The inverse user frequency extension was also evaluated using the vector similarity algorithm. The notion of inverse user frequency comes from the idea of inverse document frequency in information retrieval. The idea is that words occurring frequently are less useful for establishing an association between documents, because all documents have them. Inverse user frequency extends this idea, assuming that if everyone likes an item, it is not useful for establishing similarity between two given users.

To apply the idea, the ratings of a pair of users (x,y) on an item i are transformed while computing (5):

$$r_{item} = r_{item} \cdot \log\left(\frac{\text{count}(\text{users})}{\text{count}(\text{users}_{item})}\right) \quad (6)$$

where users_{item} are users who rated the item. Notice that if everyone rated an item, the rating becomes zero. The performance was very close to and slightly better than vector similarity alone:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.76021	0.59544	0.69318	0.07020
2	0.76009	0.59573	0.69255	0.07650
3	0.76068	0.59575	0.69261	0.08120
4	0.76110	0.59600	0.69226	0.08280
5	0.76157	0.59501	0.69274	0.07970
6	0.76095	0.59617	0.69323	0.08130
7	0.76106	0.59502	0.69339	0.07960
8	0.76103	0.59591	0.69272	0.08910
9	0.76034	0.59580	0.69341	0.07970
10	0.76044	0.59597	0.69289	0.08130
Avg	0.76075	0.59568	0.69290	0.08014

Table 5. Vector Similarity with Inverse User Frequency Performance

Because the performances are so close, it is useful to see the results of a paired t-test for these two algorithms. The value of t comparing MAE measures from ten-fold cross-validation was 6.015; this is high enough to give a 99% confidence that the means are different. For error rate and percent correct guesses, however, the differences were not significant. For milliseconds per prediction, the t was 2.63, giving 95% confidence.

Also evaluated was an item-based version of vector similarity. Like the other memory-based algorithms, a matrix of ratings is used, except that the orientation is transposed. Instead of vectors of ratings by user, vectors of ratings by item are stored. There has been some interest in this idea because typically there are fewer users than items, so

given an equal number of neighboring vectors in both cases algorithms may be faster. The vector similarity algorithm was chosen not because it limits neighborhood size, which it doesn't, but simply to evaluate the performance of the two approaches side-by-side. The result was that the item-based version did not perform as well as the user-based version; the average MAE over ten ten-fold cross-validations was 0.77966, and the time per prediction doubled. A paired t-test between vector similarity and the item based algorithm for MAE gives $t=16.681$, yielding 99% confidence the two means are different. T-tests for other metrics were also in the 99% range, except for the percentage of correct guesses, where $t=2.63$, 95% confidence. The performance of the item-based algorithm:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.77929	0.60417	0.68897	0.15721
2	0.77893	0.60406	0.68912	0.15534
3	0.78020	0.60432	0.68853	0.14941
4	0.77984	0.60414	0.68801	0.14871
5	0.78091	0.60460	0.68759	0.14874
6	0.77982	0.60501	0.68772	0.14922
7	0.78017	0.60447	0.68811	0.14861
8	0.77955	0.60418	0.68879	0.15231
9	0.77899	0.60449	0.68892	0.15064
10	0.77890	0.60513	0.68845	0.14810
Avg	0.77966	0.60446	0.68842	0.15083

Table 6. Item-based Vector Similarity Performance

Overall, vector similarity had good but not the best performance. Inverse user frequency helped slightly but significantly, and the item-based version was not an improvement on the original algorithm.

k-Nearest Neighbor

The implementation for k-nearest-neighbor was adapted from some information from Paul Perry[4]. The first step in the algorithm is to compute the mean square difference between each user using ratings they had in common:

$$Diff(u_i, u_j) = \frac{\sum_{items} (r_{i,item} - r_{j,item})^2}{|items|} \quad (7)$$

Next, the differences are translated into weights:

$$w(u_i, u_j) = \frac{argmax(Diff + 1) - Diff(u_i, u_j)}{argmax(Diff + 1)} \quad (8)$$

