Student Name: Iliadis Viktoras
Student ID: 8180026

Big Data Management Systems

Assignment 2 - MapReduce/Hadoop

All the code and 2M generated data points can be found in the following repository that will host the code and all related files for all course exercises.

- **Question 1 - Hadoop Installation**

  Hadoop was installed following the official documentation: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

```
hadoop version

Hadoop 3.3.5
Source code repository https://github.com/apache/hadoop.git -r
706d88266abcee09ed78fbaa0ad5f74d818ab0e9

Compiled by stevel on 2023-03-15T15:56Z
Compiled with protoc 3.7.1
From source with checksum 6bbd9afcf4838a0eb12a5f189e9bd7
This command was run using
/home/mainuser/hadoop/hadoop-3.3.5/share/hadoop/common/hadoop-common-3.3.5.jar
```

  Dependencies Installation

```
%pip install numpy matplotlib mrjob
```

- **Question 2 - Data Generation**

  The following code generates a file with 2M datapoints (x,y) coordinate pairs. The generation is biased towards the creation of three clusters, by choosing a-priori three centers that you can find in the centroids.txt file. This is done by selecting a random distance r and a centroid randomly and then adding the generated distance r to the x and y coordinates of the centroid to create a new datapoint.

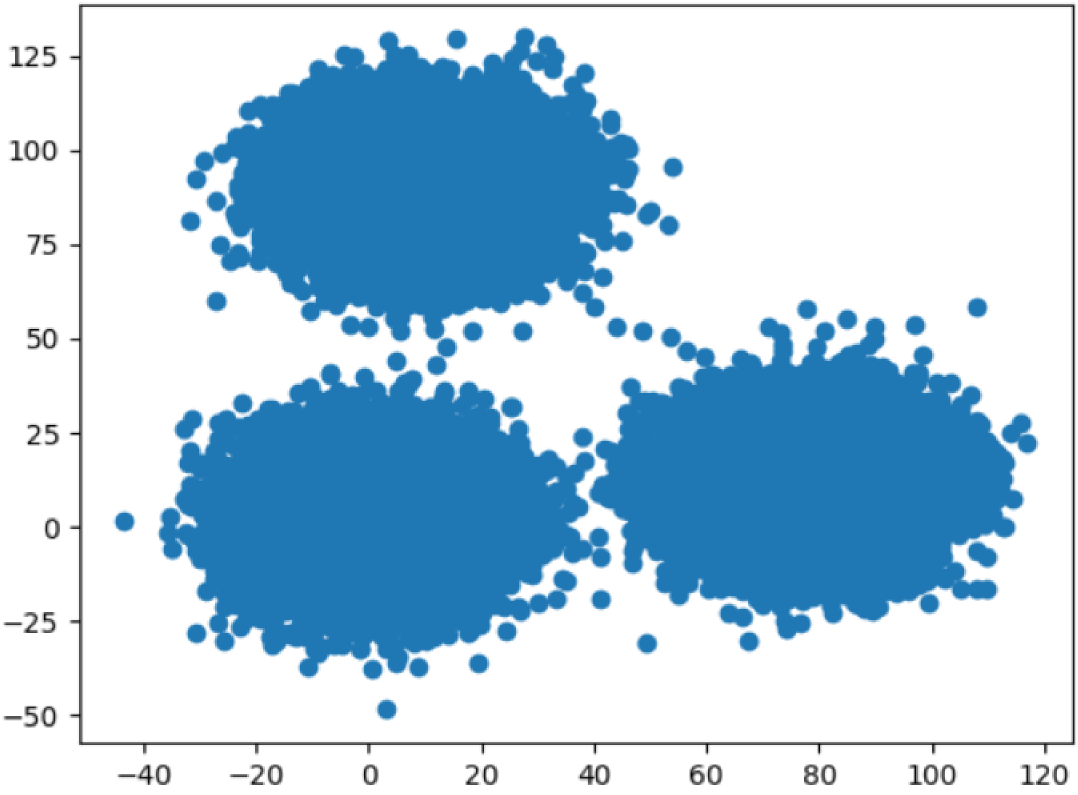
```python
import random
import numpy as np

# Read the coordinates from the centroids.txt file
with open('centroids.txt', 'r') as f:
    x1, y1 = map(float, f.readline().split())
    x2, y2 = map(float, f.readline().split())
    x3, y3 = map(float, f.readline().split())

# The three centers are stored in the variables x1, y1, x2, y2, x3, y3
# These values are read from the centroids.txt file, which contains the x and y coordinates
# of each of the three centers.

