

EECS6893 Big Data Analytics

HW1

Lijie Huang
UNI: lh3158
Oct 7th, 2022

1. Question One

Implement iterative K-means in Spark. We've provided you with starter code [kmeans.py](#) on Canvas, which takes care of data loading. Complete the logic inside the for loop, and run the code with different initialization strategies and loss functions. Feel free to change the code if needed, or paste the code into a Jupyter notebook. Take a screenshot of your code and results in your report.

My codes in jupyter notebook are in screenshots below.

```
In [1]: import operator
import sys
from pyspark import SparkConf, SparkContext
import numpy as np
import matplotlib.pyplot as plt
from scipy import linalg
from sklearn.manifold import TSNE

In [2]: # Macros.
MAX_ITER = 20
DATA_PATH = "/input/data.txt"
C1_PATH = "/input/c1.txt"
C2_PATH = "/input/c2.txt"
NORM = 2

In [3]: # Helper functions.
def closest(p, centroids, norm):
    """
    Compute closest centroid for a given point.
    Args:
        p (numpy.ndarray): input point
        centroids (list): A list of centroids points
        norm (int): 1 or 2
    Returns:
        int: The index of closest centroid.
    """
    closest_c = min([(i, linalg.norm(p - c, norm))
                    for i, c in enumerate(centroids)],
                   key=operator.itemgetter(1))[0]
    return closest_c
```

```

eeecs6893hw1-364018 > cluster-hw1
(jupyter kmeans) Last Checkpoint: 34 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 nbdiff

In [4]: def loss(data, centroids, norm):
    """
    param data: original data points in RDD
    param centroids: centroids used to calculate loss
    param norm: int 1 or 2
    return: the loss based on centroids
    """
    norms = data.map(lambda point: linalg.norm(np.subtract(centroids[closest(
        point, centroids, norm=norm)], point), norm) ** norm)
    cost = norms.reduce(lambda norm1, norm2: norm1 + norm2)
    return cost

In [5]: def plot_loss(loss1, loss2):
    fig = plt.figure(figsize=(10, 10))
    plt.plot(range(len(loss1)), loss1, "r", label="cost of c1")
    plt.plot(range(len(loss2)), loss2, "g", label="cost of c2")
    plt.legend(loc="upper right", title="Classes")
    plt.xlabel("Iteration")
    plt.ylabel("Loss")

In [6]: def plot_loss_single(loss, arg):
    fig = plt.figure(figsize=(10, 10))
    plt.plot(range(len(loss)), loss, "r", label="cost of " + arg)
    plt.legend(loc="upper right", title="Classes")
    plt.xlabel("Iteration")
    plt.ylabel("Loss")

In [7]: def plot_cluster(data):
    index = data.map(lambda x: x[0]).collect()
    points = data.map(lambda x: x[1][0]).collect()
    points_embedded = TSNE(n_components=2, perplexity=50, random_state=100).fit_transform(points)
    fig = plt.figure(figsize=(12, 10))
    scatter = plt.scatter(points_embedded[:, 0], points_embedded[:, 1], marker='+', c=index, cmap='jet')
    plt.legend(*scatter.legend_elements(), loc="upper right", title="Classes")

```

```

eeecs6893hw1-364018 > cluster-hw1
(jupyter kmeans) Last Checkpoint: 35 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 nbdiff

def kmeans(data, centroids, norm=2):
    """
    Conduct K-means clustering given data and centroid.
    This is the basic version of k-means, you might need more
    code to record cluster assignment to plot TSNE, and more
    data structure to record cost.

    Args:
        data (RDD): RDD of points
        centroids (list): A list of centroids points
        norm (int): 1 or 2

    Returns:
        RDD: assignment information of points, a RDD of (centroid, (point, 1))
        list: a list of centroids
        and define yourself...
    """
    # iterative k-means
    cost_ls = [loss(data=data, centroids=centroids, norm=norm)]

    for _ in range(MAX_ITER):
        # Transform each point to a combo of point, closest centroid, count=1
        # point -> (closest_centroid, (point, 1))

        # Re-compute cluster center
        # For each cluster center (key), aggregate its values
        # by summing up points and count

        # Average the points for each centroid: divide sum of points by count

        # Use collect() to turn RDD into list
        combo = data.map(lambda point: (closest(point, centroids, norm=norm), (point, 1)))
        centroids = combo.reduceByKey(lambda comb1, comb2: (
            np.add(comb1[0], comb2[0]),
            comb1[1] + comb2[1])).map(lambda x: np.divide(x[0], x[1]))
        centroids = centroids.collect()
        cost = loss(data=data, centroids=centroids, norm=norm)
        cost_ls.append(cost)
        centroids = data.map(lambda point: (closest(point, centroids, norm=norm), (point, 1)))

