CS 5683: Big Data Analytics

Project-3: *k*-Means Clustering in Spark

Total Points: 100 (10% toward final)

Due date: Oct 16, 2020 at 11:59pm

In this project, we will analyze k-means clustering algorithms with multiple centroid initializations and distance measures using Apache Spark. Students should not use clustering libraries available in Spark MLlib for this project. This is a group project. Students can form groups of maximum size 2. However, students can choose to work independently also.

This project will help to understand the fine details of implementing clustering algorithms with Spark. In addition, this project will help to understand the impact of multiple cluster initialization techniques and distance measures. Let's consider that we have a set X of $\boldsymbol{n}$ data points in a ***d*-dimensional space** $\mathbb{R}^d$. Given the number of clusters $\boldsymbol{k}$, we need to define the following:

1. Initial $\boldsymbol{k}$ cluster centroids $\boldsymbol{C}$
2. Distance metrics
3. Cost functions that our algorithm minimize

Initialize k-cluster centroids:

We will analyze 2 types of k-cluster centroid initializations.

1. ***k*-means**: Randomly select $\boldsymbol{k}$ data points as initial clusters
2. ***k*-means++**: We will use the following intuitive approach to pick random $\boldsymbol{k}$ farthest points.
   - **a.** Pick the first random data point and fix it as the first cluster centroid $\boldsymbol{c_1}$
   - **b.** For each remaining data point $\boldsymbol{x_i}$ measure their *squared Euclidean distance* $\boldsymbol{D_i}$ to the existing cluster centroids with following equation
     $$\boldsymbol{D_i = max_{(j:1\rightarrow m)} \left|\left|x_i - c_j\right|\right|^2}$$
     where $\boldsymbol{m}$ is the number of clusters. As given in the equation, if there are multiple cluster centroid, we have to use the farthest cluster centroid
   - **c.** Choose the new cluster centroid in random with probability = $\boldsymbol{D_i}$
   - **d.** Repeat steps (**b**) and (**c**) until we get $\boldsymbol{k}$ cluster centroids

Distance metrics and Cost functions:

Euclidean distance: Given two data points A and B in a d dimensional space such that $A = [a_1,a_2,\ldots,a_d]$ and $B = [b_1,b_2,\ldots,b_d]$, the Euclidean distance between A and B is defined as:

$$||a - b|| = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2}$$

The corresponding cost function Φ that is minimized when we assign data points to clusters using the Euclidean distance metric is given by:

$$\Phi = \sum_{x \in X} \min_{(c \in C)} ||x - c||^2$$

In this cost function we square the distance measure. This is intentional, as the squared Euclidean distance is guaranteed to minimize

Manhattan distance: Given two data points *A* and *B* in a *d* dimensional space such that $A = [a_1, a_2, \ldots, a_d]$ and $B = [b_1, b_2, \ldots, b_d]$, the Manhattan distance between *A* and *B* is defined as:

$$||a - b|| = \sqrt{\sum_{i=1}^{d} |a_i - b_i|}$$

The corresponding cost function Ψ that is minimized when we assign data points to clusters using the Manhattan distance metric is given by:

$$\Psi = \sum_{x \in X} \min_{(c \in C)} |x - c|$$

Iterative *k*-Means Algorithm: We looked at the basic *k*-Means clustering algorithm in the lecture which is as follows:

1. Initialize *k* random centroids (clusters)
2. Assign each data point to the nearest centroid based on Euclidean distance
3. Re-compute centroids, based on assignment of points, as the mean of points
4. Repeat steps 2 and 3 until convergence
5. Compute the performance of the algorithm with a cost function Φ or Ψ

Project Dataset: We will utilize a dataset with 4601 rows and 58 columns. Each row is a document represented as a 58 dimensional vector of features. Each component in the vector represents the importance of a word in the document.

