MIE 524 Data Mining
Assignment 4

This assignment allows students to implement PageRank and k-means on Spark.

- Programming language: Python (Google Colab Environment)
- Due Date: Posted in Syllabus

Marking scheme and requirements: Full marks will be given for (1) working, readable, reasonably efficient, documented code that achieves the assignment goals, (2) for providing appropriate answers to the questions in a Python notebook (named `MIE524_A4.ipynb`) committed, together with any associated output files, to the student's assignment repository, and (3) attendance in the post-assignment lab quiz.

Please note the plagiarism policy in the syllabus. If you borrow or modify any multiline snippets of code from the web, you are required to cite the URL in a comment above the code that you used. You do not need to cite tutorials or reference content that demonstrate how to use packages – you should certainly be making use of such content.

What/how to submit your work:

1. All your code should be included in a notebook named `MIE524_A4.ipynb` that is provided in the cloned assignment repository.
2. Commit and push your work to your GitHub repository in order to submit it. Your last commit and push before the assignment deadline will be considered to be your submission. You can check your repository online to make sure that all required files have actually been committed and pushed to your repository.
3. Your committed notebook should include all the computed outputs by first running it from top to bottom on Google Colab and then exporting the notebook.
4. A link to create a personal repository for this assignment is posted on Quercus.

This assignment has 4 points in total and the point allocation is shown below:

- Q1): 2 points
- Q2): 2 points

Note: Partial points may be given for partial answers. Points will be deducted for missing or incomplete answers. Points could also be deducted for styling.

Attribution: This assignment is inspired by contents from Stanford's CS247.

Q1 - PageRank

Note: The general computation should be done in **Spark**, and you may also include numpy operations whenever needed.

Assume we have a directed graph $G = (V, E)$ with n nodes (numbered $1, 2, \cdots, n$) and m edges, all nodes have positive out-degree, and $M = [M_{ji}]_{n\times n}$ is a an $n \times n$ matrix such that for any $i, j \in [\![1, n]\!]$:

$$M_{ji} = \begin{cases} \frac{1}{\deg(i)} & \text{if } (i \leftarrow j) \in E, \\ 0 & \text{otherwise} \end{cases}$$

Here, $\deg(i)$ is the number of outgoing edges of node i in G. If there are multiple edges in the same direction between two nodes, treat them as a single edge. By the definition of PageRank, assuming $1-\beta$ to be the teleport probability, and denoting the PageRank vector by the column vector $\mathbf{r}$, we have the following equation:

$$\mathbf{r} = \frac{1-\beta}{n}\mathbf{1} + \beta\mathbf{M}\mathbf{r},$$

Based on this equation, the iterative procedure to compute PageRank works as follows:

1. Initialize: $\mathbf{r}^{(0)} = \frac{1}{n}\mathbf{1}$
2. For i from 1 to k, iterate $\mathbf{r}^{(i)} = \frac{1-\beta}{n}\mathbf{1} + \beta\mathbf{M}\mathbf{r}^{(i-1)}$

Task

You will be experimenting with randomly generated graphs (assume the graph has no dead-ends) provided in the `data` folder. There are $n = 100$ nodes in the small graph and 1000 nodes in the full graph, and $m = 8192$ edges, 1000 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. It is easy to see that the existence of such a cycle ensures that there are no dead ends in the graph. There may be multiple directed edges between a pair of nodes, and your solution should treat them as the same edge. The first column in `q1-graph-full.txt` refers to the source node, and the second column refers to the destination node.

Implementation hint: You may choose to store the PageRank vector $\mathbf{r}$ either in memory or as an RDD. Only the matrix $\mathbf{M}$ of edges is too large to store in memory, and you are allowed to store matrix $\mathbf{M}$ in an RDD. e.g. `data = sc.textFile("q1-graph-full.txt")`. On an actual cluster, an RDD is partitioned across the nodes of the cluster. you can then

run `M = data.collect()` which fetches the entire RDD to a single machine at the driver node and stores it as an array locally.

a) Complete the `my_PageRank` class to run the process for PageRank in Spark for 40 iterations (with $\beta = 0.8$) and obtain a PageRank vector $\mathbf{r}$. The matrix $\mathbf{M}$ can be large and should be processed as an RDD in your solution.

b) Compute the PageRank scores and list the top 5 node ids with the highest PageRank scores.

c) List the bottom 5 ids with the lowest PageRank scores.