The predicted rating is a weighted sum of the k-nearest neighbors who rated that item:

$$Pred (user , item) = \frac{\sum_{neighbors} (rating_{neighbor,item} \times w(user , neighbor))}{\sum_{neighbors} w(user , neighbor)} \quad (9)$$

The neighborhood size influences the performance of the algorithm. Though the effect is small, the optimal performance on the GroupLens dataset occurs at k=19:

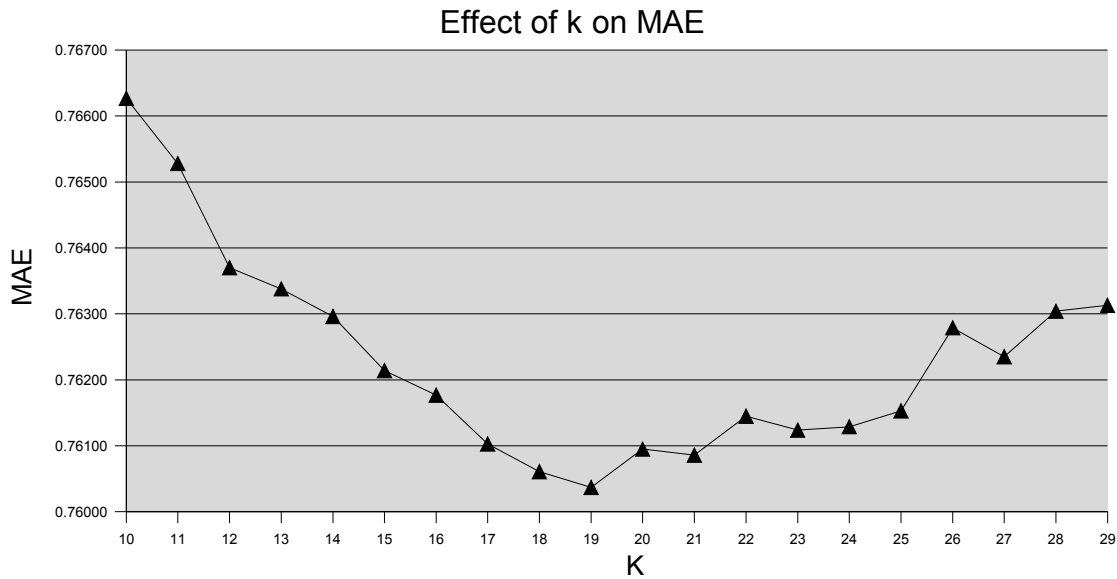


Figure 2. Effect of Neighborhood Size on MAE

The 100K GroupLens dataset guarantees that each user rated as least 20 movies – by coincidence the optimal performance is found at a neighborhood of about the same size. The overall k-means performance was as follows:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.76027	0.59518	0.70128	0.53233
2	0.76031	0.59503	0.70081	0.51748
3	0.76030	0.59487	0.70106	0.51923
4	0.76180	0.59534	0.69958	0.52034
5	0.76161	0.59611	0.69997	0.52047
6	0.76097	0.59615	0.70059	0.51531
7	0.76155	0.59584	0.69964	0.51423
8	0.76094	0.59518	0.70174	0.51577
9	0.76064	0.59547	0.70010	0.52435
10	0.76037	0.59561	0.70135	0.51799
Avg	0.76088	0.59548	0.70061	0.51975

Table 7. k-Nearest Neighbor Performance

The k-nearest neighbor performance was on par with vector similarity. A paired t-test for the two algorithms using MAE gave $t=0.0414$, so the performance of the algorithms is not significantly different with respect to MAE. Notice that the percentage of correct guesses by k-nearest neighbor is over 70%, which is significantly better: $t=5.849$ gives 99% confidence. The time per prediction by k-nearest neighbor is significantly longer. This is because the neighbors are selected at prediction time. The prediction time was significantly worse with 99% confidence, $t=73.829$. The difference in error rates was not significant.

