CS757 FINAL PROJECT

AARON LEE

AHMED ALABDULLAH

Large-Scale Clustering

with Canopy and K-Means

1. **SUMMARY:**

Our project is an implementation of clustering, specifically the k-means clustering algorithm with canopy clustering as a preprocessing step.

Following initial data cleansing, the algorithm operates in two major steps. The first step is the creation of canopies, which are loosely-defined clusters whose members may be part of several canopies at once. Those canopies are then sub-clustered by k-means into the final clusters.

Canopies provide a major improvement over direct k-means, as it speeds up the processing for large datasets. Quickly drawing out the initial shape of clusters before k-means is employed with a more rigorous distance metric to outline the final clusters.

1. **DATASET:**

We used the MovieLens dataset, we utilized all three variations of the dataset but our primary focus was the 10M dataset.

**2.1 MUNGING:**

Some preprocessing was required to massage the data into a format suitable for the algorithm’s requirements.

As a first preprocessing step, movies that rarely feature in the ratings dataset were thrown out as their effect on the results was negligible.

**2.1.1: Remove Rarely Occurring Data**

After examining the distribution of ratings across movies. We determined a good cutoff point would be around the 35-45 percentile mark for the dataset (Figure 1).

**10K data set**

percentile[=5,](http://piratepad.net/ep/search?query=5,) counts[=1](http://piratepad.net/ep/search?query=1)

percentile[=10,](http://piratepad.net/ep/search?query=10,) counts[=2](http://piratepad.net/ep/search?query=2)

percentile[=15,](http://piratepad.net/ep/search?query=15,) counts[=3](http://piratepad.net/ep/search?query=3)

percentile[=20,](http://piratepad.net/ep/search?query=20,) counts[=5](http://piratepad.net/ep/search?query=5)

percentile[=25,](http://piratepad.net/ep/search?query=25,) counts[=6](http://piratepad.net/ep/search?query=6)

percentile[=30,](http://piratepad.net/ep/search?query=30,) counts[=9](http://piratepad.net/ep/search?query=9)

percentile[=35,](http://piratepad.net/ep/search?query=35,) counts[=12](http://piratepad.net/ep/search?query=12)

percentile[=40,](http://piratepad.net/ep/search?query=40,) counts[=16](http://piratepad.net/ep/search?query=16)

percentile[=45,](http://piratepad.net/ep/search?query=45,) counts[=21](http://piratepad.net/ep/search?query=21)

percentile[=50,](http://piratepad.net/ep/search?query=50,) counts[=27](http://piratepad.net/ep/search?query=27)

percentile[=55,](http://piratepad.net/ep/search?query=55,) counts[=34](http://piratepad.net/ep/search?query=34)

percentile[=60,](http://piratepad.net/ep/search?query=60,) counts[=43](http://piratepad.net/ep/search?query=43)

percentile[=65,](http://piratepad.net/ep/search?query=65,) counts[=51](http://piratepad.net/ep/search?query=51)

percentile[=70,](http://piratepad.net/ep/search?query=70,) counts[=65](http://piratepad.net/ep/search?query=65)

percentile[=75,](http://piratepad.net/ep/search?query=75,) counts[=80](http://piratepad.net/ep/search?query=80)

percentile[=80,](http://piratepad.net/ep/search?query=80,) counts[=100](http://piratepad.net/ep/search?query=100)

percentile[=85,](http://piratepad.net/ep/search?query=85,) counts[=127](http://piratepad.net/ep/search?query=127)

percentile[=90,](http://piratepad.net/ep/search?query=90,) counts[=169](http://piratepad.net/ep/search?query=169)

percentile[=95,](http://piratepad.net/ep/search?query=95,) counts[=227](http://piratepad.net/ep/search?query=227)

**1M data set**

percentile[=5,](http://piratepad.net/ep/search?query=5,) counts[=2](http://piratepad.net/ep/search?query=2)

percentile[=10,](http://piratepad.net/ep/search?query=10,) counts[=7](http://piratepad.net/ep/search?query=7)

percentile[=15,](http://piratepad.net/ep/search?query=15,) counts[=14](http://piratepad.net/ep/search?query=14)

percentile[=20,](http://piratepad.net/ep/search?query=20,) counts[=23](http://piratepad.net/ep/search?query=23)

percentile[=25,](http://piratepad.net/ep/search?query=25,) counts[=33](http://piratepad.net/ep/search?query=33)

percentile[=30,](http://piratepad.net/ep/search?query=30,) counts[=44](http://piratepad.net/ep/search?query=44)