```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** eeecs6893hw1-364018 > cluster-hw1
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3
- Code Cell (In [9]):**

```

return combo, centroids, cost_ls

In [9]: # Spark settings
conf = SparkConf()
sc = SparkContext(conf=conf)

# Load the data, cache this since we're accessing this each iteration
data = sc.textFile(DATA_PATH).map(
    lambda line: np.array([float(x) for x in line.split(' ')]))
.cache()

# Load the initial centroids c1, split into a list of np arrays
centroids1 = sc.textFile(C1_PATH).map(
    lambda line: np.array([float(x) for x in line.split(' ')]))
.collect()

# Load the initial centroids c2, split into a list of np arrays
centroids2 = sc.textFile(C2_PATH).map(
    lambda line: np.array([float(x) for x in line.split(' ')]))
.collect()

# TODO: Run the kmeans clustering and complete the HW
combo1, centroids1, cost1 = kmeans(data=data, centroids=centroids1, norm=NORM)
combo2, centroids2, cost2 = kmeans(data=data, centroids=centroids2, norm=NORM)

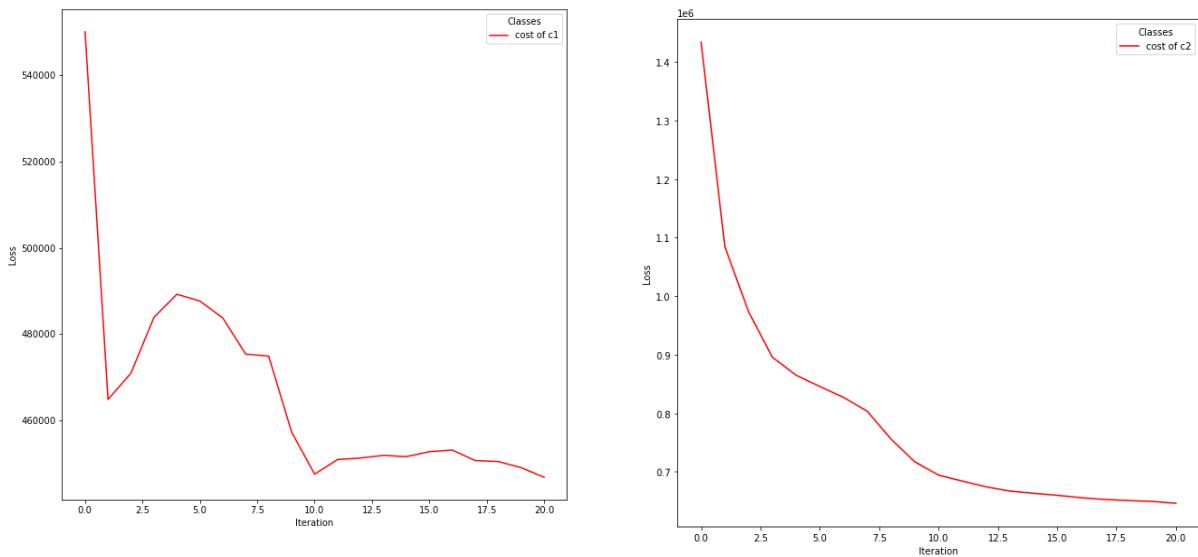
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/09/29 20:36:02 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
22/09/29 20:36:02 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
22/09/29 20:36:02 INFO org.apache.spark.SparkEnv: Registering BlockManagerMasterHeartbeat
22/09/29 20:36:02 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator

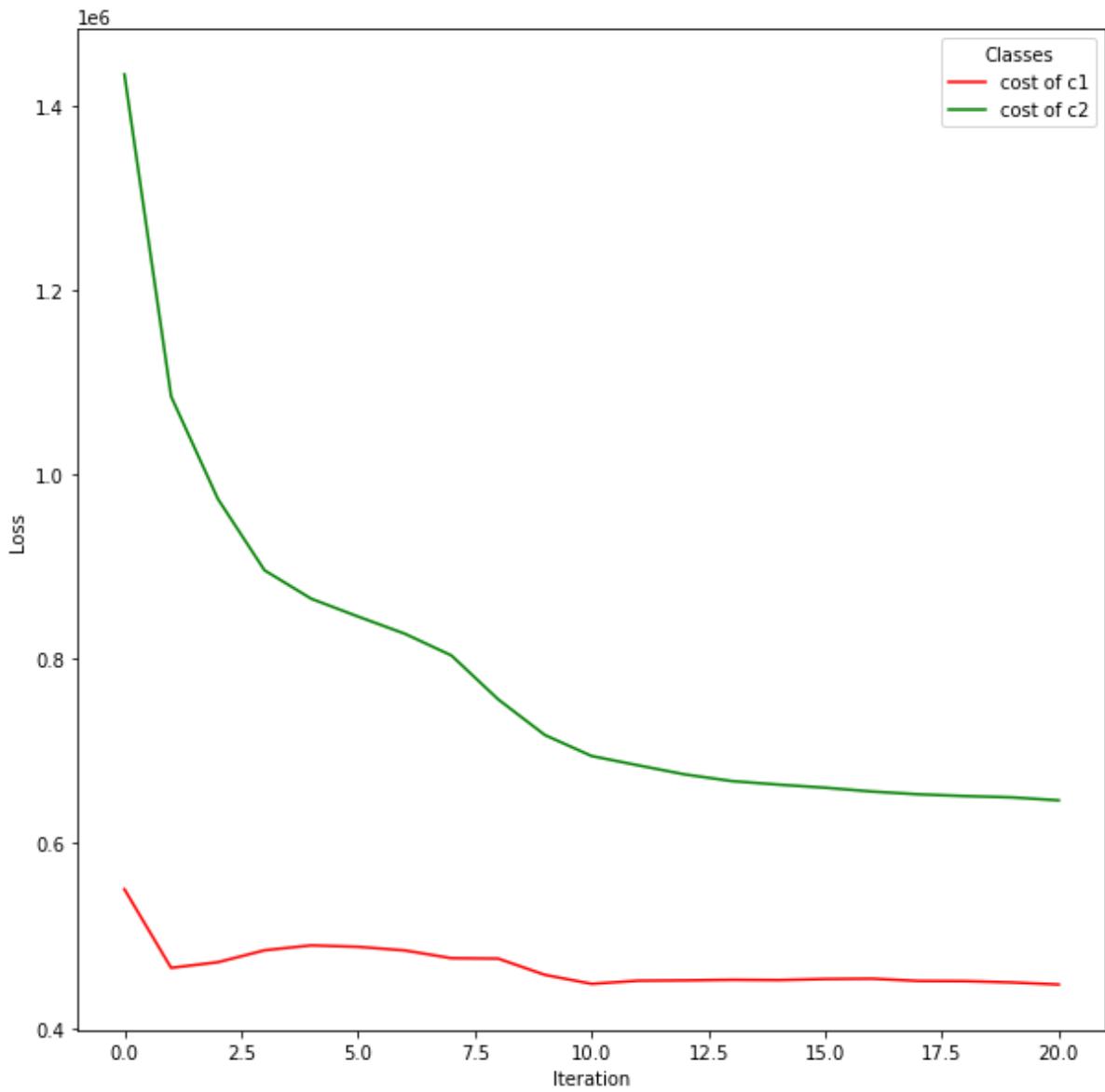
```
- Output Cell (In [10]):**

```
In [10]: plot_loss(cost1, cost2)
```

