FINAL PROJECT Option #1: GEO-LOCATION CLUSTERING USING THE k-MEANS ALGORITHM

Due: Monday 11 May 2020 (11:59PM) – **NO EXTENSION**

Project Goal

In this project you and your group will use SPARK to implement an iterative algorithm that solves the **clustering problem** in an efficient distributed fashion. *Clustering* is the process of grouping a set of objects (or data points) into a set of k clusters of similar objects. Thus, objects that are similar should be in the same cluster and objects that are dissimilar should be in different clusters.
Clustering has many useful **applications** such as finding a group of consumers with common preferences, grouping documents based on the similarity of their contents, or finding spatial clusters of customers to improve logistics. More specific use cases are

- *Marketing*: given a large set of customer transactions, find customers with similar purchasing behaviors.
- *Document classification*: cluster web log data to discover groups of similar access patterns.
- *Logistics*: find the best locations for warehouses or shipping centers to minimize shipping times.

We will approach the clustering problem by implementing the **k-means algorithm**. k-means is a distance-based method that iteratively updates the location of k cluster *centroids* until convergence. The main user-defined ingredients of the k-means algorithm are the distance function (often Euclidean distance) and the number of clusters k. This parameter needs to be set according to the application or problem domain. (There is no magic formula to set k.) In a nutshell, k-means groups the data by minimizing the sum of squared distances between the data points and their respective closest centroid.

Goal: Implement *k*-means in SPARK and use it for geo-location clustering on various datasets of spatial locations. **Can you find an even bigger geo-location dataset for which computation is not feasible in the pseudo-cluster? Cluster this data by executing your SPARK program on Amazon EMR and report your results and experiences.**

Problem 1: Understanding the *k*-means Algorithm

k-means is a distance-based method that iteratively updates the location of *k* cluster centroids until convergence. The main user-defined ingredients of the k- means algorithm are the distance function (often Euclidean distance – for geo-location data we will also explore the great-circle distance) and the number of clusters *k*. This parameter needs to be set according to the application or problem domain. (There is no magic for-mula to set *k*.) In a nutshell, *k*-means groups the data by minimizing the sum of squared distances between the data points and their respective closest centroid. MMDS chapter 7.3 (`http://infolab.stanford.edu/~ullman/mmds/ch7.pdf`) gives pseudo code and implementation strategies for the *k*-means clustering algorithm.

Problem 2: Data Preparation

This problem prepares three datasets. Submit your programs for data pre-processing to the Github repository. Detailed submission instructions follow below.

Step 1: Prepare device status data

Review the contents of the file `data/devicestatus.txt`. Upload this file to a S3 bucket. You will have to pre-process the data in order to get it into a standardized format for later processing. This is a common part of the ETL (Extract-Load-Transform) process called *data scrubbing*.
The input data contains information collected from mobile devices on Loudacre's network, including device ID, current status, location and so on. *Loudacre Mobile* is a (fictional) fast-growing wireless carrier that provides mobile service to customers throughout western USA. Because Loudacre previously acquired other mobile provider's networks, the data from different subnetworks has a different format. Note that the records in this file have different field delimiters: some use commas, some use pipes (|) and so on. Your task is to use PySpark in Jupyter Notebooks to:

- Load the dataset
- Determine which delimiter to use (hint: the character at position 19 is the first use of the delimiter)
- Filter out any records which do not parse correctly (hint: each record should have exactly 14 values)
- Extract the date (first field), model (second field), device ID (third field), and latitude and longitude (13th and 14th fields respectively). You might want to store latitude and longitude as the first two fields to make it consistent with the other two datasets.
- Filter out locations that have a latitude and longitude of 0.
- The model field contains the device manufacturer and model name (e.g. Ronin S2.) Split this field by spaces to separate the manufacturer from the model (e.g. manufacturer Ronin, model S2.)
- Save the extracted data to comma delimited text files in S3.
- Confirm that the data in the file(s) was saved correctly. Provide a screenshot named `devicedata.png` showing a couple of records in your Github repository.

- Visualize the (latitude, longitude) pairs of the device location data. You do not have to use SPARK for the visualization. You can use any technique and 3rd party library (e.g., https://www.youtube.com/watch?v=XiZbrii49pI)
- Submit your implementation by adding a Jupyter Notebook file (Mobilenet.ipynb) including your SPARK statements to the `clustering/Step1` folder in your Github repository. **Do NOT add any data!**

Step 2: Get and Visualize synthetic location data

Upload the synthetic clustering data from `http://statistical-research.com/wp-content/uploads/2013/11/sample–geo.txt to S3` and visualize the (latitude, longitude) pairs. You do not have to use SPARK for the visualization.

- Submit your implementation by adding a Jupyter Notebook file (Synthetic.ipynb) including your visualization to the `clustering/Step1` folder in your Github repository. **Do NOT add any data!**

Step 3: Get and Pre-process the DBpedia location data

Upload the large-scale clustering data of (latitude, longitude) pairs extracted from DBpedia (`lat–longs.zip`) to S3. Each record represents a location/place that has a Wikipedia article and latitude/longitude information. The format is: `lat long name–of–page`.
In total, there are 450,151 points in a 2D space (i.e., space with spherical geometry – *maybe* it would make sense to use the **great circle distance** when analyzing this data...). To get a smaller sample of this dataset for testing purposes, you could put a bounding box around the US and filter only those records inside the bounding box. Try to visualize this data. Eventually, you want to cluster the whole world using the entire dataset.