percentile[=35,](http://piratepad.net/ep/search?query=35,) counts[=58](http://piratepad.net/ep/search?query=58)

percentile[=40,](http://piratepad.net/ep/search?query=40,) counts[=74](http://piratepad.net/ep/search?query=74)

percentile[=45,](http://piratepad.net/ep/search?query=45,) counts[=97](http://piratepad.net/ep/search?query=97) <----

percentile[=50,](http://piratepad.net/ep/search?query=50,) counts[=123](http://piratepad.net/ep/search?query=123)

percentile[=55,](http://piratepad.net/ep/search?query=55,) counts[=153](http://piratepad.net/ep/search?query=153)

percentile[=60,](http://piratepad.net/ep/search?query=60,) counts[=187](http://piratepad.net/ep/search?query=187)

percentile[=65,](http://piratepad.net/ep/search?query=65,) counts[=227](http://piratepad.net/ep/search?query=227)

percentile[=70,](http://piratepad.net/ep/search?query=70,) counts[=278](http://piratepad.net/ep/search?query=278)

percentile[=75,](http://piratepad.net/ep/search?query=75,) counts[=350](http://piratepad.net/ep/search?query=350)

percentile[=80,](http://piratepad.net/ep/search?query=80,) counts[=427](http://piratepad.net/ep/search?query=427)

percentile[=85,](http://piratepad.net/ep/search?query=85,) counts[=549](http://piratepad.net/ep/search?query=549)

percentile[=90,](http://piratepad.net/ep/search?query=90,) counts[=722](http://piratepad.net/ep/search?query=722)

percentile[=95,](http://piratepad.net/ep/search?query=95,) counts[=1045](http://piratepad.net/ep/search?query=1045)

**10M data set**

percentile[=5,](http://piratepad.net/ep/search?query=5,) counts[=5](http://piratepad.net/ep/search?query=5)

percentile[=10,](http://piratepad.net/ep/search?query=10,) counts[=11](http://piratepad.net/ep/search?query=11)

percentile[=15,](http://piratepad.net/ep/search?query=15,) counts[=17](http://piratepad.net/ep/search?query=17)

percentile[=20,](http://piratepad.net/ep/search?query=20,) counts[=26](http://piratepad.net/ep/search?query=26)

percentile[=25,](http://piratepad.net/ep/search?query=25,) counts[=34](http://piratepad.net/ep/search?query=34)

percentile[=30,](http://piratepad.net/ep/search?query=30,) counts[=44](http://piratepad.net/ep/search?query=44)

percentile[=35,](http://piratepad.net/ep/search?query=35,) counts[=59](http://piratepad.net/ep/search?query=59)

percentile[=40,](http://piratepad.net/ep/search?query=40,) counts[=78](http://piratepad.net/ep/search?query=78)

percentile[=45,](http://piratepad.net/ep/search?query=45,) counts[=101](http://piratepad.net/ep/search?query=101)

percentile[=50,](http://piratepad.net/ep/search?query=50,) counts[=135](http://piratepad.net/ep/search?query=135)

percentile[=55,](http://piratepad.net/ep/search?query=55,) counts[=177](http://piratepad.net/ep/search?query=177)

percentile[=60,](http://piratepad.net/ep/search?query=60,) counts[=232](http://piratepad.net/ep/search?query=232)

percentile[=65,](http://piratepad.net/ep/search?query=65,) counts[=320](http://piratepad.net/ep/search?query=320)

percentile[=70,](http://piratepad.net/ep/search?query=70,) counts[=440](http://piratepad.net/ep/search?query=440)

percentile[=75,](http://piratepad.net/ep/search?query=75,) counts[=622](http://piratepad.net/ep/search?query=622)

percentile[=80,](http://piratepad.net/ep/search?query=80,) counts[=920](http://piratepad.net/ep/search?query=920)

percentile[=85,](http://piratepad.net/ep/search?query=85,) counts[=1439](http://piratepad.net/ep/search?query=1439)

percentile[=90,](http://piratepad.net/ep/search?query=90,) counts[=2353](http://piratepad.net/ep/search?query=2353)

percentile[=95,](http://piratepad.net/ep/search?query=95,) counts[=4378](http://piratepad.net/ep/search?query=4378)

*Figure 1: Distribution of movies in the ratings across the MovieLens dataset*

*example: in the 10KData set, 40% of movies have at most 16 ratings.*

This produced a significant decrease in the number of movies, across all three datasets at least 35% of movies were discarded, but this was a mere blip as far as the ratings data is concerned as the 10M set lost less than 1% of its ratings, the loss for the 1M and 100K datasets was approximately 4%.

