CSCI-GA.3033-016. Big Data and ML Systems - Fall 2017

Assignment 1b

October 16, 2017

Submission: Please submit in electronic format (LaTeXor MS Word or plain text or scanned handwritten solutions). Please include all supporting files along with the solutions in a compressed archive (tar, zip, etc.). Ensure that your responses are adequate and clear. Please note that you are free to use Python, Scala or Java when programming in Spark. You may also use one of Python or Java if you are programming in Hadoop (for those problems asked). Please list any external references that you may have used to help you solve any of the problems. Finally, over and above what the write-ups for each problem asks, if there are additional assumptions that you made, or additional points to be mentioned, please mention them.

Credits to the contributors to some of the problems and datasets in the CS246 (Mining Massive Data Sets) course at Stanford in the Winter 2017 quarter.

1. Write a program in **Spark** or **Hadoop** that implements a simple "People You Might Know" social network friendship recommendation algorithm. The key idea is that if two people have a lot of mutual friends, then the system should recommend they connect with each other.

   Input: Download the input file from this link.

   The input file contains the adjacency list and has multiple lines in the following format:

   `<User><TAB><Friends>`

   Here, `<User>` is a unique integer ID corresponding to a unique user and `<Friends>` is a comma-separated list of unique IDs corresponding to the friends of the user with the unique ID `<User>`. Note that the friendships are mutual (i.e., the edges are undirected): if A is friend with B then B is also friend with A. The data provided is consistent with the rule that as there is an explicit entry for each side of each edge.

   Algorithm: Let us use a simple algorithm such that, for each user U, the algorithm recommends $N = 10$ users who are not already friends with U, but have the largest number of mutual friends in common with U.

   Output: The output should contain one line per user in the following format:

   `<User><TAB><Recommendations>`

   where `<User>` is a unique ID corresponding to a user and `Recommendations` is a comma-separated list of unique IDs corresponding to the algorithm's recommendation of people that `<User>` might know, ordered by decreasing number of mutual friends. Even if a user has fewer than 10 second-degree friends, output all of them in decreasing order of the number of mutual friends. If a user has no friends, you can provide an empty list of recommendations. If there are multiple users with the same number of mutual friends, ties are broken by ordering them in a numerically ascending order of their user IDs.

   Also, please provide a description of how you are going to use MapReduce jobs to solve this problem. Don't write more than 3-4 sentences for this: we only want a high-level description of your strategy to tackle this problem.

   Note: It is possible to solve this question with a single MapReduce job. But if your solution requires multiple MapReduce jobs, then that's fine too.

   What to submit:

a) Source code

b) Short write-up of how you solved the problem

c) Compute and include the recommendations for the users with following user IDs: 8941, 9020, 9021, 9993.

2. **PageRank computation**: Write a program in **Spark** to compute the PageRanks for each webpage in the following sample web graph – `http://snap.stanford.edu/class/cs246-data/graph.txt`.

   Input: Each number in the file represents a node and each line represents a directed edge going from the first node to the second.

   `<Vertex 1><TAB><Vertex 2>`

   What to submit:

   a) Short description of map and reduce functions that you used.

   b) Output file for the graph containing the list of all node IDs and their corresponding PageRank scores, in a simple format as below.

      `<Vertex ID><TAB><PageRank>`.

      Compute both simple PageRank as well as modified PageRank assuming dead ends and spider traps. Use $\beta = 0.8$.

3. **Document Similarity and Clustering**: In this problem, you will use the ideas on clustering and computing similarities that you learnt in Chapter 3 in the book, including k-Means, *shingling* and *Jaccard similarity*. This is to be implemented in **Spark**.

   Input: Download the dataset from here. The dataset contains a list of short news articles in a CSV format, where each row contains the columns – the short article, date of publication, title and category.

   k-Means: The iterative k-Means algorithm is presented below.

Algorithm 1 Iterative k-Means Algorithm

```
1: procedure ITERATIVE k-MEANS
2:     Select k points as initial centroids of the k clusters.
3:     for iterations := 1 to MAX_ITER do
4:         for each point p in the dataset do
5:             Assign point p to the cluster with the closest centroid
6:         end for
7:         for each cluster c do
8:             Recompute the centroid of c as the mean of all the data points assigned to c
9:         end for
10:    end for
11: end procedure
```

   map-reduce: Implement iterative k-Means using the map-reduce paradigm, where a single step of map-reduce completes one iteration of the k-Means algorithm. So, to run k-Means for i iterations, you will have to run a sequence of i map-reduce jobs.

   Number of clusters: Remember that k-Means by itself does not tell you the number of clusters actually present in the data. It performs a clustering, in an unsupervised fashion, given the number of clusters it is expected to find (through our input k). So in order to determine the best number of clusters for the data, we need to examine inter-cluster distance and intra-cluster distance for a range of k values, or equivalently, intra-cluster variance and total variance. Generally, the "knee" or the "elbow" in the plot of total intra-cluster variance (or distance) vs the value of k indicates the best number of clusters. Read this on how to use the elbow method to determine the optimal

number of clusters. The quantity SSE is the same as intra-cluster variance viz. the sum of squared difference between each datapoint and the centroid of the cluster it belongs to.

Initialization: The manner in which the k centroids are initialized to begin with sometimes makes a difference. You can randomly choose any k of all the vectors as initial centroids, to begin with. There are other ways you can initialize – refer to section 7.3.2 in the book for some ideas. You are free to use other methods of your own or from external references (please cite references; see below).

Tasks: These are the tasks you are required to do for this problem.

- Before performing k-Means, you will need to convert the documents into vectors of equal size. For this, use *minhash signatures*. You can experiment with the size of signatures. A size of 100 may be a good idea. For shingles, you may use any type of shingles – character or word. If character, then 5-character shingle may be a good starting point. If word, then check out section 3.2.4 in the book for some ideas for news articles. Implement minhashing using map and reduce functions for additional credit.
- Run k-Means on the signatures to do the clustering (make k a parameter in your code i.e. don't hardcode it). Use Euclidean distance measure. Try a sequence of k values (say, from 1 to 10) and determine the best number of clusters using the elbow method.
- Now, once the clustering is done and you have determined the best number of clusters, manually check the documents in each cluster to see if the above sequence of computations have correctly clustered the documents according to similarity.

Note: It is expected that you use map and reduce functions to implement the k-Means algorithm. Using map and reduce functions for computing minhashes is optional, and will earn you additional credit.

What to submit:

a) The elbow plot that was created to determine optimal number of cluster for k-Means.

b) Short write-up detailing the assumptions and flow of computations in your code. Also briefly state, from your manual observation, the performance of your clustering. In particular, highlight any bad examples (most likely as a result of using shortened minhash signatures). Also mention the rationale behind your choices, wherever you have made them, such as (but not limited to) the shingles, k in k-Means and size of signatures.

c) List of references (if any)