Iterative *k*-Means Algorithm in Spark: All the above mentioned steps can be parallelized with Apache Spark

TO-DO:

1. Create 2 Spark programs – **kmeans.py** and **kmeans++.py** (Of course, it can be **.ipynb** also)
    a. **kmeans.py** – Clustering algorithm with random centroid initialization
    b. **kmeans++.py** – Clustering algorithm k-means++ style of centroid initialization
2. Your programs must do the following:
    a. Programs must have parameters for the number of clusters (***k***) and the number of algorithm iterations (***I***). For this project, set ***k=10*** and ***I=20***
    b. You can parallelize the data when reading the data file with Spark. This will parallelize your RDD operations
    c. Initialize *k* centroids with Spark:
        i. For **kmeans(.py),** this is possible with just 1 iteration of the data
        ii. For **kmeans++(.py)**, this is possible with ***k*** **(**10**)** iterations of the data
        iii. [HINT-1]: You can collect and broadcast centroids *but not the dataset*
        iv. [HINT-2]: Since k-means algorithms pick random centroids each time you execute your program, save your initial ***k*** centroids in a file before going to the next step
    d. Clustering with Spark:
        i. For both **kmeans(.py)** and **kmeans++(.py)**, use both Euclidean distance and Manhattan distance metrics to perform clustering
        ii. Your algorithms should run for ***I*** **(20)** iterations to create ***k*** **(10)** clusters. ***No need to check for convergence***
        iii. [NOTE]: Do not forget to update cluster(s) centroids after each iteration
    e. Clustering algorithm performance:
        i. Compute cost functions Φ or Ψ at every iteration in both **kmeans(.py)** and **kmeans++(.py)**. [HINT]: You do not need a separate iteration to compute these cost functions. *You can merge this step with the Clustering with Spark (d) step itself*
        ii. Plot with a line graph: Plot the cost functions Φ(i) and Ψ(i) as the function of number of iterations ***I****=1,2….20* for both **kmeans(.py)** and **kmeans++(.py)**
        [NOTE]: Your plots must include all necessary details like axes names, title, and legend
        iii. Calculate the percentage change in both Φ and Ψ cost functions after 10 iterations of **kmeans(.py)** and **kmeans++(.py)**. Explain your reasoning. [HINT]: The percentage refers to (cost[***I=1***]-cost[***I=10***])/cost[***I=1***]
        iv. Compare and reason the performance of **kmeans(.py)** and **kmeans++(.py)** with the same percentage change as above
        v. [HINT]: Multiple resources are available for line plots in Python using *matplotlib* library. Example:
        https://matplotlib.org/tutorials/introductory/pyplot.html
        https://www.w3resource.com/graphics/matplotlib/basic/matplotlib-basic-exercise-5.php
        Students are welcome to explore other libraries like *seaborn* also.

Submission requirements:

1. Student programs ***must*** be in ***PySpark***
2. Programs should be well documented – the grader should understand program modules clearly with your documentation
3. Submit only the following four files:
    a. **kmeans.py** and **kmeans++.py** [YES, you can submit notebook (.ipynb) files also]. Include team member names in both files
    b. **k-centroids** from both **kmeans.py** and **kmeans++.py**
4. Include the plots and their reasoning in appropriate programs
5. To compare performance of two programs, include your reasoning in **kmeans++.py**
6. [IMPORTANT] Since this is a group project, give the participation report (which team member contributed to which module in both programs) at the end of **kmeans++.py**

Grading Rubric:

1. Centroid initialization – Total: 15 points
    a. Random initialization: 5 points
    b. k-means++: 10 points
2. Clustering: 40 points
    a. Euclidean distance metric formulation: 5 points
    b. Manhattan distance metric formulation: 5 points
    c. Programming: 20 points
    d. Cost functions: 10 points (5 points each)
3. Algorithms performance: (25 points)
    a. Plots: 10 points (5 points each)
    b. Compare performance of cost functions: 10 points (5 points each)
    c. Compare performance of two algorithms: 5 points
4. Submission requirements: (15 points)
    a. Documentation and program file organization: 7 points
    b. Team work: 8 points

Simple reminder:

1. Since this is a group project, it requires planning to split the work among team members. So start early

2. You can discuss the logic and implementation details with your friends. But, you should write your own program and documentation. *We will not grade your work if we suspect cheating.* Give references wherever necessary, if you are using a logic from any internet sources. Start your work early as this project involve many sub-tasks which require some logical thinking