In the accompanying code description, this result is achieved by the file RemoveRareRatings.java, which does not run as a Map-Reduce job but as a simple Java program.

**2.1.2: DATA MASSAGING**

Next, the data was formatted into a structure that would prove useful for the algorithm.

So far, the data is still in the original format of the MovieLens dataset:

*userID::movieID::rating::timestamp*

Figure 2.1: MovieLens dataset format

We convert this format using the following process:

We omit all timestamp information as it’s not useful to us.

We aggregate the ratings of one user into a single line, this will be a single “data point” in our data set.

*userID (tab separator) ratings-vector*

Figure 2.2: Massaged input

The ratings vector for each user has the following structure:

*movieID1:rating1, movieID2:rating2, movieID3:rating3*

We also normalize the data from a 5-star scale into a 10-star scale.

The code responsible for this part of our program is the Munger.java class. Which runs as a Map-Reduce job.

* + 1. **DATA NORMALIZATION:**

In the MovieLens dataset, the ratings use a 5-star scale, while the 10K and 1M dataset use increments of 1-star, the 10M dataset uses 0.5 increments. Because we are going to employ Jaccard Distance on bags as our first distance metric, which expects whole numbers. The 5-star scale had to be normalized into a 10-star scale. Every rating is simply multiplied by 2 to get the normalized output. This process is done in the Munger.java class.

1. **CLUSTERING**

In additional to smaller pre- and post- processing steps, the clustering algorithm operates in two major steps:

1. Canopy Clustering: An initial quick clustering of the datasets, utilizing a cheaper distance metric.
2. K-Means Clustering: The canopies are then clustered into the final clusters using a more expensive metric.
   1. **CANOPY CLUSTERING:**

In large datasets where it may not be feasible to process the data at once using a distance metric that is slow and expensive, canopy clustering is used to speed up the clustering process by determining an initial outline of the final clusters using a cheaper distance metric.

The difference between a canopy and a cluster is that data points may belong to different canopies at the same time. That is, canopies may overlap. Canopy clustering is useful in cases where the number of initial cluster may be large, the dataset is highly dimensional in its feature set, and the dataset is large. [1]

The way our distributed implementation of the algorithm works is as follows:

Given a subset of the data:

Pick a point, *c,* at random as a canopy centroid

Calculate similarity using Jaccard distance on bags with all other points

If the similarity exceeds the threshold **T1,** or the loose similarity, then the point is included in the canopy centered at *c. It is still considered for inclusion in future canopies.*

Only if the similarity exceeds the tighter threshold **T2,** or the tight similarity, then the point is removed from the original set and is no longer considered in other canopies.

This effectively means that canopies may overlap.

**3.1.1 CANOPY DISTANCE METRIC (JACCARD DISTANCE ON BAGS)**

We considered several metrics that may work for our high-dimensional dataset in order to avert falling under the curse of dimensionality, after considering Eucledian and Cosine distances, we decided to go with the *Jaccard Distance on Bags* as a cheap metric (we still use cosine distance as a distance metric for K-Means):

Jaccard Distance:

Jaccard Similarity between sets is the size of their intersection divided by the size of their union, and the Jaccard distance is 1-SIM where SIM is the Jaccard similarity between two sets.

This will work well if our featureset is of a binary nature (0/1, true/false) but since each feature may have one of several values (10, after normalization) we opted to employ Jaccard bags instead.

Suppose a user x has given a rating of 3.5 for movie y in the MovieLens dataset.

3.5 becomes 7 post normalization, and so for the Jaccard set of user x, we include the movieID y 7 times.

We then calculate the Jaccard distance accordingly (1 –Jaccrd Similarity between Bags)

[INSERT CANOPY RESULTS HERE]

[EXPLAIN WHAT HAPPENS WITH SCALING]

**3.1.2 Picking The Right Value For K:**

Using Canopy Clustering presents us with a unique advantage. With traditional k-means clustering, k has to be initially guessed with heuristics. Canopy clustering automatically clusters around k canopies, thus giving us an initial lower bound of k implicitly as a byproduct.

**3.1.2 Canopy Implementation:**

The canopy implementation primarily resides in Step1.java in our source code.

At the map phase, an in-memory structure is built to store an associative array of key-value pairs where the key is the userID and the value is its detailed feature vector.

Once the entire map is built, canopies are emitted.