Submitted by Brian Sa and Patrick Shih for CS229 Fall 2006

# K-MEANS FOR NETFLIX USER CLUSTERING

## I. Introduction and model

In the Netflix collaborative filtering problem, the goal is – given a set of training data $x = \{(u_i, m_i, t_i, r_i)\}$ consisting of a sample of prior movie ratings $r_i$ (an integer from 1 to 5) associated with user $u_i$, movie $m_i$, time $t_i$ – to accurately predict the rating that should be associated with a new point $(u, m)$. As part of a larger Stanford effort, we seek to use k-means to cluster users with similar movie preferences.

Intuitively, this means that users have intrinsic user types wherein all users of that type have aligned preferences across all movies, and furthermore that individual user vectors of that type reflect choosing from some unknown distribution specific to the user's class. The variance of ratings for any particular movie for users within a class can be interpreted as the natural variation of user ratings given known absolute preferences for a movie. As opposed to the Mixture of Multinomials group (c.f. project by Dimitris, Hau Jia, and Raylene), we make no assumptions about the underlying model other than the most basic premises: that users rate movies in a predictable way and that for a given user, different movie ratings are not all conditionally independent.

More formally, each user has a vector of ratings over all movies $x^* \in \Re^n$ which comprises the hypothetical ratings that the user would rate each of the 17770 movies in the Netflix library. The Netflix prize ratings database provides incomplete user vectors, $x^{(i)} \in \Re^d$ with $d \leq n$, that are the projections of the complete user vectors $x^{(i)*}$ onto some arbitrary lower dimension *d*. It is usually the case that $d << n$, with an average across the training set of $\overline{d} \approx 200$. Furthermore, d varies with each user *i* as it is not the case that all users have rated the same number of movies nor have all users rated the same set of movies.

Our original motivation in pursuing k-means was to perform linear regression within clusters, although the scope of clustering a dataset of this scale and sparsity proved enough of a challenge. K-means is also an ideal unsupervised method for classifying users in the vast Netflix data set because it converges extremely quickly in practice.

The method of k-means as applied to incomplete user vectors $x^{(i)}$'s is as follows:

1.) Initialize cluster centroids by one of two methods:
   a. Assign $\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(k)}$ to *k* randomly chosen $x^{(i)}$ 's.
   b. Or use the heuristic described in the k-means++ paper[†]
2.) Repeat until convergence:
   a. Let $c^{(i)} := \arg\max_j f_{score}(x^{(i)}, \mu^{(j)})$ for all *i*, where $f_{score}()$ is a scoring function described in more detail later.
   b. For each *j*, let $\mu_l^{(j)} = \dfrac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} \cdot \hat{x}_l^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j, \hat{x}_l^{(i)} \neq 0\}}$
   for *l* = 1, …, *n*; $\hat{x}_l^{(i)} \in \Re^n$ is $x^{(i)}$ projected from $d \mapsto n$, with all added dimensions padded with 0's, in effect ignoring incomparable ratings where either the centroid or the user is missing a rating for movie *l*.

The objective maximized in this case is:

$$J(c, \mu) = \sum_{i=1}^{m} f_{score}(x^{(i)}, \mu^{(c^{(i)})})$$

For comparison, the traditional method of applying k-means to a sparse data set is to fill in missing vector elements with default values. Furthermore, k-means is typically formalized as minimizing a distortion function that represents either the angle between vectors (cosine similarity) or the Euclidean distance:

$$J_{traditional}(c, \mu) = \sum_{i=1}^{m} \left\| x^{(i)} - \mu_{c^{(i)}} \right\|_2^2$$

