# CSE427 – Final Project #3: $k$-means for Geo-location Clustering in SPARK

M. Neumann

Due 05/05/2016 1pm (**no extension!**)

## Project Goal

In this project you and your group will interactively get to know SPARK and use it to implement an iterative algorithm that solves the **clustering problem** in a parallel 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. See MMDS chapter 7.1 and 7.3 for more details (`http://infolab.stanford.edu/~ullman/mmds/ch7.pdf`).

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.

## Getting Started

Update your SVN repository, you will find additional materials for the *final project* in the folder `final_project/spark`.

Download the VM configured with SPARK from HERE:

```
https://classes.cec.wustl.edu/cse427/student.vmdk.zip
```

To setup this VM follow the instructions on setting up the VM for the course webpage:

```
http://sites.wustl.edu/neumann/resources/vm-setup/
```

I recommend using 2048MB of base memory.

Run the following command every time you (re)start your VM:

```
$  $DEV1/scripts/training_setup_dev1.sh
```

User and password are both `training`. Guest Additions are installed, so you should for instance be able to copy and past (files and clipboard) from your host machine. To enable this go to Settings > General > Advanced in VirtualBox and enable the functions.

# Indicating Group Work

Use the file `partners.txt` to indicate group work. Follow these instructions exactly, to ensure to get credit!

- `partners.txt` needs to include up to 3 wustlkeys in the first three lines (one line per wustlkey)
- **first line/wustlkey is the repository, where the solution is located**. We will **only** consider the submission in this repository!
- Every student in a group needs to have **the same** `partners.txt` in the `final_project/spark` folder in their repository (indicating that the partnerships are **mutually accepted**)!
- If you do not have a partner, try to find one! If you want to submit on your own, indicate your wusltkey in the first line of `partners.txt` and leave the second line blank.

# Usage Agreement

By downloading and using the dataset `lat_longs` (cf. Problem 2 – Step 3) you agree as follows:

- I agree to **delete** this dataset once the project has been completed.
- I will not redistribute this data in any form.

# Problem 1: System Exploration

All steps in this problem need to be completed as part of **milestone 1**.

## Step 1: Hue File Browser and Data Deployment

The goal of this problem is to familiarize yourselves with SPARK and the new VM. First, note that we can use the **Hue File Browser** to interact (upload/ view files and directories/ view file contents etc.) with HDFS . Access Hue by opening Firefox and following the hue bookmark or typing this

url `http://localhost:8888`. Create an account using `training` both as user and password. Find the little File Browser icon on the upper right that says *Manage HDFS* and click on it.

The data used for this problem is located in the following local folder:

```
/home/training/training_materials/data.
```

Put some sample data (e.g. `frostroad.txt`) into HDFS .

## Step 2: Spark Documentation

Now, find the **Spark Documentation** by opening Firefox and following the Spark Doc bookmark or typing this url `file:/usr/lib/spark/docs/_site/index.html`. From the Programming Guides menu, select the Spark Programming Guide and briefly review the guide. This will be a useful reference throughout the project. From the API menu, select the programming language (Scala, Pyton, Java) you want to work with. Remember that this API exists – you might want to use it as a reference later!

## Step 3: Spark Shell and RDDs

Explore the **Spark Shell and RDDs** using the Python (start it using `pyspark`) or Scala (start it using `spark-shell`) SPARK shell. Load the text file `purplecow.txt` from your local file system into a variable called `mydata`. Recall that the default file system is HDFS to access data on your local file system use `file:` and then the path on your local file system. Count the number of lines in that file. In the same way, execute the `collect` operation on the dataset (this operation should only be used for small sample data!). Note that you can use **command completion** for paths and to see all available transformations and operations you can perform on an RDD (e.g./ type `mydata.` and use the TAB key). Now, perform the same operations on the `frostraod.txt` poem in HDFS . To exit the shell, type `exit`.

## Step 4: Word-Count and Job Execution

By following the example in the lecture slides, perform **Word Count in Spark** on the local text file `purplecow.txt` using the SPARK shell. You can monitor your jobs and stages in the **Spark Application UI** in the web browser (`http://localhost:4040`).

