Introduction

Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. (Source: Wikipedia)

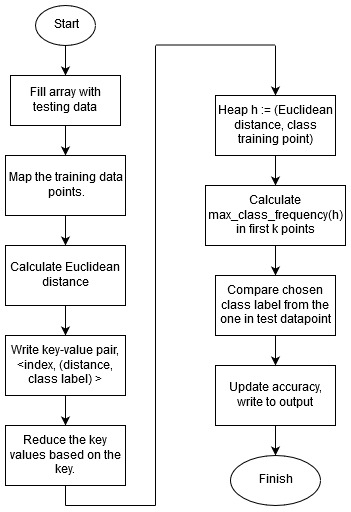
This assignment aims at implementing k-NN and k-means using both Hadoop and spark. We have to do a comparative study of the efficiency in both of them.

k-NN

The dataset for k-NN had a million datapoints, each with 9 dimensions. The dataset was divided into training set and test set. k-NN is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. We used k-NN for classification. Generally, k is taken as an odd number to avoid ties in classifications.

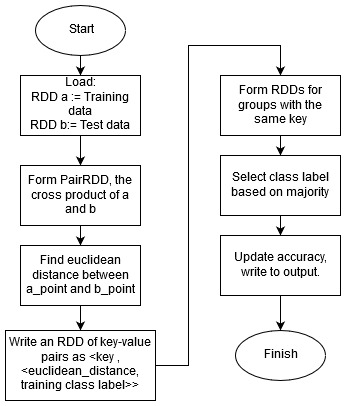
Algorithm for Hadoop:

1. Array arr <- testdatapoints.
2. Start the job.
3. Map training points.
4. Calculate euclidean\_distance(test\_data\_point, training\_data\_point).
5. Write key-value pair: <index\_test\_point, (euclidean\_distance, class\_training\_point)>.
6. Reduce the key-value pairs based on key.
7. Heap h <- (euclidean\_distance,class\_training\_point).
8. Calculate max\_class\_frequency(h) in first k points.
9. Compare chosen class with class in test datapoint.
10. Write value to output.



Algorithm for k-NN in Spark:

1. Call Spark application with <path-to-training> <path-to-test> <path-to-output>
2. RDD a <- Training\_points
3. RDD b <- Test\_points
4. Form a PairRDD with cross product of a and b.
5. Calculate euclidean\_distance(a\_point,b\_point).
6. Write an RDD of key-value pairs as <key,<euclidean\_distance,training\_class>>.
7. Write an RDD that groups on key.
8. Select classification based on majority among k values.
9. Write result to output file.



Use of map and reduce in Hadoop

Map: The map function (part of the Mapper Class) accepts the training data points and iterates for every test data point. It calculates the Euclidean distance and pairs it with the classification of the training points. Each of these values (<distance, class>) has the index of the test data point as the key. It writes these key value pairs into the context.

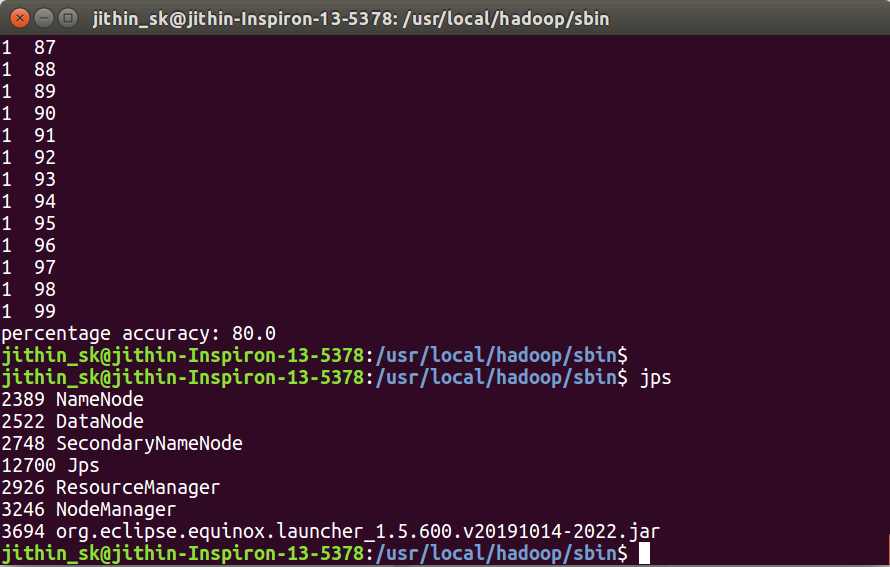
Reduce: The reduce function (part of the Reducer Class) accepts the key-value pairs based on the key (the index of the test data point), and finds the nearest k values by using a max-heap. It finds out the number of occurrences of a particular class in the first k and assigns the test data point the class with the maximum occurrence. It updates the accuracy and then writes the test point index and class into the file.

