DS503 Project 2 Report

Group 3

Wenhan Ji, Bhavsar, Siddhant

Problem 1

1 Dataset Creation

1.1 Point Dataset(P.csv - 117.3MB)

Use python random package to generate 12000000 rows point dataset P.csv.
The x, y of points ranges from [0, 10000].

```
x = np.random.randint(0,10000,12000000)
y = np.random.randint(0,10000,12000000)
```

1231,9854
4157,2391
8245,1735
6482,7107
9249,4536
3150,9721

1.2 Rectangular Dataset(R.csv - 54MB)

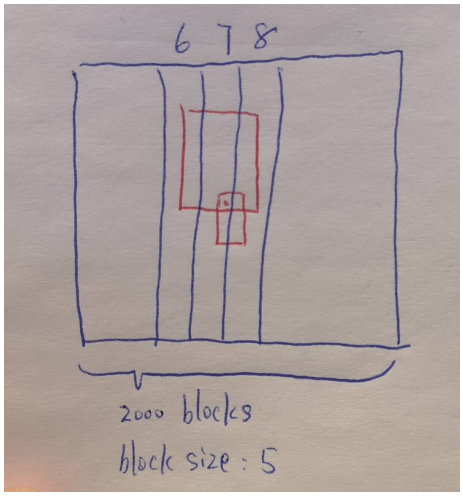
Use python random package to generate bottom left axis of rectangular ranges from [0,10000]. Then generate random value of height and weight. So we can get the top right axis.

```
x_bottom_left = np.random.randint(0,10000)
y_bottom_left = np.random.randint(0,10000)
width = np.random.randint(1,5)
height = np.random.randint(1,20)
x_top_right = x_bottom_left + width
y_top_right = y_bottom_left + height
```

Here the first element is region. The other four is the bottom left and top right axis.

1,5682,1920,5683,1929
2,3309,7311,3312,7314
3,7123,9592,7127,9599
4,2491,5218,2493,5222
5,9269,6958,9271,6974
6,558,9892,559,9898

2 Spatial Join



Here we cut the whole coordinate into 2000 blocks vertically. Each block's width is 5. In this way to make the job work in a distributed pattern.
For point, block number = x / 5.
For rectangular, occupy block number are [leftBottomX /5, rightTopX/5]

In mapper, we select out the points that's inside the window and rectangular which has overlap with window. We figure out the which block does points and rectangular occupies and set the block number as key. In reducer, for each block, we select out the points that's inside the rectangular.

Mapper (two mapper function to process two file input separately).

- **First mapper reads the points P.csv.**

Figure out whether the point is inside the window. If it's inside window, write to context.
Figure out the point's block location. Set the key as block location, value as point axis.

- **Second mapper reads the rectangular R.csv.**

Select out the rectangular which has overlap with the window.
Figure out which blocks do the window occupy and which blocks do the rectangular occupy.
Figure out the overlap block with each other and write the rectangular to context.
For example:

Now we find rectangular has overlap with window.
Window locates on block 2, 3, 4, 5. Rectangular locates on 4,5,6.
Write the key as 4, value as rectangular input value.
Write the key as 5, value as rectangular input value.

Shuffle & Sort:
In this phase, we group by key which is block number. Every point and rectangular in the same block will all be assigned to one reducer task.
Reducer:
Loop over the list which contains all points and rectangular belongs to one block.
Create two HashSet, one to store point and one to rectangular in this block.
Then loop over two HashSet, if points is inside the rectangular, write out the rectangular region and point.

Run Job Commands:
We use the following command to run the job.
The first and second parameter is the location of points and rectangular inputs.
The second parameter is the window location.