Now, examine and run the word count Python implementation (`wordcount.py`) provided in `$DEV1/exercises/yarn` on the local file `purplecow.txt` and on `frostroad.txt` stored in HDFS . There could be quite some warnings, however, you should also see the result printed on the screen. Where are your jobs (both SPARK shell and script) executed? How can you change the execution mode (local, local multi-threaded, cluster)?

Finally, run the word count job on the (pseudo) cluster. There won't be an output on the screen. However, you can view the job status and statistics from the **Hue Job Browser**. Open Hue in the web browser (cf. Step 1) and click on the Job Browser icon on the upper right that says *Manage Jobs*. Now, you should see your word count job there (hopefully it completed successfully). Now, leave the Job Browser open and resubmit your job, you should see the progress of this job and also note a button on the very right that you can use to Kill your job. Try it!

Jobs that run/ran on the cluster can also be viewd from the **YARN Resource Manager** (RM). You can access it from your web browser (`http://localhost:8088`). The RM also shows you that your cluster only consists of one node – pseudo cluster (click on *Nodes* in the left panel). So, in our toy environment it is better to run SPARK jobs in local mode using multiple threads in order to *simulate* the behavior in a real multi-node cluster.

# Problem 2: Data Preparation

This problem prepares three datasets and defines **milestone 2**. Submit your programs for data pre-processing as submission of **milestone 2**. Detailed submission instructions follow below.

## Step 1: Prepare device status data

Review the contents of the file `$DEV1DATA/devicestatus.txt`. 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

- 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 the `/loudacre/devicestatus_etl` directory on HDFS .
- Confirm that the data in the file(s) was saved correctly. Provide a screen-shot named `devicedata.png` showing a couple of records in your SVN repository.

Submit your implementation by adding a folder called `milestone2` to the `final_project/spark` folder in your SVN repository. Add a file including your SPARK statements to this folder, e.g., `step1.pyspark`. **Do NOT add any data!**

**Add the new files/folders to your SVN repo before committing:**

```
$ svn add milestone2
$ svn add milestone2/step1.pyspark
$ svn add milestone2/devicedata.png
$ svn commit -m 'milestone 2 submission' .
```

### Step 2: Get and Visualize synthetic location data

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

Submit your plot as `step2.png` by adding to the `final_project/spark/milestone2` folder in your SVN repository. **Do NOT add any data!**
**Add the new files/folders to your SVN repo before committing:**

```
$ svn add milestone2/step2.png
$ svn commit -m 'milestone 2 submission' .
```

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

Download the large-scale clustering data of (latitude, longitude) pairs extracted from DBpedia (`https://classes.cec.wustl.edu/cse427/lat_longs.zip`). 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 started on this dataset, you could put a bounding box around the US and filter only those records inside the bounding box to get a smaller sample. Eventually, you could try to $k$-center the whole world...

## 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. Consider the lecture slides and also the set of slides on RDD persistence provided in the project folder in your SVN repository. 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.

### Step 2: Understanding and Implementing k-means

MMDS chapter 7.3 (`http://infolab.stanford.edu/~ullman/mmds/ch7.pdf`) gives pseudo code and implementation strategies for the $k$-means clustering algorithm. 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 – that is, (x1+x2, y1+y2)
- `EuclideanDistance`: given two points, returns the Euclidean distance of the two.
- `GreatCircleDistance`: given two points, returns the great circle distance of the two.

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.

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 changes between iterations is less than `convergeDist`. A "perfect" solution would be 0; this number represents a "good enough" solution. For this project, use a value of 0.1.

Parse the input file, which should also specified as a variable via the command line, 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 $k$ = 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 if you use the US locations only).

Try to 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 summarizing these results and briefly discuss your findings. You can read off the runtimes

from the Spark Application UI after job completion. 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.

## Final Submission Instructions

Submit your report including **documentation**, as well as, **results** as `project_report.pdf` by adding it to the `final_project/spark` folder in your SVN repository. Submit your implementation by adding your implementations to the `final_project/spark/src` folder in your SVN repository. **Do NOT add any data!**

**Add the new files/folders to your SVN repo before committing:**

```
$ svn add src/*
$ svn add project_report.pdf
$ svn commit -m 'final project submission' .
```