Naïve Bayes

In contrast to the preceding algorithms, the *NaiveBayes* recommender is a model-based rather than memory-based recommender. The logic for the algorithm was taken from course material and from Witten[1]. Smoothed counts are used. The format of our Weka record and the problem do not line up as well as some of the work done in our class. Every record in the ratings dataset contains a user, an item, and a rating, with the rating taking the role of the class attribute. Generally, naïve Bayes predicts the posterior probability of a class, given the prior probabilities of other attributes. For the recommender it is the rating specific to the item that is predicted rather than the class, given the user's ratings on other movies. The ratings.arff dataset is really a kind of compact representation of a sparse vector associated with every user. The problem for this recommender becomes:

$$P(\text{rating}_{item} | R_{items}) = \frac{P(R_{items} | \text{rating}_{item}) P(\text{rating}_{item})}{P(R_{items})} \quad (10)$$

where the denominator is ignored, rating_{item} is the rating for a given item, and R_{items} are the user's existing ratings. Naïve Bayes performance on this data set was not as good as the memory-based algorithms. Also, the training and prediction times were significantly longer:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.78792	0.59770	0.67497	14.89589
2	0.78763	0.59769	0.67464	15.79172
3	0.78700	0.59642	0.67704	15.11672
4	0.78620	0.59602	0.67692	15.81127
5	0.78361	0.59393	0.67842	14.61314
6	0.78658	0.59606	0.67536	17.26879
7	0.78776	0.59675	0.67553	17.35468
8	0.78891	0.59720	0.67562	15.84782
9	0.78748	0.59693	0.67480	16.05155
10	0.78731	0.59723	0.67584	18.10827
Avg	0.78704	0.59659	0.67591	16.08599

Table 8. Naive Bayes Performance

For some data, this algorithm works quite well, especially when the joint probabilities of the other attributes in the record are in fact independent. It makes sense that the performance on this data is worse than some other methods: user's ratings on different movies are related to each other by the user; discrete ratings are not stochastically independent. The algorithm closest to naïve Bayes for MAE was the item-based version of vector similarity; a paired t-test gives $t=6.110$, giving 99% confidence that the vector similarity algorithm was better. Closest in error rate was vector similarity with inverse user frequency; a paired t-test for these two algorithms using error rate gives $t=1.387$, giving 80% confidence the vector similarity variation performed better. In the next section, the algorithm is compared with the association heuristic for percent of correct guesses and milliseconds per prediction.

Association Heuristic

This recommender is based on metrics used in market basket analysis: support, confidence, and improvement. Witten[1] uses the term “coverage” instead of support, and “accuracy” instead of confidence, but the metrics are the same. Support and confidence are described formally by Agrawal and Srikant[5]. For this recommender, these metrics are in reference to particular ratings among all ratings. Experiments were performed using different size item sets, and pairs of items were found to be most useful. The following metrics were used for the heuristic:

$$Support(m_{i,r1} \Rightarrow m_{j,r2}) = \frac{count_{users}(m_{i,r1}, m_{j,r2})}{count_{users}(m_i, m_j)} \quad (11)$$

Support for a pair of movies (m_i, m_j) with a particular rating r is the count of users who rated the movies with the given ratings divided by the count of users who rated both movies. It is also possible to look at the support simply as the count of users who rated two given movies over the count of all users, which gives a broader sense of support for

the two movies. However, in this case what was interesting was a measure of support for the ratings, not just whether the movies were seen or not. The next metric used was confidence, or accuracy:

$$Confidence(m_{i,r1} \Rightarrow m_{j,r2}) = \frac{count_{users}(m_{i,r1}, m_{j,r2})}{count_{users}(m_{i,r1})} \quad (12)$$

Confidence is that percentage of users who rated movie m_i some given rating that also rated m_j some given rating. So, if 100 users gave Star Wars a 5, and 40 of those users gave Casablanca a 4, there is 40% confidence that given rating of 5 for Star Wars, Casablanca will be a 4.