(1) L1 Within-Cluster Cost

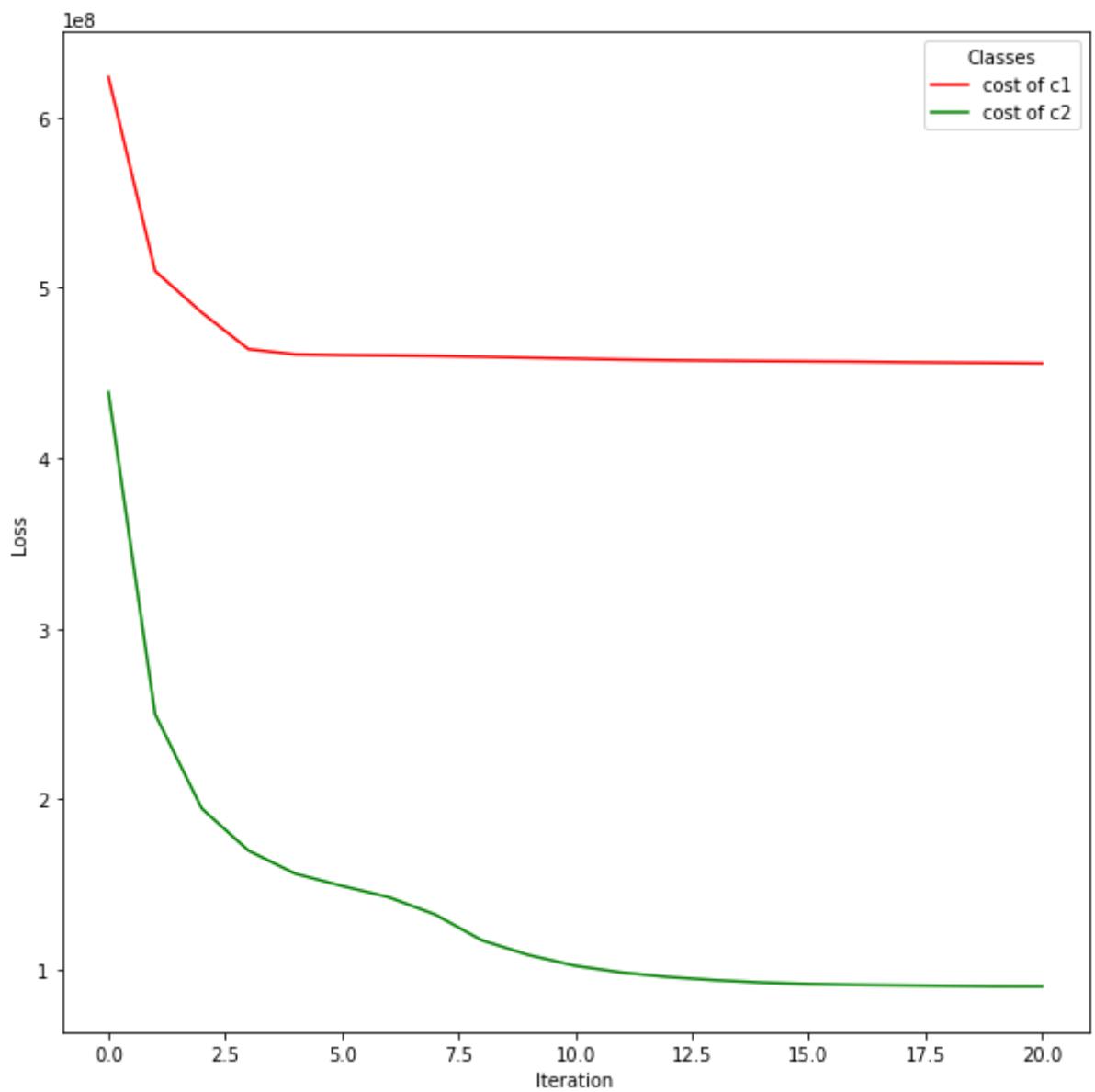
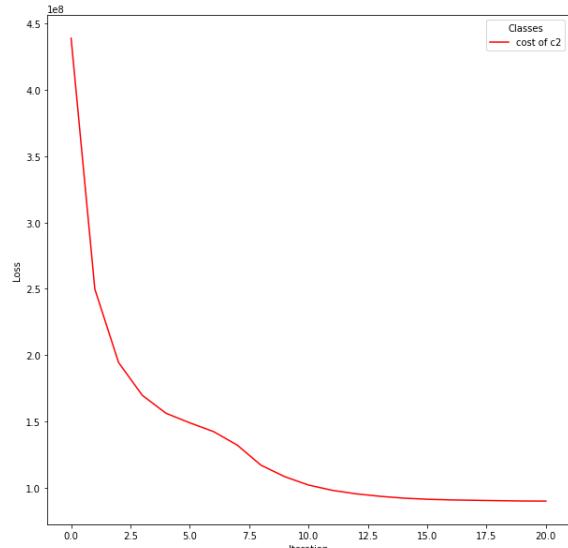
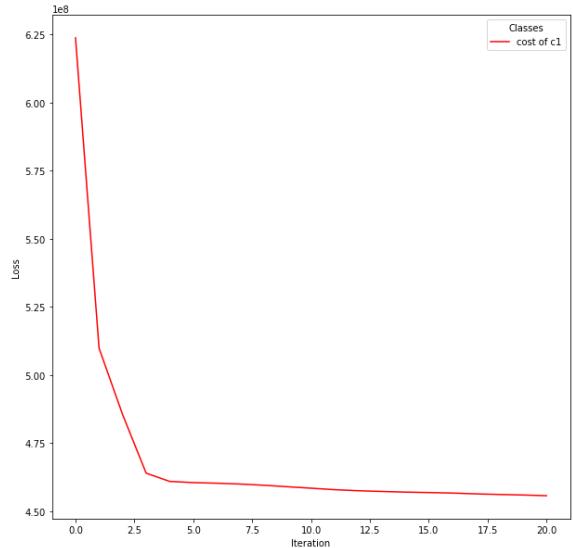
Run clustering on data.txt with c1.txt and c2.txt as initial centroids and use L1 distance as similarity measurement. Compute and plot the within-cluster cost for each iteration. You'll need to submit two graphs here. (20%)