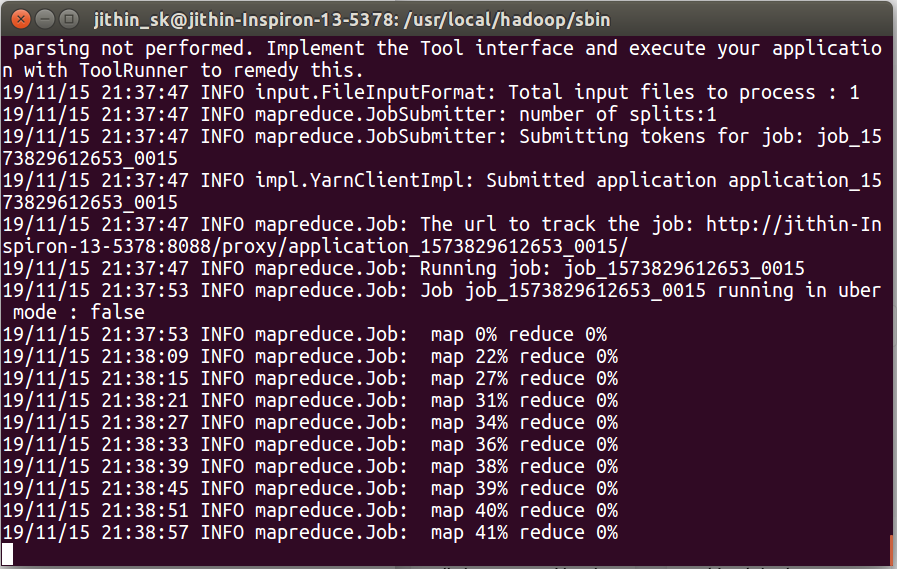
RDD Transformations and Actions:

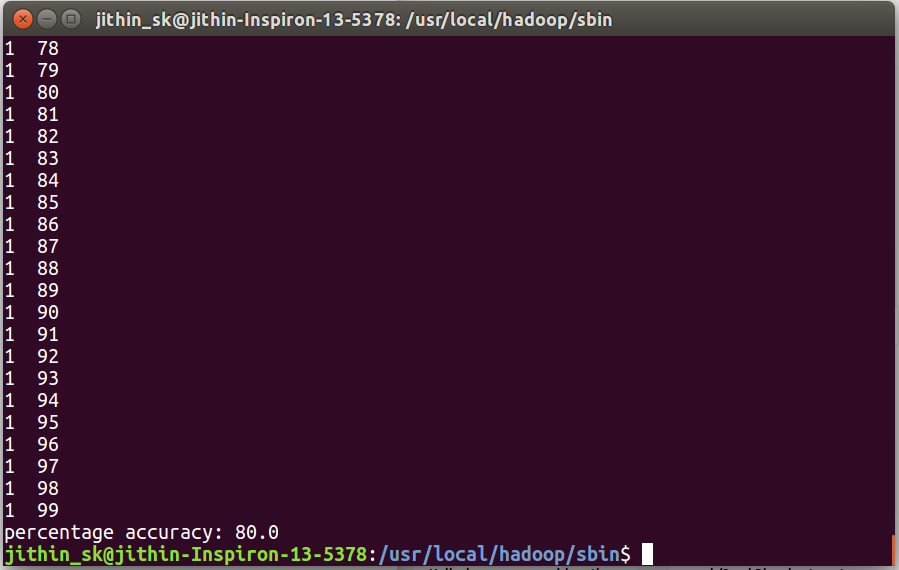
1. Transformation1: Loading test data points into RDD.
2. Transformation2: Loading training data points into RDD.
3. Transformation3: Creating pairRDD with cross product of the 2 RDDs.
4. Action1: Calculating Euclidean distance.
5. Transformation4: Creating RDD with key-value pairs. Key = index of test data point. Value = <distance,class> pair.
6. Transformation5: Grouping the key-value pairs in RDD based on key.
7. Action2: Selecting the classification from the nearest k values.
8. Transformation6: Writing the values to an output file.

k-NN in Hadoop in action:

We used a 1+1 master-slave configuration to run k-NN in Hadoop.





We used 1000 training examples and 100 testing examples, as using all of the instances was taking a long time. An accuracy of 80% was achieved. 

It took 289060 milliseconds for the map part of the program to run. The reduce operation took 2358 milliseconds. The time measured was displayed on terminal while running the map-reduce job on Hadoop.