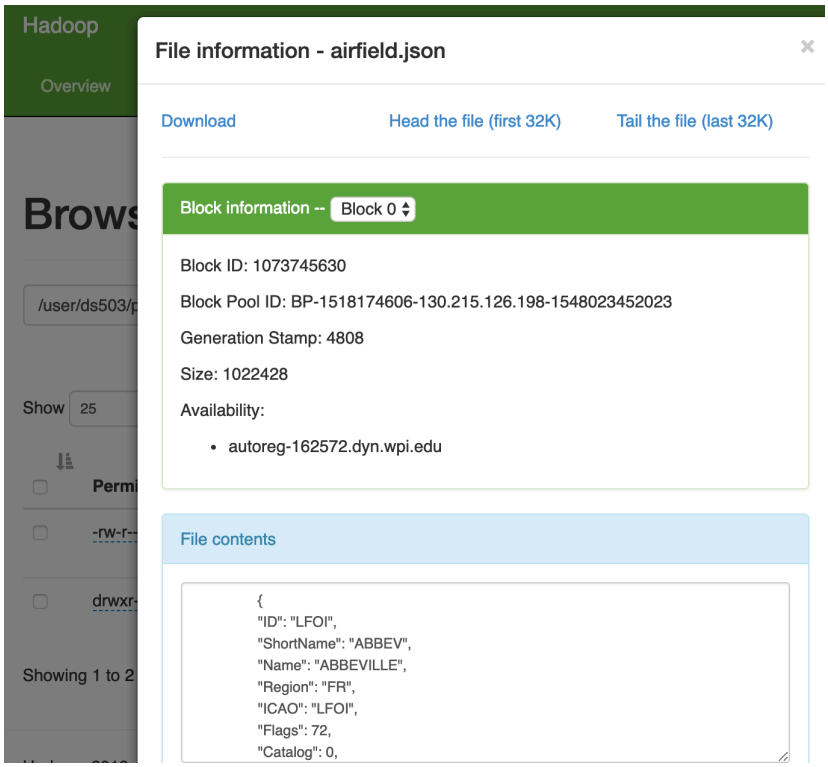
```
hadoop jar SpatialJoin.jar
/user/ds503/project2/problem1/P.csv
/user/ds503/project2/problem1/R.csv
/user/ds503/project2/problem1/output
1000,1000,1030,1030
```

Output:
The following is output with region as key and joined points as value.

```
267522   1001,1009
499936   1001,1009
1040560  1001,1009
1557109  1002,1020
267522   1002,1020
499936   1002,1020
```

Problem 2

1 load the dataset into hdfs



2.1 Custom Input Format

Here, we write the new JsonInputFormat that extends the TextInputFormat.
In JsonInputFormat, we override the RecordReader method to return our new JsonRecordReader object.
We write a new JsonRecorderReader class that extends the RecorderReader. When read the file, we observe the start tag "{" and end tag "}". In between, we consider it as one value.
In the getCurrentValue() method, we write a json2CSV function to change the json format to required csv format.

The following shows the input value of our mapper.

2.2 Group by elevation and do the count

In mapper, we parse out the Elevation value and set it to be key, value is one.
In reducer, for each elevation, we sum all the 1 which is exactly the count.

- **Run**

We use the following command to run the job. The first parameter is input json file. The second parameter is output directory.

hadoop jar JsonInput.jar
/user/ds503/project2/problem2/airfield.json
/user/ds503/project2/problem2/output

- **Result**

1	27
2	6
3	36
4	15
5	6
6	15
8	3

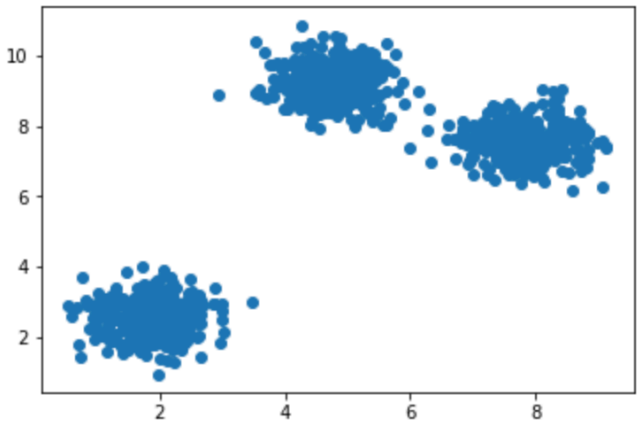
Problem 3

1. Dataset Creation

We use python to create 1000 samples with 2 features.

