Michael Silber
Elior Tapiro

Data Analysis Mini Project: Report

1. Describe your suggested improvement to the algorithm. Explain why you chose it and why it should be helpful.

Our Improvement

In our case we chose not to improve the algorithm and instead decided to implement a separate algorithm completely which utilizes the K-Means algorithm. We will break the steps of our algorithm as follows:

1. We firstly load the subset data into our OOP environment: Each movie and user is represented by an object of its corresponding class. Also each movie will serve as a vector in a $|movieSubset| + 51$ dimensional space for our K-Means algorithm (Explained in detail below).

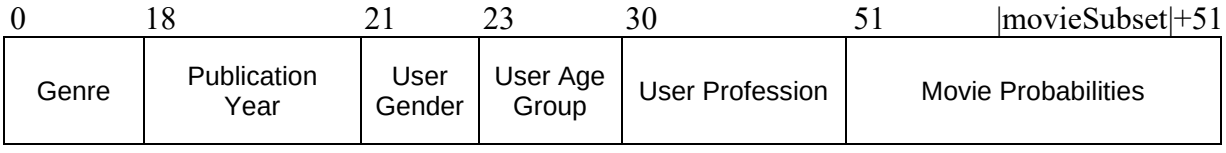
2. For $k \in \mathbb{N}, \frac{|movieSubset|}{4} \leq k \leq \frac{|movieSubset|*3}{4}$ do:

   A) For $j \in \mathbb{N}, 0 \leq j \leq \frac{|movieSubset|*2}{k}$ do:

      I. Choose k random movies from the subset that will serve as initial centroids in our K-Means algorithm
      II. Run K-Means until convergence (a small delta change in centroids between iterations chosen by us according to trials).
      III. If the clustering that we converged to has the smallest cost out of those we processed yet, update it as the current minimal clustering.

3. After the loop at step 2. finished we currently hold the minimal clustering out of all trials with the minimal k for this movie subset.

How do our movie vectors look like?



The above vector describes the general outline of movie vectors in our K-Means algorithm, we will further analyze the vector composition in this section. Generally our vectors are split into two main regions, the general movie parameters subsection which ranges through indices 0-50, and the movie probabilities subsection which is from index 51 onwards. The vector length is dependent on the subset size and will have a length of $|movieSubset| + 51$.

The general movie parameters subsection consists of all the information we have in our data base that in some shape or form helped us find patterns in our user data in order to cluster movies:

Genre: We delegated an index for each genre and if a movie applied to a certain genre then it would get a value of $\frac{1}{\#\ genres\ of\ movie}$ in order to normalize the values. All other genre coordinates remain with a value of 0.
Publication Year: We decided to split the publication years into 3 separate categories: Pre 1980, 1980-1990, and 1990-2000. This was because we didn't see a big improvement from this feature and any movie 20 years old was effectively grouped as an old movie in our eyes, then semi-old movies from the mid 80's, and recent movies from the past decade (dataset is up to the year 2000). A movie will get a value of 1 in only one coordinate according to the year category it fits into, the others remain 0.
User Gender: This two coordinate field consists of the relative population of users who watched the movie and are either male or female. E.g. the male coordinate consists of $\frac{\#males\ who\ watched}{\#users\ who\ watched}$.
User Age Group: This field consists of the age groups who watched the movie as split by the original data base. Each age group coordinate consists of the relative population of users who watched the movie from a specific age group. E.g. the first coordinate represents the under 18 age group and gets a value of $\frac{\#under\ 18\ who\ watched}{\#users\ who\ watched}$.
User Profession: This field consists of the relative population of users who watched the movie from a specific profession. The professions are grouped as they appear in the original data set. E.g. the farmer coordinate gets a value of $\frac{\#farmers\ who\ watched}{\#users\ who\ watched}$.

The second section in our movie vectors consists of the probabilities of the movie to appear with each of the other movies in the subset. Furthermore, assume were looking at the vector of a movie m_i then the j^{th} coordinate in its probabilities section will have the value of $p(m_i, m_j)$ as calculated by the formula in the assignment instructions. This implementation raises a very valid question, what should m_i have in its i^{th} coordinate? Well, we had no full proof implementation, but (after attempts to find the perfect value) we decided that this coordinate will receive the bigger of the two following values: $\max_j p(m_i, m_j)$ and $p(m_i)$. Although not full proof, this implementation does work excellently and has a lot of logic behind it. If $p(m_i)$ is bigger, then we prefer to distance it from other movies and put that value in its coordinate, which distances it because that value is bigger than the value any other movie will have in that coordinate. Otherwise we take the "best mate" which the movie m_i has and give it $p(m_i, m_j)$ in that coordinate, thereby giving both movies the same value in the same coordinate, which in turn gives them a distance of 0 from each other in that coordinate, hence making them closer to each other.