[†] Arthur, David and Vassilvitskii, Sergei. "k-means++: The Advantages of Careful Seeding." To appear in SODA 2007.

```
Examining cluster 20...  Number of users: 44
  6:                                    *
  8:  *                                 *
 18:                            *
 28:         *                       *
 30: *    *  *        *     ** **    * *     *
 38:                                 *
 39:                                 *
 44:                                 *
 52:  *                                 *
 55:                                 *
 58:                                 *
 77:                                 *
 81:                                 *
 83:          *
 84:                  *              *
 97:                        *
108:              *         *   *
111:                        **  *    *
113:                                    *
118:                                    *
138:                                       *
143:          *                 *    *
148:   *  *                 *
166:     *
175:          *         *     * *    * **
176:                               *
181:   *
187:                            *    *
189:                                 *
191:    *      *    * * *   ** ** *  ***
196:                              *
197:    ** * ***                *    *    * *
199:                       *    *    *  *
209:           *
216:                                 *
241:         *          *       *  *
246:           *
248:                                  *
…
```

```
%(Size) is the number of users in the cluster
%(Span) is the number of ratings spanned by all users in the cluster
%(Overlapping) is the number of ratings rated on by two or more users
%(Outcasts) is the number of users who share no movies in common with other users
%Result of KMeans with k = 30 run on training set 1000x_smaller_training_1.dat
Cluster     Size        Span        Overlapping     Outcasts
0           18          1304        314             2
1           9           721         26              0
2           20          2094        962             0
3           14          730         127             0
4           30          3638        1879            0
5           1           995         0               1
6           12          1406        315             0
7           16          733         179             0
8           24          996         318             0
9           21          2626        1246            0
10          14          1396        334             0
11          16          1566        460             0
12          19          2738        1386            0
13          11          654         128             1
14          6           869         153             0
15          16          2174        967             0
16          4           691         17              0
17          36          3289        1760            0
18          15          2625        750             0
19          11          793         131             0
20          44          3168        1448            0
21          12          834         183             0
22          15          1338        374             0
23          10          786         213             0
24          6           939         107             0
25          36          2898        1709            0
26          3           1375        86              0
27          21          2720        1269            0
28          11          1899        467             0
29          9           740         138             0
```

**Figure 1**. *Left:* a graphical depiction of a cluster; movie id's are on the y-axis; users along the x-axis; *'s represent the presence of a rating for a particular user and movie. *Right:* a typical distribution of clusters; k=30; number of users = 480; scoring function used was MMP continuous (see below)

We will explain why this cannot be applied to the Netflix data set (without some tweaks) in the section on scoring functions.

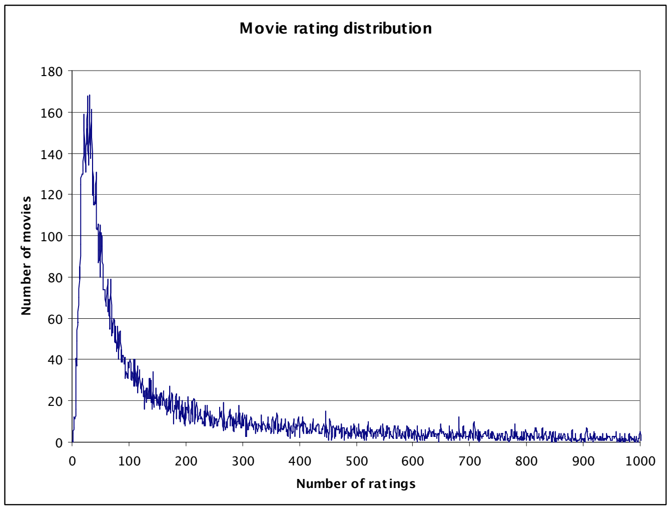

**Figure 2**. *Data set size.* 480,000 users. *Scoring function.* MMP continuous. *k*=10. *Notes.* Shown is the distribution of one cluster. The tail of the distribution is not shown.

## II. Parameters:

The parameters investigated in this project are the number of clusters *k*, heuristic initialization (*h*), and the scoring function $f_{score}()$.

### II.a. The effect of heuristic initialization (*h*)

The k-means algorithm is dependent on the initial centroids and as such is not guaranteed to discover the global optimum. That is, the quality of the clusters, as quantified by the objective function described earlier, is highly variable for different trials. A common method to overcome this is to run the algorithm multiple times with different initial centroids and return the best clustering found. Since clustering on the complete Netflix data set is computationally expensive, it is beneficial to start with clusters chosen by some heuristic so as to speed up convergence while also guaranteeing the quality of the resulting clusters.

Heuristic initialization as described in the k-means++ paper[†] was implemented for these purposes. At least in theory, carefully choosing initial centroid values has the advantages of reducing the number of iterations until convergence and of guaranteeing a clustering that is relatively consistent when repeated. The heuristic assigns the first centroid by choosing a user randomly. For each successive centroid, it chooses a user with probability proportional to its Euclidean distance to centroids that have already been assigned. The motivating idea is to choose centroids that are maximally distinguished from each other leading to more meaningful clusters on the first iteration. The results are compared with the standard method of initialization whereby *k* user vectors are selected randomly from the data set and used as the initial centroids.