```

```
print(”Initial centers:”)
print(f”Center 1: ({x1}, {y1})”)
print(f”Center 2: ({x2}, {y2})”)
print(f”Center 3: ({x3}, {y3})”)
print()

# Print the initial centers to the console for debugging purposes.

n = 2000000
data = []

# The code will generate 2 million (x, y) data points, which will be stored in the data list.

for i in range(n):
    # Generate a random distance with some randomness added
    r = abs(np.random.normal(loc=0, scale=3)) + abs(np.random.normal(loc=0, scale=2))

    # Add the generated distance to one of the three centers chosen randomly
    center = random.choice([(x1, y1), (x2, y2), (x3, y3)])
    x, y = center[0] + r * np.random.normal(loc=0, scale=1), center[1] + r * np.random.normal(loc=0, scale=1)

    # Add the generated (x, y) point to our data list
    data.append((x, y))

# This loop generates 2 million (x, y) data points. For each point, a random distance r is generated using the
# normal distribution with mean 0 and standard deviation 3, with an additional normal distribution with mean 0 and
# standard deviation 2 added to introduce more randomness. Then, one of the three centers is chosen randomly,
# and the generated distance r is added to the x and y coordinates of the center. The resulting (x, y) point is
# added to the data list.

# Save the generated data to a text file
with open(’clustered_data.txt’, ’w’) as file:
    for x, y in data:
        file.write(f”{x}, {y}\n”)

# Finally, the generated data is saved to a text file named ”clustered_data.txt”, with each point on a new line.

print(”Data generation complete”)
print(n, ”data points saved to ’clustered_data.txt’”)

# Print a message indicating that the data generation is complete and how many data points were generated and saved.
```

```
%run data_generator.py

Initial centers:
Center 1: (10.0, 90.0)
Center 2: (1.0, 2.0)
Center 3: (78.0, 12.0)

Data generation complete
2000000 data points saved to ’clustered_data.txt’
```

Preview of the generated data points

```python
# use matplotlib to plot the data
import matplotlib.pyplot as plt

# Read the data from the text file
data = []
with open('clustered_data.txt', 'r') as file:
    for line in file:
        x, y = line.strip().split(',')
        data.append((float(x), float(y)))

# Plot the data
plt.scatter([x for x, y in data], [y for x, y in data])
plt.show()
```



- **Question 3 - Moving Data to HDFS**

  We place the datapoints in the hdfs using the following command

  ```
  hdfs dfs -put clustered_data.txt clustered_data.txt
  ```

  And we also run

  ```
  hdfs dfs -rm -r clustered_data.txt
  ```

  beforehand to remove any old file if it exists.