In addition to this description it is important to say that all values in our vectors are normalized between [0,1] and then given a multiplier factor according to the importance we saw for each field in the clustering process. Ideally we would learn these factors from some other learning algorithm, for example a neural net, but in the scope of this mini-project we decided them by practical trials. We also ended up using our own normalization factors but tried "Feature Scaling"[1], which obtained us similar results but with less of an option to tweak values.

Why we chose K-Means as opposed to other algorithms?

On a personal note, a first and undeniably influencing deciding factor in our decision process was that my partner and I took Dr. Meni Adler's Distributed Systetm Programming course this semester in which we tackled the clustering problem and were introduced to K-Means as part of the course. We couldn't help but see the parallels from the discussions in lecture to our mini-project which heavily influenced our thought process in choosing an algorithm.

[1] https://en.wikipedia.org/wiki/Feature_scaling

Now, the main reason we chose the K-Means algorithm is that we felt it is a natural solution to our clustering problem. In our data base we have many different parameters about the movies that we could use to influence our clustering decisions (e.g. movie probability, genre of the movie, movie publication year, to name a few), we felt that any good solution to this problem would take all of them into account. In other words we didn't see how any "proper" decision making process which wasn't a learning algorithm could take all of these parameters into account.

We did consider a graphical approach using the CLICK Algorithm[2] by Tel Aviv University's Roded Sharan and Ron Shamir, which offers a weighted clustering algorithm for graphs, where nodes would represent our movies and edges would represent the weighted probabilities that two movies are "mates" as they say in the paper. While CLICK offers a very viable solution to the problem graphically, it could only take the movie probabilities into account and wouldn't be a process which considered the other parameters we have at our hands. For this reason we decided not to implement CLICK, even though we must admit that it is a solution which must be checked out and may be more successful than ours for solving this problem. Either way, our desire to take all the different movie parameters into account and "learn" from them really narrowed the options down to machine learning.

That said, also in the machine learning realm we had many different options to chose from: neural nets, K-Means, and the Markov Clustering Algorithm (MCL). We cancelled out neural nets seeing as they not only require big data to train properly, but we also didn't have tagged data. MCL also seemed over-kill; there was no "friendly" way to use the model to describe our problem with all the different parameters as states and the clustering as outputs. Also we learned from Dr. Adler, who is a big supporter of the Markov model, that Markov models work best on big data and again that is not something we had at our disposal.

Contrary to the other options we discussed above, K-Means takes any parameter we choose to build our vectors with, and naturally we chose to build are vectors with every parameter we had at our disposal. The only complication was finding the right way to describe each parameter as a field in our vector, which essentially was the essence of the implementation. Of course not every parameter should have the same influence on the algorithm, and we could easily adjust this by choosing different normalization factors, multipliers, and a different distance function which the algorithm uses to compare vectors. All in all, we chose K-Means because the algorithm could represent our problem with all the features we wished to include in a wholistic vector space and find clusters according to the distance between vectors as we saw fit. Such a solution in our eyes was the most natural approach to the problem.

2. Give an example of a movie subset where your improvement indeed improved the results significantly, show the clustering before and after your improvement, along with the costs before and after.

The response to this question is appended to the zip in directory "Report q2".

[2] https://www.cs.tau.ac.il/~roded/click.pdf

3. Describe the main challenges you had in the implementation of this project, and how you handled them.

We faced two main challenges during the implementation of our project: a good initialization and choice of k, and a good distance function for our K-Means algorithm.