For a sanity check, we have provided a smaller graph (`q1-graph-small.txt`). In that dataset, the top node has id 53 with value 0.036.

Q2 - K-means over Spark

Note: This problem should be implemented in Spark. You should not use the Spark MLlib clustering library for this problem. You may store the centroids in memory if you choose to do so.

Let us say we have a set $\mathcal{X}$ of n data points in the d-dimensional space $\mathbb{R}^d$. Given the number of clusters k and the set of k centroids $\mathcal{C}$, we now proceed to define various distance metrics and the corresponding cost functions that they minimize.

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

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

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

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2 \tag{2}$$

Note, that in the cost function, the distance value is squared. This is intentional, as it is the squared Euclidean distance and the algorithm is guaranteed to minimize.

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

$$|a - b| = \sum_{i=1}^{d} |a_i - b_i| \tag{3}$$

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

$$\psi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} |x - c| \tag{4}$$

Iterative k-Means Algorithm: We learned the basic k-Means algorithm in class which is as follows: k centroids are initialized, each point is assigned to the nearest centroid and the centroids are recomputed based on the assignments of points to clusters. In practice, the above steps are run for several iterations. We present the resulting iterative version of k-Means in Algorithm 1.

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:         Calculate the cost for this iteration
8:         for each cluster c do
9:             Recompute the centroid of c as the mean of all the data points assigned to c
10:        end for
11:    end for
12: end procedure
```

Task

Complete the `my_kmeans` class and other supporting functions to implement iterative k-means using **Spark**. Please use the dataset from the `data` for this problem. The folder has 3 files:

1. `q2-data.txt` contains the dataset which has 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.

2. `q2-c1.txt` contains k initial cluster centroids. These centroids were chosen by selecting $k = 10$ random points from the input data.

3. `q2-c2.txt` contains initial cluster centroids which are as far apart as possible, using Euclidean distance as the distance metric. (You can do this by choosing 1st centroid c_1 randomly, and then finding the point c_2 that is farthest from c_1, then selecting c_3 which is farthest from c_1 and c_2, and so on).

Set the number of iterations (`MAX_ITER`) to 20 and the number of clusters k to 10 for all the experiments carried out in this question.

When assigning points to centroids, if there are multiple equidistant centroids, choose the one that comes first in lexicographic order.

a) Using the Euclidean distance (refer to Equation 1) as the distance measure, compute the cost function $\phi(i)$ (refer to Equation 2) for every iteration i. This means that, for your first iteration, you'll be computing the cost function using the initial centroids located in one of the two text files. Run the k-means on `q2-data.txt` using `q2-c1.txt` and `q2-c2.txt`. Generate a graph where you plot the cost function $\phi(i)$ as a function of the number of iterations $i = 1..20$ for `q2-c1.txt` and also for `q2-c2.txt`.
(Hint: Note that you do not need to write a separate Spark job to compute $\phi(i)$. You should be able to calculate costs while partitioning points into clusters.)

b) What is the percentage change in cost after 10 iterations of the K-Means algorithm when the cluster centroids are initialized using `q2-c1.txt` vs. `q2-c2.txt` and the distance metric being used is Euclidean distance? Is random initialization of k-means using `q2-c1.txt` better than initialization using `q2-c2.txt` in terms of cost $\phi(i)$? Explain your reasoning.
(Hint: to be clear, the percentage refers to (cost[0]-cost[10])/cost[0].)

c) Repeast part a) using the Manhattan distance.
(Hint: It's possible that for Manhattan distance, the cost do not always decrease. K-means only ensures monotonic decrease of cost for squared Euclidean distance. Look up K-medians to learn more.)

d) Repeat part b) using the Manhattan distance.