The third metric used was improvement, or lift. This metric is based on the notion that for stochastically independent events, the joint probability is equal to the product of the individual probabilities. Improvement is the joint probability divided by the product of the individual probabilities. If improvement is greater than 1, it may be inferred that the joint probability represents a meaningful association. Improvement is used in my heuristic to qualify the score of a rule:

$$Improvement(m_{i,r}, m_{j,r}) = \frac{P(m_{i,r}, m_{j,r})}{P(m_{i,r})P(m_{j,r})} \quad (13)$$

Improvement can help eliminate candidate rules that have high confidence and support but are not meaningful – the same way inverse user frequency helps qualify similarity by reducing the weight of similarities that are present for all users in the system. In the example above, there is a 40% confidence that Casablanca is a 4 given Star Wars is a 5. But it may also be true that 60% of all users gave Casablanca a 4. In this case, the association between Star Wars and Casablanca isn't really useful.

Using the metrics above, an original algorithm was developed through experimentation. The reason for exploring an original algorithm was two-fold: first, the size of the number of potential items in the itemset discouraged the use of known algorithms like Weka's Apriori, and second, my feeling was that the association metrics applied to the dataset using “common sense” might yield some good results. In fact, the MAE was better than Naïve Bayes, but the time per prediction was the largest of any algorithm.

Heuristic Algorithm:

```
unseen u; /* the movie we're asked to rate */
P(r);    /* probability distribution for the result */
forall movies seen s do
    maxScore := 0;
    bestIndex := -1;
    for ( r = minRating; r <= maxRating; r++ ) do begin
        support := Support( s=givenRating, u=r);
        confidence:= Confidence( s=givenRating, u=r);
        improvement:=Improvement( s=givenRating, u=r);
        score = support x confidence;
        if ( score > maxScore ) {
            maxScore := score;
            bestIndex := r;
        }
        if ( improvement > 1 ) {
            P(r) := P(r) + score;
        }
    end;
    P(bestIndex) := P(bestIndex) + maxScore + 1;
end;
```

The algorithm develops a probability distribution by scoring the chance that a rating for a given movie will be a given rating based on past ratings. The notion of probability here is not a rigorous mathematical model, but as the name suggests, an original heuristic. It is based on a sum of scores for each rating and uses Weka's probability distribution to store the sums. The score is the product of the confidence and the support. Intuitively, the maximum combination of support and confidence is a better choice for a recommendation, so the maximum score is rewarded beyond the sum of the scores. Scores that show no improvement are ignored.

The results of the Association Heuristic were as follows:

Trial	MAE	Error Rate	% Guessed	millis/pred
1	0.76180	0.59308	0.67992	55.41022
2	0.76410	0.59480	0.67952	64.57758
3	0.76342	0.59503	0.68005	55.87174
4	0.76346	0.59440	0.67972	59.46784
5	0.76141	0.59218	0.68190	57.91828
6	0.76586	0.59522	0.67833	73.95057
7	0.76346	0.59413	0.67989	68.32543
8	0.76528	0.59521	0.67857	66.71965
9	0.76715	0.59649	0.67827	59.31308
10	0.76557	0.59567	0.67868	72.99409
Avg	0.76415	0.59462	0.67949	63.45485

Table 9. Association Heuristic Performance

As mentioned, the time per prediction stands out in this algorithm. The support, confidence, and improvement metrics could be pre-computed during the training phase. Whenever they are computed they do however have a cost associated with them that memory-based algorithms don't. Comparing the performance of the association heuristic with naïve Bayes using paired t-tests gives 99% confidence of significant differences for all metrics – although the differences are not all in the same direction: the association heuristic performed significantly better than naïve Bayes with regards to MAE error rate, and percentage of correct guesses, but significantly worse with respect to milliseconds per prediction.

4 Clustering

The intuition behind applying clustering to the collaborative filtering problem is that by grouping similar users together, the in-cluster variation is reduced, and so a metric that uses a weighted average would be more accurate because the in-cluster average would have less variance, and an algorithm that develops a probabilistic model of ratings would benefit from building many small models based on similar populations. The *ClusteredRecommender* uses simple k-means clustering. Various numbers of clusters were tested. The input into the clustering algorithm is the set of long sparse vectors representing user ratings. This effectively assigns each user in the training data to a cluster instance. For each cluster created, a separate input data set is created and a dedicated recommender is instantiated and trained. When tested, first the cluster is selected based on the user, and then the recommendation is fetched from that user's recommender. Any of the algorithms in the project presented above may be used with the *ClusteredRecommender*. Because the in-cluster averages are expected to be more accurate, the *ClusteredRecommender* performance is evaluated using the Pearson correlation coefficient algorithm, which uses weighted-average. The following chart shows the MAE for the Pearson algorithm alone, and then the MAE using k-means clustering for various numbers of clusters. All of the metrics were produced using the same random seed and algorithm.