At first we implemented the K-Means algorithm to choose centroids as random points in the vector space, however often times these centroids were chosen badly, and resulted in huge fluctuations in convergence. Everything was up to chance: if a good set of centroids was initialized we may get a good clustering, and otherwise we may choose centroids in the middle of nowhere in our $|movieSubset| + 51$ dimensional space and get empty clusters, forcing other centroids to cluster more movies than intended. In addition we didn't know which k to choose for the algorithm, a major decision in an algorithm which is centered at clustering exactly k clusters. After researching and watching Andrew Ng's Machine Learning course on the K-Means algorithm we decided to take his advice and initialize our centroids with random vectors in our subset[3], thereby ensuring that centroids aren't picked in the middle of nowhere. In addition, Andrew Ng advised to repeat this process a number of times in order to ensure proper clustering, because K-Means varies greatly depending on the initialization, by possibly converging to "bad" local optima. By choosing the best convergence out of namely 100 runs for a given k, as Andrew advised, we greatly decreased our chances of converging at a bad local optimum. Finally, we considered running an algorithm prior to K-Means to determine our k, for example the correlation algorithm and choosing the k based on the number it found, but saw that the solution wasn't practical and varied greatly depending on which subset we ran on. We ended up deciding that there is no choice but to run the algorithm on different k values and pick an optimal for each run, but we did close the interval from $\frac{1}{4}|movieSubset| \leq k \leq \frac{3}{4}|movieSubset|$ because anything outside the interval really wasn't very likely practically. At this state we were running K-Means both a large number of times for each k and also on different values of k, so unsurprisingly we implemented this in a concurrent manner with a threadpool where every execution of the K-Means algorithm ran as a different thread task.

The second major challenge we faced dealt with the distance function used by the K-Means algorithm. While Euclidean distance worked splendidly for the general movie parameters (fields 0-50 in the vector) it just didn't make sense for the probabilities (and provided horrible results). The intuition came to us when analyzing the cost function. The cost for each cluster is measured with the probability inside $\log(\frac{1}{prob})$, so the larger the probabilities in our cluster the "exponentially smaller" our cost became. In other words, in the context of our assignment, we are rewarded for high probabilities inside a cluster much more than we are punished for low probabilities in a cluster. So it wouldn't make sense to treat higher and lower probabilities with the same weight. Our goal is to match movie vectors who have a few coordinate fields very close even if they have other probability coordinates miles apart, meaning Euclidean distance simply wouldn't cut it for the probabilities. Therefore we decided to change our distance function so that the general movie parameters were calculated as a Euclidean distance but the probabilities would be calculated by our own heuristic function (we couldn't find any proven material from outside sources). We tried many different combinations, such as inverses and exponential adjustments to the coordinate variance, but in the end a very simple heuristic completely out-performed any other configuration we ran testing this algorithm: If the distance is measured from some centroid c to some vector v, then for the probabilities section we would calculate for each coordinate i, $v[i](v[i] - c[i])^2$ instead of the Euclidean $(v[i] - c[i])^2$. This gives a weight to each coordinate variance between v and c according to the probability value of that coordinate in the vector v. In other words, this weight we added to the function added a decision of how important this coordinate's value is to this vector's distance. If $v[i]$ is relatively low to the vector then it receives a lower weight in the distance

[3] https://www.youtube.com/watch?v=PpH_hv55GNQ&t=327s

function because we “care less” about low probabilities. If $v[i]$ is relatively high to the vector then it receives a higher weight because it’s important for this vector to be clustered with movies with similar values in this coordinate. While we believe there probably exists a better distance function or heuristic, this improvement was a game changer in the cost performance of our algorithm.

4. Select 3 movie subsets that produce interesting results in your clustering algorithm. Each subset should include at least 20 movies. For these subsets, list the output clustering that you got with your algorithm, with and without your improvement. Also list in a table the cost of the clustering your algorithm obtained, before and after the improvement. You are allowed to share movie subset files with other teams so that each team can find interesting subsets for its own report. Explain for each movie subset what is interesting about the results for this case.

The subsets are appended to the root folder of the project as specified in the pdf, the clusters and the improvements in costs appear in files according to each subset and its respective number in the directory "Report q4".

Subset 1: The subset consisted of movies from varying genres, i.e. horror, children, comedy, and romance/drama. We hoped to find that movies of the same genre would be clustered together by our algorithm. The following are a few interesting clusters that were grouped:

Children/Adventure Cluster:
Cluster 10: 20.624503242606806
158 Casper, 48 Pocahontas, 2 Jumanji

These movies are classic children/adventure movies, we also have the following Disney cluster:

Disney Cluster:
Cluster 11: 11.14404713893452
1 Toy Story, 364 Lion King, The

It's interesting to see how Toy Story and Lion King were clustered together as Disney movies, but Pocahontas which is also Disney was in a different cluster. Finally, we also got two great clusters for the horror and drama genres.