	0	1
0	1.868372	3.326518
1	4.472808	8.349993
2	1.163874	2.846222
3	5.144145	8.711892
4	7.817852	8.001391

Our data has 3 clusters. Below shows scatter plot.



2. Cluster the data.

We set the K and data dimension length as two parameter.
In this project, we let K = 3, dimLen = 2.

- **Centroid Initialization**

According to K and dimLen, we write a function initCentroid to randomly pick 3 centroids. Then we write it to hdfs.

File contents

```
5.333330139537765,5.305162719267644
5.963960402833438,5.5334597096117735
5.7184032601673165,5.428631183071263
```

- **Read centroid to String.**

We write a function to read the centroid file in hdfs and turn to String.

- **Global Variable**

We use conf.set to set four global variables.
The *k* and *dimLen* are fixed 3 and 2.
Iteration is the iteration number.
Centroid: if it's the first iteration, we read from our initialized centroid file. Otherwise, we read from new centroid that write from reducer in last iteration.

```
conf.set("centroid", centroid);
//k = number of clusters
conf.set("k", String.valueOf(k));
//dimLem = number of dimension for each point
conf.set("dimLen", String.valueOf(dimLen));
//i
conf.set("iteration", String.valueOf(iteration));
```

- **Stop iteration condition**

Condition1: if iterate more than 20 rounds

```
int iteration = 1;
while(iteration <= 20)
```

Condition2: if the sum of Euclidian distance between old and new centroid is smaller than 0.05.

```
if(iteration!=1) {
    Path centroidDiffPath = new Path( pathString: "/user/ds503/project2/proble
    double centroidDiff = readCentroidDifference(centroidDiffPath);
    if (centroidDiff < 0.05) {
        System.out.println("Successfully converge.");
        System.exit( status: 0);
    }
}
```

- **Mapper:**

In setup, we get the global variable centroid.
In map, for each point, we loop over all centroid to find the closet one.
Output key: the closet centroid
Value: currentPointLocation : 1 (used for count)

- **Combiner – optimization**

With combiner, it's faster to run every iteration.
For each point that assigned to the same centroid, we sum over x, y, z and do the count locally. The output key of combiner is still the centroid, value turns to "50,80,100 : 30".
It means there are 30 points assigned to this centroid, and their sum of x, y, z is 50, 80, 100.

- **Reducer**

In setup:
Set an oldNewCentroidDistanceSum to sum over the distance between old and new centroid.
In reduce:
For each centroid, we sum over all axis values and count. Then do division to compute average which is new centroid. Then we compute the distance between old and new centroid and add to oldNewCentroidDistanceSum.
Output value: the new centroid
Output key: NullWritable
In cleanup
We store oldNewCentroidDistanceSum to hdfs, which is used for the condition of stopping loop.

- **Run**

hadoop jar KMeans.jar 3 2

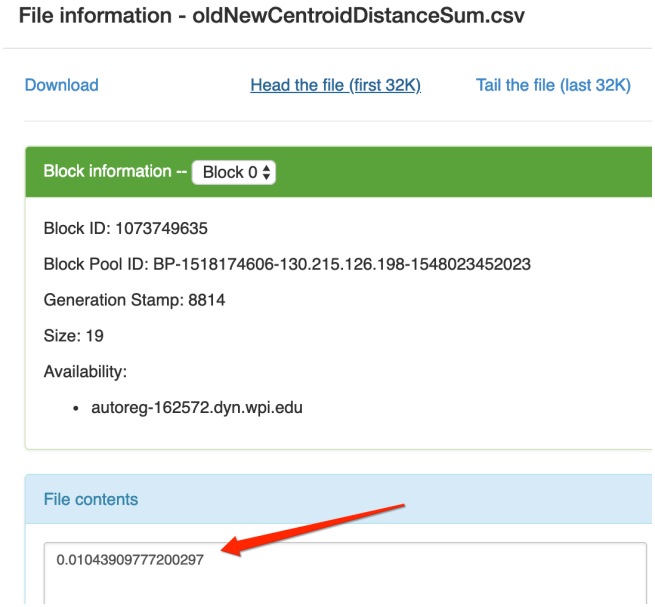
The first parameter is k. the second parameter is the number of features.