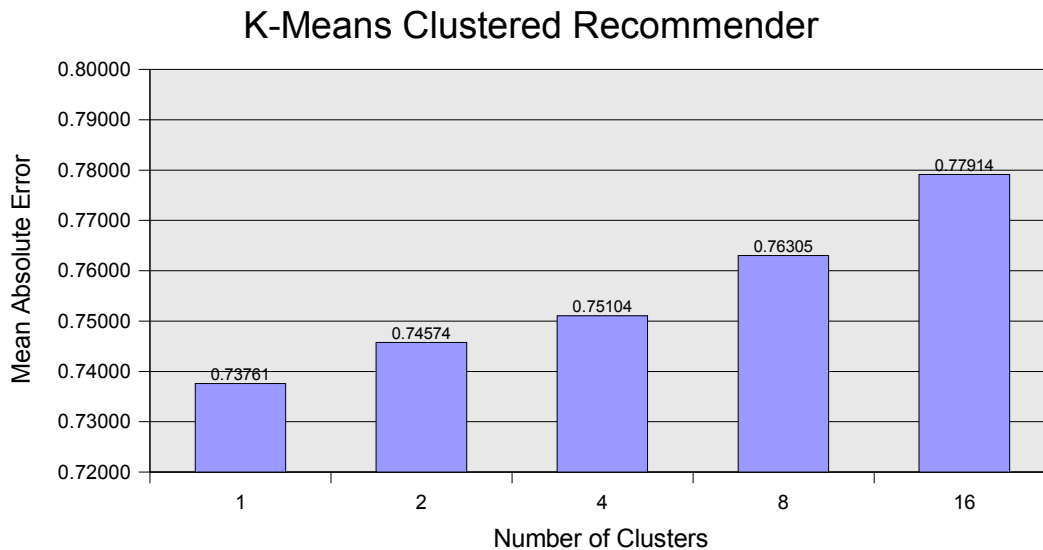


Figure 3. k-Means Clustering Recommender MAE

From the chart above, it can be seen that the effectiveness of the recommender is reduced as the number of clusters is increased.

5 Partitioning

The GroupLens data included information about the users and the movies, so it seemed logical to investigate the effect of segmenting the data by user and item attributes to see if performance could be improved. For users, there was information about gender, age, and job; movies had a list of genres to which they belonged. There were 19 genres total, and a single movie can belong to multiple genres (e.g. action-comedy). This led to different methods when partitioning by user or item attributes: for user attributes, any combination of attributes may be selected as a partition key for the data; the training data is hash-partitioned by that key, and separate classifiers are built for each partition. Each instance belongs to one and only one partition. For movie attributes, again the data is partitioned, but this time a single instance may belong to multiple partitions. There is a partition and classifier created for every genre, and items are added to each partition they are associated with – for example, action-comedies are added to the action and the comedy partition. In the test phase, an item belonging to many partitions is classified by all of them, and the average prediction is returned.

The effect of partitioning by various attributes can be observed by using a single classifier and a single random seed. The following data was computed using the vector similarity algorithm:

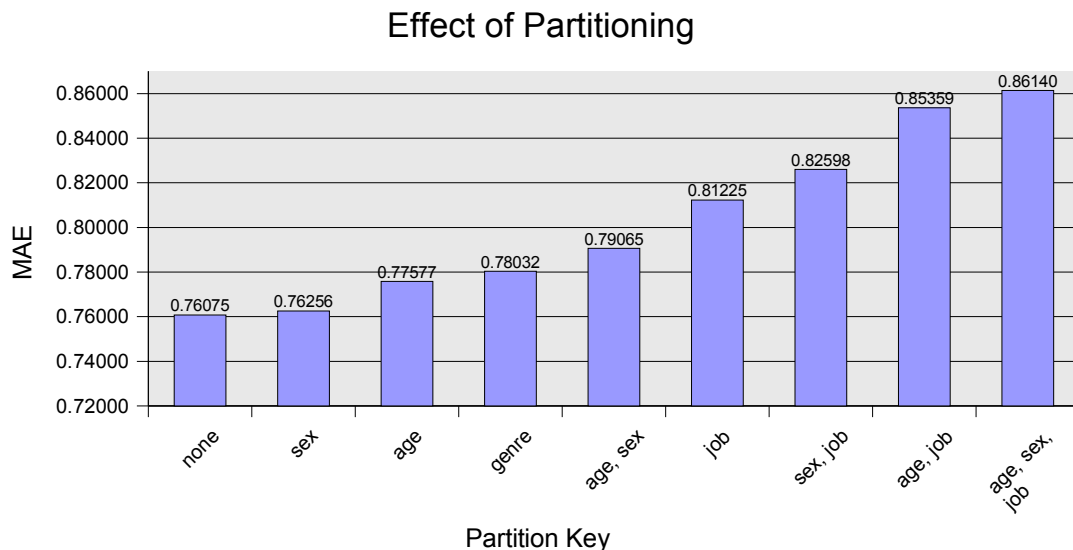


Figure 4. Effect of Data Partitioning on MAE

Just as with clustering, separating the data into partitions based on user or item attributes reduces the effectiveness of the recommender. The recommenders did better when trained on all of the data.

6 Overall Performances

It is interesting to see how the performances compared with each other. Here are some overall results for each metric. The first chart shows the principal metric used, MAE:

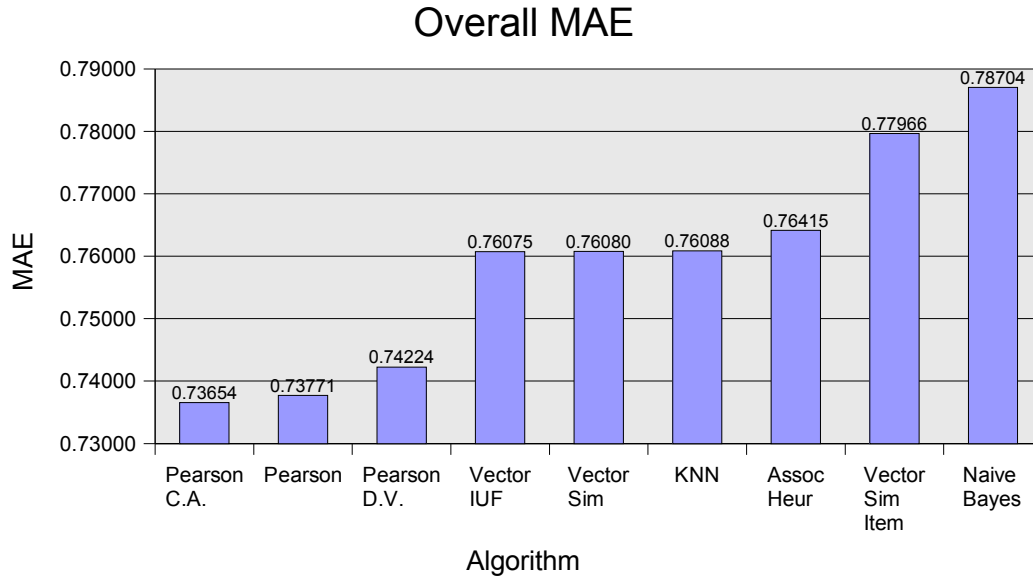


Figure 5. Overall Mean Absolute Error

The error rate of the algorithms shows percentage of time the rounded prediction was wrong. It is interesting to notice that the order is different from the MAE chart:

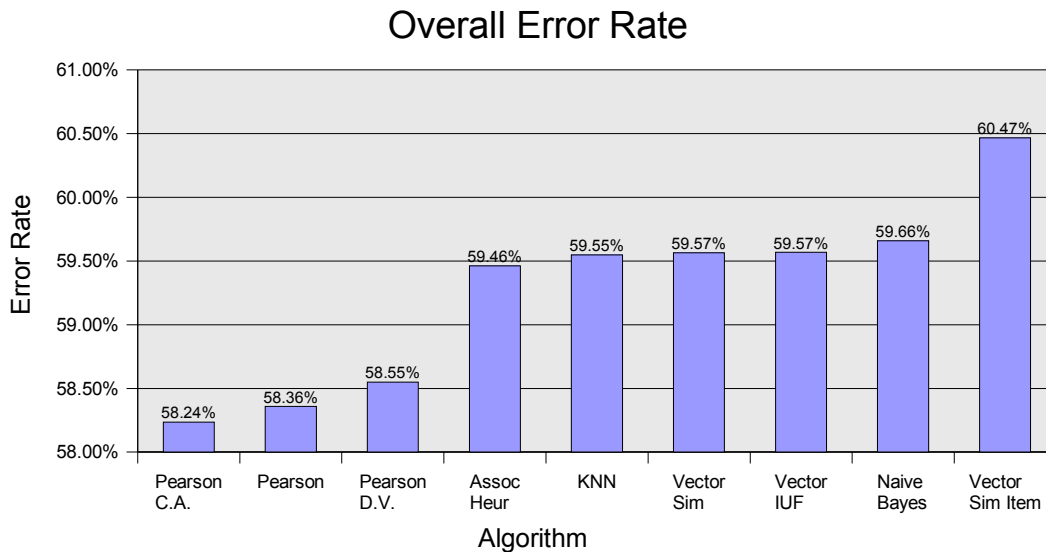


Figure 6. Overall Error Rate

The percentage of correct guesses is the ability of an algorithm to make the right binary classification for a user: a movie is good (4-5), or not (1-3):

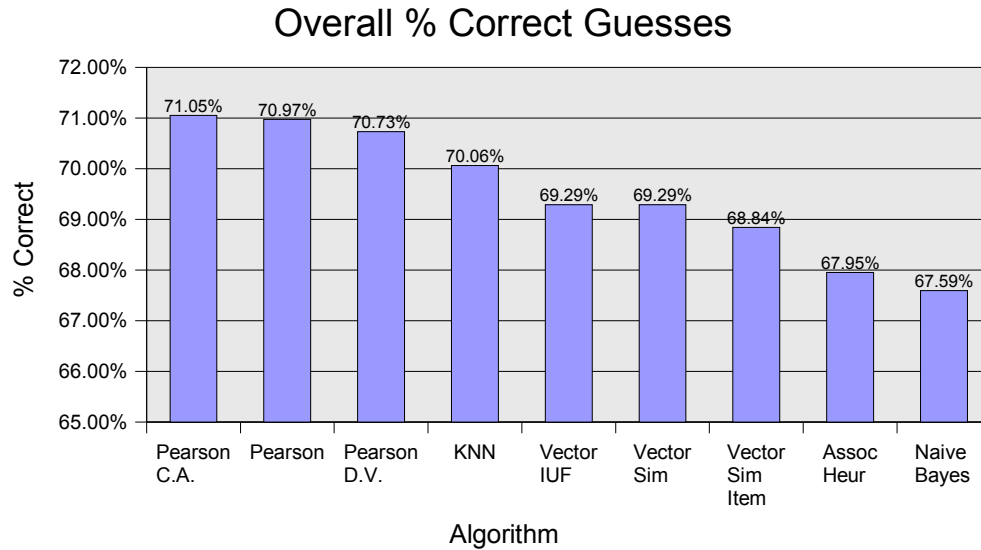


Figure 7. Overall Percent Correct Liked/Not liked

Finally, we look at the time it took to make each prediction. It should be noted that this figure is biased towards algorithms that push computation to the training phase, and that through some design effort, the slower algorithms could be sped up by pushing time spent during predictions to the training phase.