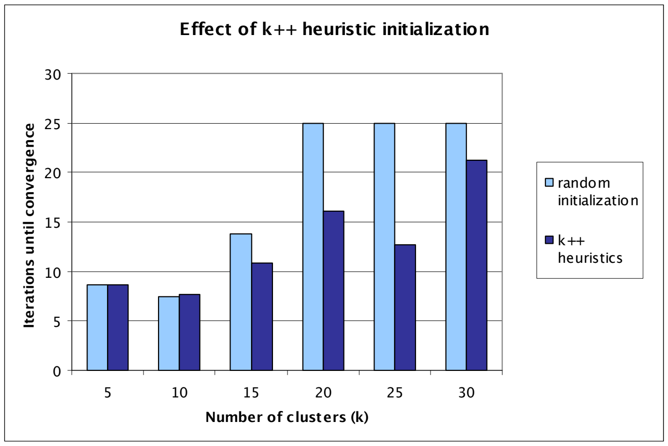

**Figure 3**. *Data set size.* 4800 users. *Scoring function*. MMP continuous. *Notes*. Number of iterations capped at 25. Each data point is the average over 5 trials run on different data sets.

According to the data (Figure 3, 4), k++ heuristic initialization decreases the number of iterations until convergence for all *k* and *m*. This implies that the heuristic starts the algorithm off with centroids closer to ideal than a random selection of users.

[†] Arthur, David and Vassilvitskii, Sergei. "k-means++: The Advantages of Careful Seeding." To appear in SODA 2007.

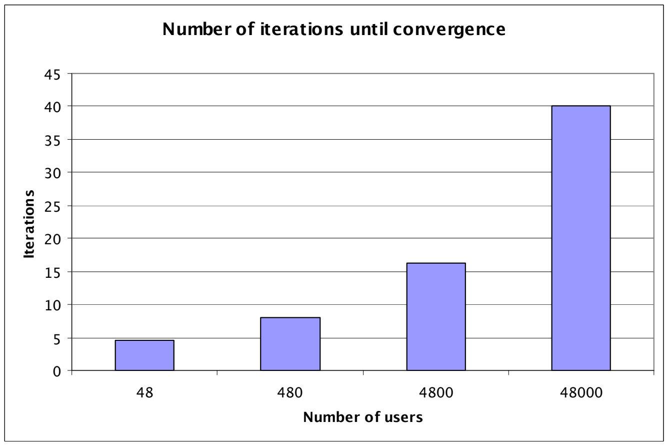

**Figure 4**. *Scoring function*. MMP continuous. *Notes*. Each data point is the average over 3 trials run on different data sets.

## II.b. Choosing the right scoring function $f_{score}()$

To preamble our discussion of scoring functions, we will start by explaining why the traditional method of filling incomplete vectors with default values fails in this application. Density (or sparsity, as is the case here) of the training data determines the ratio of default values to actual ratings. For the Netflix data set, which is approximately 1% dense, there are about 100 default values for every actual rating. Depending on the size of the training set *m* and the number of clusters *k*, the centroid vectors are filled with a significant proportion of default values unless *m* is large or *k* is small. This allows comparisons between two default values, which represent the maximum similarity attainable by either Euclidean distance or cosine similarity. Since scoring functions like Euclidean distance or cosine similarity make no distinction between faux values and real values, the resulting *signal to noise ratio* is very low. The number of such comparisons between default values is related to the density of the centroid, a measure of which can be found in a statistic we call the *span*. The *span* represents the number of movies in a cluster that have been rated by at least one person. Even if an attempt is made to improve the signal to noise ratio by decreasing the weight of default values, a significant problem still remains. It turns out that user vectors will always try to maximize the number of comparisons between default values since these achieve perfect similarity, i.e. result in a Euclidean distance of 0 or a cosine similarity of 1. Thus, unless the *span* covers virtually the entire set of movies, the clustering is utterly useless (see figure 1 for a typical clustering using k-means with default values).

