Lab 3 – K-means Clustering of Netflix Data

CSE 490h – Winter 2007

Assigned - 4/4/18 **Due -** 11:59 pm on Tuesday, 5/1/07 (the two week due date is intentional)
Partners - Sure.
Reading - **Canopy Clustering -** www.kamalnigam.com/papers/canopy-kdd00.pdf
K-means Clustering - en.wikipedia.org/wiki/K-means_algorithm

Goal - Cluster the Netflix movies using K-means clustering. We're given a set of movies and a list of which review which user has given to which movies. We want to output four hundred or so sets of related movies. Getting there should impart on you a respectable degree of map-reduce fu.

Input - /user/jhebert/netflixpieces
Data is one entry per line of the form "movieId, userId, rating, dateRated."

Overview -
K-means clustering is one of the simplest clustering algorithms, called k-means because we iteratively improve our partition of the data into k sets. We choose k initial points and mark each as a center point for one of the k sets. Then for every item in the data set we mark which of the k sets it is closest too. We then find the average center of each set, by averaging the points which are closest to the set. With the new set of centers we repeat the algorithm. BUT AHHHH!!! THE COMPLEXITY! Initially this is a pretty simple process, for each iteration we compare each point to each possible center. Predictably, this doesn't scale very well so we will make use of canopy clustering to reduce the number of distance comparisons we need to make. SWEET. As discussed in the reading we create a set of overlapping canopies wherein each data item is a member of at least one canopy. We then revise our original clustering algorithm – rather than comparing each data point to each k-set-center, we only compare it to the centers that are located in the same canopy. This vastly reduces the amount of computation involved, the caveat is that if two points never occur in the same canopy they will not occur in the same final cluster.

Applications - news.google.com, clusty.com, Amazon.com related products, etc

Suggested Steps (each is a MapReduce) -
- Data prep. Get the data into a format you can work with. You can do this yourself, alternatively we've created a sequenceFile of movieVectors "key: movieId value: <userID;rating> <userID2;rating2>..." that you can use. (easy)
- Canopy selection. Use the algorithm from the paper. (non - trivial)
- Mark data set by canopy; create a second data set in which movie vectors are also marked by canopy. (easy)
- K-means iteration. Use the canopy centers as the source of initial K-means. (tricky)
- Viewer. (easy)

Notes -
1. Understand the whole process before you start coding. Read the canopy clustering paper and any relevant wikipedia articles before asking questions.
2. Canopy selection requires a simple distance function; we used the number of user IDs in common. It also uses close and far distance thresholds; we used 8 and 2.
3. You use a cheap distance metric for the canopy clustering and a more advanced metric for the K-means clustering. We used vector cosine distance for our more advanced distance metric (search for it).

4. For the k-means-iter map-reduce we found it easiest to map over the entire data set and load the current k-mean-set-centers into memory during the configure() call to the mapper.
5. In the k-means-inter reduce where the new centers are found by averaging the vectors we found it necessary to minimize the number of features per vector, allowing them to better fit into memory for the next map. To do this we picked the top hundred thousand occurring userIDs per movie and only output the average of those user ratings.
6. Be sure to use reporter.SetStatus() to output basic debug information, such as how many k-centers were loaded into memory, the number of points mapped to so far, or the number of features per vector. Mmmmmmmmm status.