- Submit your implementation by adding a Jupyter Notebook file (DBPedia.ipynb) including your SPARK statements to the `clustering/Step1` folder in your Github repository. **Do NOT add any data!**

Problem 3: Clustering Big Data – *k*-means in Spark

Step 1: Understanding Parallel Data Processing and Persisting RDDs

This is the theory part of the project. Review Module 7 and Module 8 to understand the main data concept in SPARK – Resilient Distribute Datasets (RDDs). You will need to persist an RDD (at least once) in your *k*-means implementation. Additionally, make yourself familiar with how to view stages and tasks, e.g., using the Spark Application UI or Spark History Server.

(see https://docs.aws.amazon.com/emr/latest/ManagementGuide/app-history-spark-UI.html)

Step 2: Implementing k-means

Detailed implementation requirements/specifications are listed below.

The following functions will be useful for calculating *k*-means:

- `closestPoint`: given a (latitude/longitude) point and an array of current center points, returns the index in the array of the center closest to the given point
- `addPoints`: given two points, return a point which is the sum of the two points.

- `EuclideanDistance`: given two points, returns the Euclidean distance of the two.
- `GreatCircleDistance`: given two points, returns the great circle distance of the two.

Note, that the `addPoints` function will be used to compute the new cluster centers. As we are working with spatial data given as latitude-longitude pairs implementing this function in a meaningful way will need some thought!

The used distance measure (Euclidean or great circle), as well as the parameter k (number of clusters) should be read as an input from the command line.

Select a suitable convergence criterion and create a variable `convergeDist` that will be used to decide when the k-means calculation is done, i.e. when the amount the locations of the means change between iterations is less than `convergeDist`. A "perfect" solution would be 0, which is not achievable due to numeric computations. Hence, `convergeDist` is a parameter that represents a "good enough" solution. Select a *small* value for it that makes sense for your dataset and convergence criterion.

Parse the input file, which should be specified as a S3 address, into (latitude,longitude) pairs. Be sure to persist (cache) the resulting RDD because you will access it each time through the following iterations.

Now, plan and implement the main part of the k-means algorithm. Make sure to consider an efficient implementation being aware of tasks, stages, and cached RDDs.

When the iteration is complete, display and return the final k center points and store the k clusters (i.e., all data points plus cluster information).

Step 3: Compute and Visualize Clusters

In this step, you will compare the clusters using Euclidean distance vs. great circle distance.

Calculate the k-means clusters for the **device location data** using $k = 5$.

Calculate the k-means clusters for the **synthetic location data** using $k = 2$ and $k = 4$.

Calculate the k-means clusters for the large-scale **DBpedia location data**. You will need to experiment with the number of clusters (maybe use $k = 6$ for a start or $k = 2$ or 4 if you use the US locations only). Argue, what choice of k makes sense by considering the problem context, i.e., what could the clusters actually *mean/represent*?

Visualize the clusters and cluster centers (use a random subset of data points for the last dataset) for both distance measures. Can you observe a difference?

Step 4: Runtime Analysis

Compare the runtime of your k-means implementation (using the same value for k) for all three datasets using the local mode with at least two threads. Further, rerun your implementation without using persistent RDDs and compare those runtimes to the previously obtained ones. Create a table or bar plots summarizing these results and briefly discuss your findings. After job completion you can read off the runtimes and other statistics from the Spark History Server. You might want to rerun each experiment a couple of times and use the average runtime for a more robust comparison (time permitting).

Step 5: Documentation of Approach and Results (Report)

Write the project report documenting your clustering approach, your implementation, the obtained results, and runtime analysis. This report should be readable for an informed outsider and it should not require the reader to look at or run any code.

Bonus: Big Data and Cloud Execution (4 points)

Find and/or crawl an even bigger dataset to cluster using your k-means approach. This dataset should be big enough that your k-means clustering computation is not feasible in your pseudo-cluster. Cluster this data by executing your SPARK program on Amazon EMR and report your results and experiences. If, you do not find another geo-location dataset, feel free to perform clustering on any other Big data clustering problem. Keep in mind that for higher-dimensional data, the **interpretation and visualization** of the retrieved clusters is much more challenging.
Document your cloud execution approach and provide the data source in your final project report. Add your pre-processing code to the src folder in your Github repository. Describe your findings including dataset size and runtimes in your final project report.

Final Submission Instructions

Submit your report including **documentation**, as well as, **results** as `project-report.pdf` by adding it to the `/clustering` folder in your Github repository. Submit your implementation by adding your Jupyter Notebooks to the `/clustering/src` folder in your Github repository. **Do NOT add any data!**

Copyrights

Problems are adapted and data is taken from Marion Neumann's CSE 427 (Fall 2019) course at Washington University in St. Louis, which was itself adopted from Pedro Domingos' class on Data Mining/Machine Learning at University of Washington, 2012.