- **Result**

There are 4 iteration till stopping condition satisfied.

☐	drwxr-xr-x	Henry	supergroup	0 B	Feb 19 15:28	0	0 B	iter1	🗑
☐	drwxr-xr-x	Henry	supergroup	0 B	Feb 19 15:28	0	0 B	iter2	🗑
☐	drwxr-xr-x	Henry	supergroup	0 B	Feb 19 15:29	0	0 B	iter3	🗑
☐	drwxr-xr-x	Henry	supergroup	0 B	Feb 19 15:29	0	0 B	iter4	🗑

In the last iteration, we can see the sum Euclidian distance between new and old centroid is lower than threshold 0.05. That's why it stops iteration.



For each iteration, we can get 2 files.

1- a new centroid file

```
1.8728338239522972,2.5313791511579815
4.74959048420434,9.20958809825999
7.8074211931665225,7.570509729417816
```

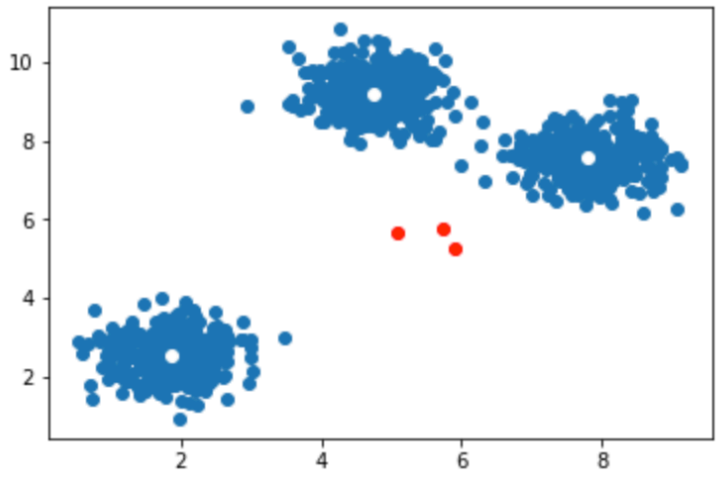
2- the distance between old and new centroid

```
0.01043909777200297

```

- **Result Visualize**

Below shows the centroid update. The red is our initialized, the white point the result centroid. We can see it moves to center of cluster.



- **Another job to get point belongs to which cluster(centroid).**

Just a single map, for every point, we find the closet centroid which is the updated one.
Now we know every point(left) belongs to which centroid(right).

drwxr-xr-x Henry supergroup 0 B Feb 19 15:29 0 0 B FinalOutput

```
2.9939568898378495,2.9414389082738945 1.8728338239522972,2.5313791511579815
3.003038800164589,2.729184007476489 1.8728338239522972,2.5313791511579815
3.033408378195924,2.1223734888119465 1.8728338239522972,2.5313791511579815
3.484371508695067,3.010841336815115 1.8728338239522972,2.5313791511579815
3.525659181424727,10.428925654601182 4.74959048420434,9.20958809825999
3.5387568093492083,8.926843822685905 4.74959048420434,9.20958809825999
3.574972207175877,9.0588190185113 4.74959048420434,9.20958809825999
3.5912804114582144,8.89272714539078 4.74959048420434,9.20958809825999
3.6655125551716097,10.122610478247996 4.74959048420434,9.20958809825999
3.701456567891422,8.793194181132389 4.74959048420434,9.20958809825999
3.7507868848226007,9.765405993589749 4.74959048420434,9.20958809825999
```