Horror Cluster:
Cluster 16: 33.973356570095376
397 Fear, The, 220 Castle Freak, 152 Addiction, The, 177 Lord of Illusions

Drama Cluster:
Cluster 18: 33.29653721805582
31 Dangerous Minds, 46 How to Make an American Quilt, 85 Angels and Insects, 35 Carrington

Subset 2: This subset was formed of many movie series. We wanted to see how correlated the series were to each other according to our algorithm.

Police Academy Clusters:
Cluster 3: 22.118585320514377
2379 Police Academy 2: Their First Assignment, 2380 Police Academy 3: Back in Training, 2381 Police Academy 4: Citizens on Patrol

Cluster 7: 8.63703549704453
2382 Police Academy 5: Assignment: Miami Beach

Cluster 32: 8.869657713800743

2383 Police Academy 6: City Under Siege

Cluster 41: 7.612273265104456
2378 Police Academy

It's interesting to see how there is one main police academy cluster, and also 3 singleton clusters of different movies in the series. Why is it that 2,3,4 are "similar" according to our algorithm, but 1,5, and 6 are separate, maybe this could be due to the publication year? This phenomenon appeared with many of the series, including Star Wars, Star Trek, Teenage Mutant Ninja Turtles just to name a few. On the other hand, we have a really good clustering for the Friday the 13th series:

Friday the 13th Cluster:
Cluster 28: 27.84305766908584
1979 Friday the 13th Part VI: Jason Lives, 1974 Friday the 13th, 1975 Friday the 13th Part 2, 1977 Friday the 13th: The Final Chapter

Subset 3: Firstly the subset we chose was influenced by many movies not meeting the 10 rating threshold which reduced the variety. We chose this subset as foreign films (not American), and saw that movies from Europe were generally clustered together, for instance in the following cluster:

Italian/French Clusters:
Cluster 17: 23.384205184437143
583 Dear Diary (Caro Diario), 2696 Dinner Game, The (Le D�ner de cons), 1176 Double Life of Veronique, The (La Double Vie de V�ronique)
Cluster 19: 37.73148751573606
570 Slingshot, The (K�disbellan), 771 Vie est belle, La (Life is Rosey), 735 Cemetery Man (Dellamorte Dellamore), 2483 Day of the Beast, The (El D�a de la bestia)

Seems there is a correlation between "romantic" countries such as Italy and France. On the other hand there were a few anomaly clusters such as the following cluster which consisted of two chinese movies along with one french:

Anomaly Cluster:
Cluster 26: 24.3621544626432
2609 King of Masks, The (Bian Lian), 302 Queen Margot (La Reine Margot), 2621 Xiu Xiu: The Sent-Down Girl (Tian yu)

5. Generate 20 random movie subsets of size 100, by selecting 100 random (legal) movies from the data set for each subset. Run your algorithm on each of them with and without your improvement, and report in a table the cost before and after the improvement. Also report the average cost of each algorithm on these 20 subsets. Which cost is better?

The subsets used in this question are appended to the zip root directory. After processing the costs, we collected the data into the following table:

Index\Algorithm	Correlated algorithm	Our algorithm
1	847.9905996413852	795.290221062317
2	855.5072172215702	799.475453226435
3	892.1656263817578	827.5935604881993
4	848.536239208681	806.3511775371013
5	872.7443153344391	803.3192020908741
6	852.5146444793137	797.2258709667814
7	846.3407848617829	779.710570652102
8	863.057962108224	806.8996000291088
9	850.3114195481996	799.1145745058483
10	839.2528885426935	792.1367499149779
11	859.8022864723177	797.532006099611
12	837.2151712548778	785.8134717525429
13	837.9290180979236	784.5623193887293
14	851.6245536256902	801.3544647839099
15	859.4626677511372	812.7337556830455
16	832.2665136302064	781.2453187553995
17	829.8684459097834	788.363816152196
18	861.7869510359874	797.9506739096923
19	847.978264778487	809.8750792412606
20	859.6876816282437	805.6680320724365
Average	**852.3021625756351**	**798.6107959156284**

According to these 20 random subset runs, our algorithm reduces the clustering cost by an average of around 53.5.