- **Question 4 - Execute 1 iteration of K-means**

  Utilizing Hadoop Streaming and the code in kmeansiteration.py, one iteration of K-means is executed. In the end, the centroids are printed to the console.

```python
from mrjob.job import MRJob
from mrjob.protocol import RawValueProtocol
import math

class KMeansIteration(MRJob):
    # Method to configure command line arguments
    def configure_args(self):
        super(KMeansIteration, self).configure_args()
        self.add_passthru_arg('--centers', help='comma-separated list of centers')

    # Method to initialize the mapper and load the center values from the command line arguments
    def mapper_init(self):
        centers = list(map(float, self.options.centers.split(',')))
        self.centers = [(centers[i], centers[i+1]) for i in range(0, len(centers), 2)]

    # Mapper method to calculate the closest center for each data point and emit the key-value pair
    def mapper(self, _, line):
        data = list(map(float, line.split(',')))
        distances = [math.sqrt((data[0]-center[0])**2 + (data[1]-center[1])**2) for center in self.centers]
        closest_center_index = distances.index(min(distances))
        yield closest_center_index, data

    # Reducer method to calculate the new center for each cluster and emit the key-value pair
    def reducer(self, key, values):
        new_center = [0.0, 0.0]
        count = 0
        for value in values:
            new_center[0] += value[0]
            new_center[1] += value[1]
            count += 1
        new_center[0] /= count
        new_center[1] /= count
        yield key, new_center

    # Set the input protocol to RawValueProtocol
    INPUT_PROTOCOL = RawValueProtocol


if __name__ == '__main__':
    KMeansIteration.run()
```

```
python kmeans_iteration.py -r hadoop --hadoop-streaming-jar
/home/mainuser/hadoop/hadoop-3.3.5/share/hadoop
/tools/lib/hadoop-streaming-3.3.5.jar
hdfs://localhost:9000/user/mainuser/clustered_data.txt
--centers 10,10,50,70,90,30

No configs specified for hadoop runner
Looking for hadoop binary in /home/mainuser/hadoop/hadoop-3.3.5
```

```
/bin...
Found hadoop binary: /home/mainuser/hadoop/hadoop-3.3.5
/bin/hadoop
Using Hadoop version 3.3.5
Creating temp directory /tmp/kmeans_iteration.mainuser.
20230422.160133.633377
uploading working dir files to hdfs:///user/mainuser/tmp/mrjob
/kmeans_iteration.mainuser.
20230422.160133.633377/files/wd...
Copying other local files to hdfs:///user/mainuser/tmp/mrjob
/kmeans_iteration.mainuser.
20230422.160133.633377/files/
Running step 1 of 1...
  packageJobJar: [/tmp/hadoop-unjar7949111280189398112/] []
  /tmp/streamjob2377951395818703915.jar tmpDir=null
  Connecting to ResourceManager at /0.0.0.0:8032
  Connecting to ResourceManager at /0.0.0.0:8032
  Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/mainuser
  /.staging/job_1682172351424_0025
  Total input files to process : 1
  number of splits:2
  Submitting tokens for job: job_1682172351424_0025
  Executing with tokens: []
  resource-types.xml not found
  Unable to find ’resource-types.xml’.
  Submitted application application_1682172351424_0025
  The url to track the job: http://LAPTOP-E3S2CMV1.:8088
  /proxy/application_1682172351424_0025/
  Running job: job_1682172351424_0025
  Job job_1682172351424_0025 running in uber mode : false
   map 0% reduce 0%
   map 33% reduce 0%
   map 53% reduce 0%
   map 76% reduce 0%
   map 100% reduce 0%
   map 100% reduce 89%
   map 100% reduce 100%
  Job job_1682172351424_0025 completed successfully
  Output directory: hdfs:///user/mainuser/tmp/mrjob
  /kmeans_iteration.mainuser.20230422.160133.633377/output
Counters: 54
File Input Format Counters
Bytes Read=75664139
File Output Format Counters
Bytes Written=126
File System Counters
FILE: Number of bytes read=87660049
FILE: Number of bytes written=176162026
FILE: Number of large read operations=0
FILE: Number of read operations=0
```

```
FILE: Number of write operations=0
HDFS: Number of bytes read=75664351
HDFS: Number of bytes read erasure-coded=0
HDFS: Number of bytes written=126
HDFS: Number of large read operations=0
HDFS: Number of read operations=11
HDFS: Number of write operations=2
Job Counters
Data-local map tasks=2
Launched map tasks=2
Launched reduce tasks=1
Total megabyte-milliseconds taken by all map tasks=56822784
Total megabyte-milliseconds taken by all reduce tasks=18595840
Total time spent by all map tasks (ms)=55491
Total time spent by all maps in occupied slots (ms)=55491
Total time spent by all reduce tasks (ms)=18160
Total time spent by all reduces in occupied slots (ms)=18160
Total vcore-milliseconds taken by all map tasks=55491
Total vcore-milliseconds taken by all reduce tasks=18160
Map-Reduce Framework
CPU time spent (ms)=63050
Combine input records=0
Combine output records=0
Failed Shuffles=0
GC time elapsed (ms)=2406
Input split bytes=212
Map input records=2000000
Map output bytes=83660043
Map output materialized bytes=87660055
Map output records=2000000
Merged Map outputs=2
Peak Map Physical memory (bytes)=506572800
Peak Map Virtual memory (bytes)=2598981632
Peak Reduce Physical memory (bytes)=282497024
Peak Reduce Virtual memory (bytes)=2602565632
Physical memory (bytes) snapshot=1221701632
Reduce input groups=3
Reduce input records=2000000
Reduce output records=3
Reduce shuffle bytes=87660055
Shuffled Maps =2
Spilled Records=4000000
Total committed heap usage (bytes)=1286078464
Virtual memory (bytes) snapshot=7697793024
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
```

```
WRONG_REDUCE=0
job output is in hdfs:///user/mainuser/tmp/mrjob
/kmeans_iteration.mainuser.20230422.160133.633377/output
Streaming final output from hdfs:///user/mainuser/tmp/mrjob
/kmeans_iteration.mainuser.20230422.160133.633377/output...
0 [1.0127311415976215, 2.0063022791532448]
1 [9.998596586236912, 89.99545027492118]
2 [78.01356094767394, 12.009210995449154]
Removing HDFS temp directory hdfs:///user/mainuser/tmp/mrjob
/kmeans_iteration.mainuser.20230422.160133.633377...
Removing temp directory /tmp
/kmeans_iteration.mainuser.20230422.160133.633377...
```