The modification to k-means as implemented in this paper can be viewed as assigning a similarity score for each pair $(\hat{x}_l^{(i)}, \mu_l^{(j)})$. Non-comparable pairs – which will be defined in this context as any pair where either $\hat{x}_l^{(i)}$, $\mu_l^{(j)}$, or both are missing – are given a similarity of 0. On an intuitive level, this represents the default condition whereby no inferences about similarity can be drawn. Then for comparable pairs, the scoring function returns a value that rewards (a positive similarity) for rating differences within a certain threshold, and returns a value that penalizes (a negative similarity) for rating differences that exceed the threshold.

Four scoring functions were evaluated for clustering quality and secondarily for the root mean squared error (RMSE) of predictions made. The prediction for user *i* at movie *j*, given by $x_j^{(i)}*$, is calculated by using the closest centroid's rating, $\mu_j^{(c^{(i)})}$, if it exists, or resorting to an average rating calculated over the entire data set (an average adjusted by the average over the user and the average over the movie).

### II.b.i. Mismatch penalty (MMP) using a discrete scoring function

The problem of matching sparse user vectors to ideal clusters is analogous to that of sequence alignment for DNA or RNA. In the case of sequence alignment, a given pair of bases under consideration can be assigned a score via a scoring matrix, which contains a pre-enumerated grid of all permutations of the 4 possible bases with scores reflecting their similarity or affinity. In protein sequence alignment substitution matrices like PAM or BLOSUM serve the same purpose for scoring evolutionary sequence divergence. Drawing our inspiration from these methods, we wrote our own scoring function that takes as input the absolute value of the difference between a user rating and a centroid rating, discretizes it to integral values, and returns as output a number reflecting the similarity of the two ratings. The crucial insight we made was to treat the non-comparable case as a baseline from which to reward for small mismatches and penalize for large mismatches.

The motivation behind a discretized scoring function is to compensate for the granularity of user ratings. That is, if for example the centroid rating is 3.5, the user rating (an integral value) can be at best 3 or 4, yielding an absolute difference of 0.5. A scoring function that is discretized in the same increments as the user ratings returns consistent scores even if the centroid ratings fluctuate somewhat.

$$f_{MMPdiscrete}(\hat{x}_l^{(i)}, \mu_l^{(j)}) = \begin{cases} \left|\hat{x}_l^{(i)} - \mu_l^{(j)}\right| < 1 \rightarrow 1 \\ \left|\hat{x}_l^{(i)} - \mu_l^{(j)}\right| < 2 \rightarrow -5 \\ \left|\hat{x}_l^{(i)} - \mu_l^{(j)}\right| < 3 \rightarrow -10 \\ \left|\hat{x}_l^{(i)} - \mu_l^{(j)}\right| < 4 \rightarrow -20 \end{cases}$$

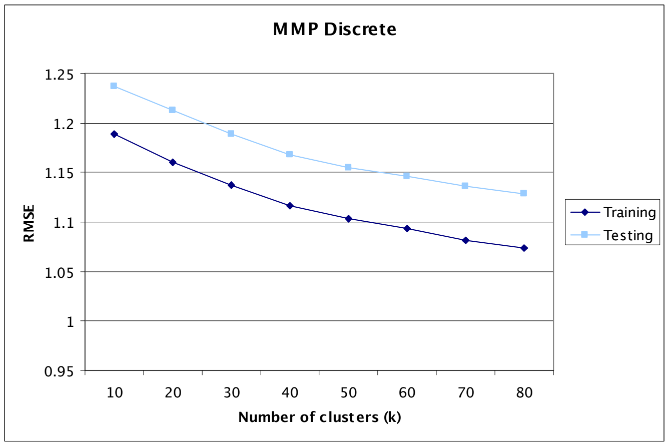

**Figure 5**. *Data set size.* 4800 users. *Scoring function.* MMP discrete. *Notes.* RMSE for predictions made both within and outside of span. Each data point is the average over 4 trials run on different data sets.

### II.b.ii. Mismatch penalty (MMP) using a continuous scoring function

Although discretization works quite well, it also has its disadvantages, namely that the arg-min of a step-function is a range, rather than a single value. Theoretically, this translates to a "looser" clustering, since a difference in ratings of as much as 1 is tolerated (or more accurately, rewarded). An ideal continuous scoring function would address this shortfall while maintaining the characteristics of the mismatch penalty scoring system.

$$f_{MMPcontinuous}(\hat{x}_l^{(i)}, \mu_l^{(j)}) = -(\left|\hat{x}_l^{(i)} - \mu_l^{(j)}\right| + 1)^2 + 2$$

This scoring function was constructed to fit the following specifications: 1) it must return 1 if the scores match exactly, 2) it must assign a negative penalty for any absolute difference exceeding 1.0, and 3) it must assign increasingly negative penalties for larger absolute differences. It is interesting to note that as originally constructed, $f_{MMPcontinuous}()$ was a cubic function; however, this proved too penalizing for large absolute differences in ratings and the algorithm did not converge.

