T-764-DATA – Spring 2025
Project 2: Implementing K-Means clustering on Spark

The primary purpose is to get familiar and work with Spark but formally the goas are as follows:

1) **Setup an environment that you can run and test your code using Spark.**
2) **Develop and run a "complex" pipeline on Spark (in Scala/Java).**
3) **Implementing optimizations and evaluating against the original code as base-line reference point.**
4) **Write a technical report about your work and findings.**

1 Preparation

You will need to get Spark working in one way or another. The official web site has instructions: https://spark.apache.org/downloads.html

Here is a guide (a bit dated though) on how to create a Scala / Spark project in the Eclipse IDE: https://data-flair.training/blogs/create-spark-scala-project/

Or, if you are familiar with and prefer a notebook-style interface you can opt for Zeppelin (look like it is not available for Windows):
https://zeppelin.apache.org/docs/latest/quickstart/install.html

You choose to integrate Spark with the an IDE (like InteliJ IDE or Eclipse) and use that for development or you can opt for developing your pipeline using the Spark-Shell. Figuring this out is part of the challenge of this project.

2 The Task to solve

Your task will be to implement a pipeline that performs the *k*-Means clustering algorithm: https://en.wikipedia.org/wiki/K-means_clustering

In essence, you start with a set of representatives **R** and as set of points **P**. The process is then to cluster the data by assigning each ***p*** in **P** to the most similar representative ***r*** in **R**. From that clustering you calculate a new set of representatives **R'** by calculating the centroid of each grouping (cluster). This process is then repeated until the set of representatives does not change (or the change is sufficiently small).

Similarity function: The process is thus heavily dependent on deriving the similarity between points, especially if the points are high-dimensional vectors. The most commonly used distance metric is the Euclidian distance, but other distance metrics are also possible like Manhattan distance (Taxicab geometry):

- https://en.wikipedia.org/wiki/Euclidean_distance
- https://en.wikipedia.org/wiki/Taxicab_geometry

k-NN search: For each point ***p*** we are looking for its k-Nearest Neighbor in the set of representatives **R**, where *k* happens to be *k*=1. Thus, it may be of value to you to implement of form of *k*-NN search (or some structure to keep track of the search). Now the *k*-NN algorithms is dealing with all kinds of issues that relate to the nature of the data, but we do not need to do any of that as our SIFT vectors have already been normalized and we can safely just apply a distance metric on all dimensions without any pre-processing. For more general info on *k*-NN we can read the Wikipedia page: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

The data you will be clustering will be 128-dimensional SIFT vectors that you will read from a Hadoop Sequence file (.seq) format: https://wiki.apache.org/hadoop/SequenceFile and I will give you the code needed to safely read the SIFTs from this file format (we did this in class actually).

Other useful links:

- Scala documentation for Spark: https://spark.apache.org/docs/latest/api/scala/org/apache/spark/index.html
- Spark standalone deployment: https://spark.apache.org/docs/latest/spark-standalone.html

3 Baseline, optimization, and evaluation

I will provide you with files with SIFT descriptors. The files come from this dataset: http://corpus-texmex.irisa.fr

This dataset ranges from 1 billion points to 1 million points and with it we have 10k queries and the ground-truth for the *k*-NN. The trouble with the original dataset is that it is in a raw binary format of unsigned char (0-255). Java does not support unsigned so I have created signed byte version that is "Java friendly". The files are in a specific format and I will proved examples of code on how to read those files.

- 100M_bigann_base.seq is about 15 GiB in size and has 100M points
- bigann_query.seq (bigann_query.ser) is the 10K query vectors
- 100M_gnd_info.txt is the ground truth for the 10K queries

You should start with an initial experiment, using the most straight forward code that uses a Euclidian distance function do build the following baseline:

- **The initial baseline**: Get a stable *k*-Means clustering using **P**=10.000 points (the small file) and an initial of **R**=100 representatives.
  Keep track of both time and iterations needed (and, if needed, you can put a max iterations or threshold on change as you see fit).

Follow up experiments that you can do would be:

- **Varying dataset size**: Increase the size of **P** by taking data from the 100M file.
- **Varying the number of clusters**: Pick more initial representatives **R**.
- **Experiments using optimized code**: Here you can try changing the distance calculation, do early-halting or loop-unwinding etc.

Optimization ideas to try out

Unroll the distance function: The distance function D(*p*, *r*) for high-dimensional vectors can be implemented as a for-loop over each dimension. This is however not always the most efficient as looping 128 times is not free. Unwinding the loop is just writing a for-loop that does more work in each iteration of the loop and thus loops less often. https://en.wikipedia.org/wiki/Loop_unrolling

Early halting: For each *p* in **P** we scan it against all *r* in **R.** But, we are just looking for *p*'s 1-NN (the *r* most similar to our *p)*. Thus, in the distance calculation process, as soon as the current distance calculated exceeds lowest distance value found in previous evaluations (some other D(*p*,*r*) had a lower value) we can terminate the current calculation and reject this *r* as the 1-NN. This is called early halting of the distance calculation.

Use an index: Build a hierarchy structure to speed up the clustering process.

4 This is a group project

You may work in groups of (up to) three students. Each of you should assume to spend no more than 20 quality hours on this project, as it accounts for 15% of your grade. **The deadline is Friday Feb. 28th at 23:59.**

5 Deliverables

You will need to hand in both a technical report and the code. In the report you can replace the Abstract, Intro & Background with just a single paragraph on describing the content. The paper should thus mainly be what you did, what experiments you did and results with some discussion on what they mean. In the methods part I expect you to use the symbols for the Spark pipeline that I presented in class.

As usual the report will account for most of your grade (70%), but I will also look at both code and how you conduct the experiments etc.

You must deliver three files:

1) A **PDF of the report** describing your solution and your findings, and effort and outcome towards the extra credits if applicable (see above).
2) A zip-file with the code and other working documents, designs etc. of your execution pipeline. The code that **should be commented** and have good structure etc.
3) A short README.txt file describing the structure of your code and conducted the experiments etc.

```
/** This function is specifically designed load a file of SIFT descriptors into an RDD.
  * @param sc SparkContext of our master node.
  * @param fileName Name and path to the file full of SIFT descriptors --
  *        format: [hadoop.io.IntWriteable, eCP.Java.SiftContainer].
  * @return RDD[eCP.Java.SiftDescriptor.Container] RDD of all the SIFT's in the file
  * (using the key as the SIFT's ID).
  */
 def loadSIFTs( sc: SparkContext, fileName: String ) : RDD[SiftDescriptorContainer] = {
    // We use the spark context to read the data from a HDFS sequence file, but we need to
    // load the hadoop class for the key (IntWriteable) and map it to the ID
   sc.sequenceFile(fileName, classOf[IntWritable],
classOf[SiftDescriptorContainer]).map(it => {
   // first we map to avoid HDFS reader "re-use-of-writable" issue.
  val desc : SiftDescriptorContainer = new SiftDescriptorContainer()
  desc.id = it._2.id
  // deep copy needed as the Java HDFS reader re-uses the memory
  it._2.vector.copyToArray(desc.vector)
  desc
  })
}
```