- **Question 5 - Complete k-means**

  Centroids in centroids.txt were manually changed before the execution to demonstrate more iterations.

  Utilizing the code in kmeans.py, the complete K-means algorithm is executed. The centroids are printed to the console after each iteration, as well as the Average Distance from the centroids of the previous iteration.

```python
from kmeans_iteration import KMeansIteration
import math


# Function to run the KMeans algorithm on the input data
def run_kmeans(data_path, centers, threshold=0.01, max_iterations=10):
    # Store the initial centers to check for convergence later
    old_centers = centers

    # Loop through the maximum number of iterations
    for i in range(max_iterations):
        print(f'Iteration {i + 1}')

        # Set up the arguments for the KMeansIteration MRJob
        job_args = [
            data_path,
            '--centers', ','.join(map(str, centers)),
            '-r', 'hadoop',
            '--hadoop-streaming-jar',
            '/home/mainuser/hadoop/hadoop-3.3.5/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar'
        ]

        # Create a new instance of the KMeansIteration MRJob and run it
        job = KMeansIteration(args=job_args)
        with job.make_runner() as runner:
            runner.run()

            # Collect the new centers from the output of the MRJob
            centers = []
            for _, center in job.parse_output(runner.cat_output()):
                centers.extend(center)
                print(center)

        # Check for convergence
        if converged(old_centers, centers, threshold):
            print('Converged')
            break

        # Update old_centers to be the current centers for the next iteration
        old_centers = centers


# Function to check for convergence between the old and new centers
def converged(old_centers, new_centers, threshold):
    total_distance = 0
    for i in range(0, len(old_centers), 2):
        # Calculate the Euclidean distance between the old and new centers
        total_distance += math.sqrt(
            (old_centers[i] - new_centers[i]) ** 2 + (old_centers[i + 1] - new_centers[i + 1]) ** 2)
    average_distance = total_distance / (len(old_centers) / 2)
    print(f'Average distance: {average_distance}')
    print(f'Threshold: {threshold}')
    return average_distance < threshold


if __name__ == '__main__':
    data_path = 'hdfs://localhost:9000/user/mainuser/clustered_data.txt'

    # Read the coordinates from the centroids.txt file
    with open('centroids.txt', 'r') as f:
        x1, y1 = map(float, f.readline().split())
        x2, y2 = map(float, f.readline().split())
        x3, y3 = map(float, f.readline().split())

    initial_centers = [x1, y1, x2, y2, x3, y3]

    # Run the KMeans algorithm
    run_kmeans(data_path, initial_centers)
```

```
python kmeans.py

Iteration 1
No configs specified for hadoop runner
[76.59013482611823, 12.074173840865583]
[1.126995594698583, 1.971656929112909]
[16.461721650478808, 83.91995985391885]
Average distance: 34.435277361403045
Threshold: 0.01
Iteration 2
No configs specified for hadoop runner
[78.00541564910183, 12.007950356032087]
[0.9999430159528663, 2.0018522657735165]
[9.996448832210588, 89.99503318486315]
Average distance: 3.473025362811198
Threshold: 0.01
Iteration 3
No configs specified for hadoop runner
[78.0054829362083, 12.008062334929722]
[1.0000850823919099, 2.0019539984736503]
[9.996395522177313, 89.99523047792616]
Average distance: 0.00016991463780164826
```

```
Threshold: 0.01
Converged
```