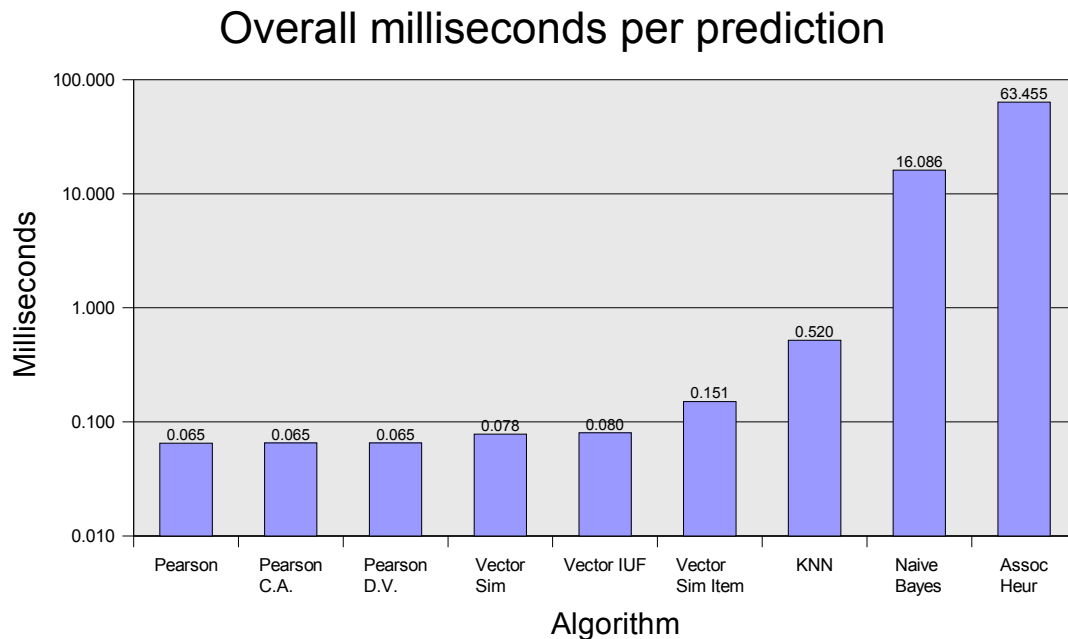


Figure 8. Overall Milliseconds per Prediction

8 Conclusion

The objective of this project was to be able to recommend movies a person would like based on a knowledge of other movies the person liked. To that end, using data from GroupLens and the Weka data mining software, a framework for evaluating recommender algorithms was developed. The best algorithm evaluated for this project predicted correctly if a user would like a movie 71% of the time, and the average prediction of the top three algorithms were within three-quarters of a star (out of five). Several algorithms and variations were tested, including the following: Pearson correlation coefficient algorithm, vector similarity, k-nearest neighbor, and naïve Bayes. An original algorithm, association heuristic, was written using association rule metrics. Several extensions to memory-based algorithms were evaluated: default voting, case amplification, and inverse user frequency. Item-based and user-based versions of vector similarity were tested. Finally, clustering and partitioning were tested to see if the performance of existing algorithms could be improved by partitioning the data and training multiple recommenders. The best overall algorithm was the Pearson correlation coefficient algorithm from Resnick[2], using case amplification from Breese [3]. It had the lowest MAE, the lowest error rate, the highest percentage of correct guesses, and was the second fastest algorithm. The fastest recommender was the unmodified Pearson algorithm. The methods that produced the fastest and best recommendations in this project were the memory-based algorithms. It was shown that algorithms can be improved by implementing the extensions of case amplification and inverse user frequency, but that default voting did not increase effectiveness. Lastly, experimentation with segmentation and clustering did not improve the performance of the recommenders; for this project it appears the most important factor in the success of a recommender beyond the algorithm itself is adequate quantity of training data.

References

- [1] Witten I. H. and Frank I. *Data Mining*, Morgan Kaufman Publishers, San Francisco, 2000.
- [2] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, John Riedl, GroupLens: an open architecture for collaborative filtering of netnews, Proceedings of the 1994 ACM conference on Computer supported cooperative work, p.175-186, October 22-26, 1994, Chapel Hill, North Carolina, United States
- [3] John S. Breese, David Heckerman and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence, pages 43-52, July 1998.
- [4] Paul Perry, A Minimalist Implementation of Collaborative Filtering, http://www.paulperry.net/notes/acf_sql.asp, November 2000
- [5] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, September 1994