The MMP Continuous scoring function was in fact able to achieve tighter clusters, as can be seen from its testing RMSE for *k*=50 (Figure 6). However, it was also sensitive to values of *k* that were too high.

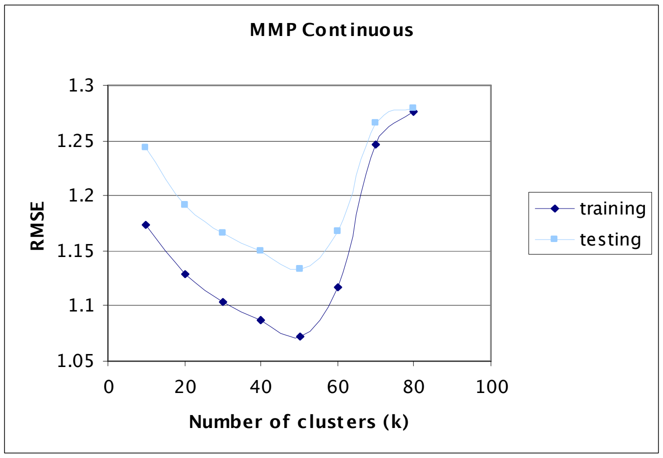

**Figure 6**. *Data set size.* 4800 users. *Scoring function.* MMP continuous. *Notes.* RMSE for predictions made both within and outside of span. Each data point is the average over 4 trials run on different data sets.

## III. Discussion

The ultimate goal of the Netflix prize is to minimize the RMSE for predictions. However, it is evident that k-means as the sole method for rating prediction, as compared to the current leading predictive algorithms, is limited at the current values of *k* and *m* (number of users). We expect that prediction will improve significantly as *k* and *m* increase (*k* > 1000 on all 480,000 users) because specificity increases proportionally to *k*. Further, the sparsity of the data places a constraint on the minimum size of a cluster. We must ensure a sufficient *span* to make valid predictions. Thus, increasing *k* requires a proportional increase in *m.*

If indeed a high value for *k* is required for optimal prediction, this implies that current values of *k* will have high bias, and this is borne out. The training error and generalization error on k-means run for constant *k*=10 and increasing *m* indicate that our algorithm is underfitting the data.

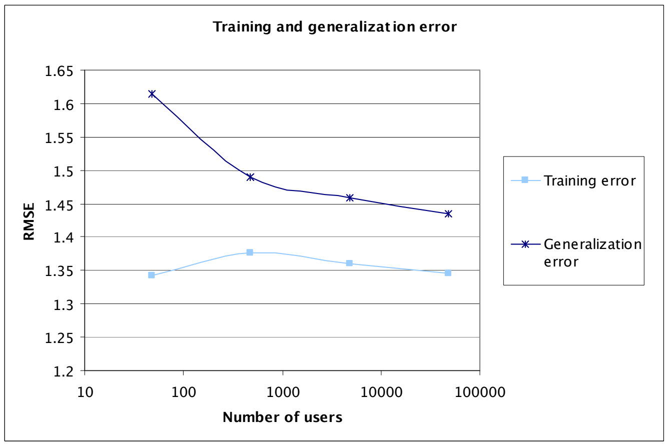

**Figure 7**. *Scoring function*. MMP continuous. *Notes*. RMSE calculated for predictions within the span of the centroid. Each data point is the average over 3 trials run on different data sets.

## IV. Future work

Continuing forward, a short term goal is to tune the parameters of the MMP Continuous scoring function for optimal clustering. Long term goals include proceeding with linear regression within clusters and utilizing information contained in movie content (such as from www.imdb.com) to improve performance.

## V. Acknowledgements

We would like to acknowledge Ted Hong and Dimitris Tsamis, our collaborators, and the entire Stanford Netflix prize team, led by Tom Do and Thuc Vu. Additional thanks to Tom Do for his assistance in writing a version of k-means utilizing MPI.