(2) L2 Within-Cluster Cost

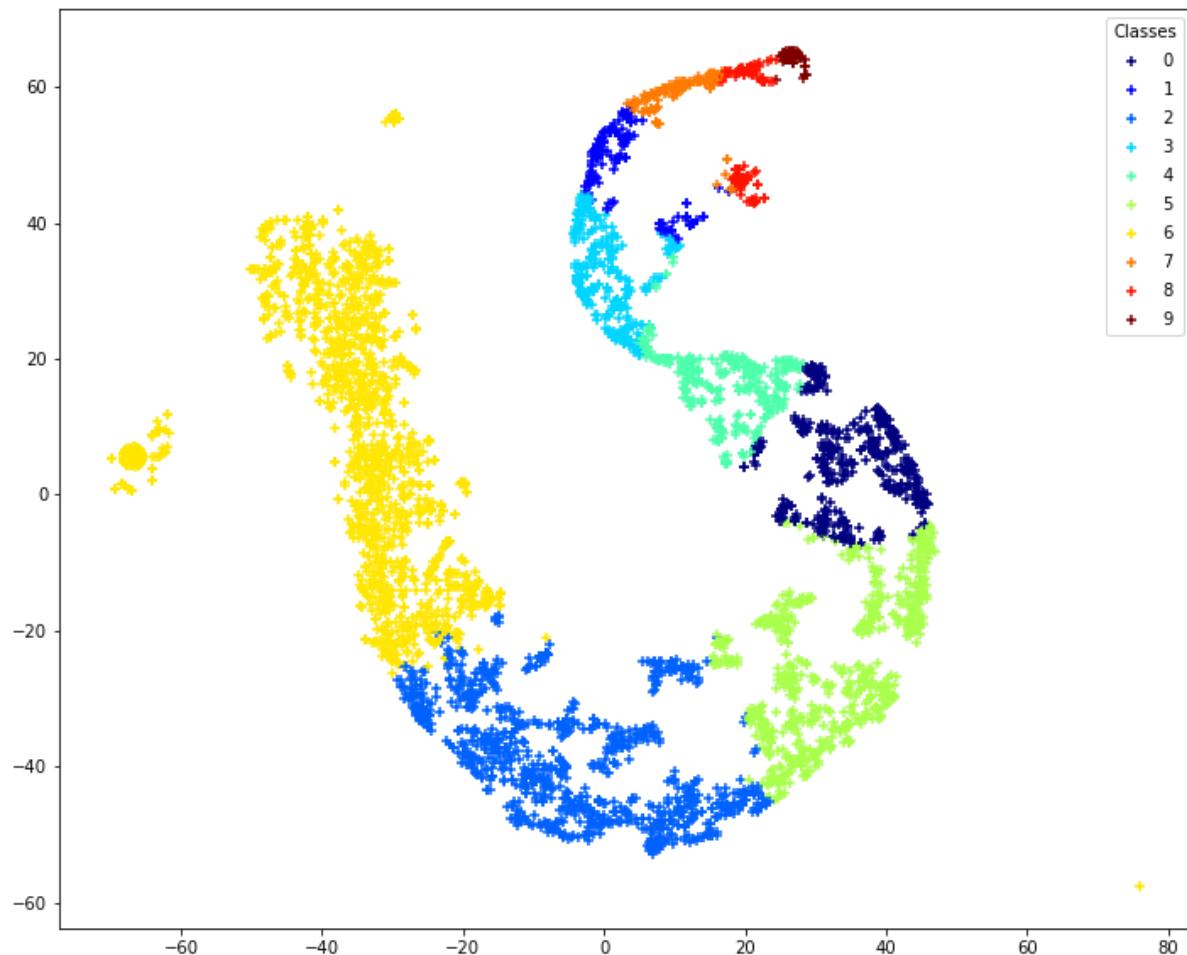
Run clustering on data.txt with c1.txt and c2.txt as initial centroids and use L2 distance as similarity measurement. Compute and plot the within-cluster cost for each iteration. You'll need to submit two graphs here. (20%)



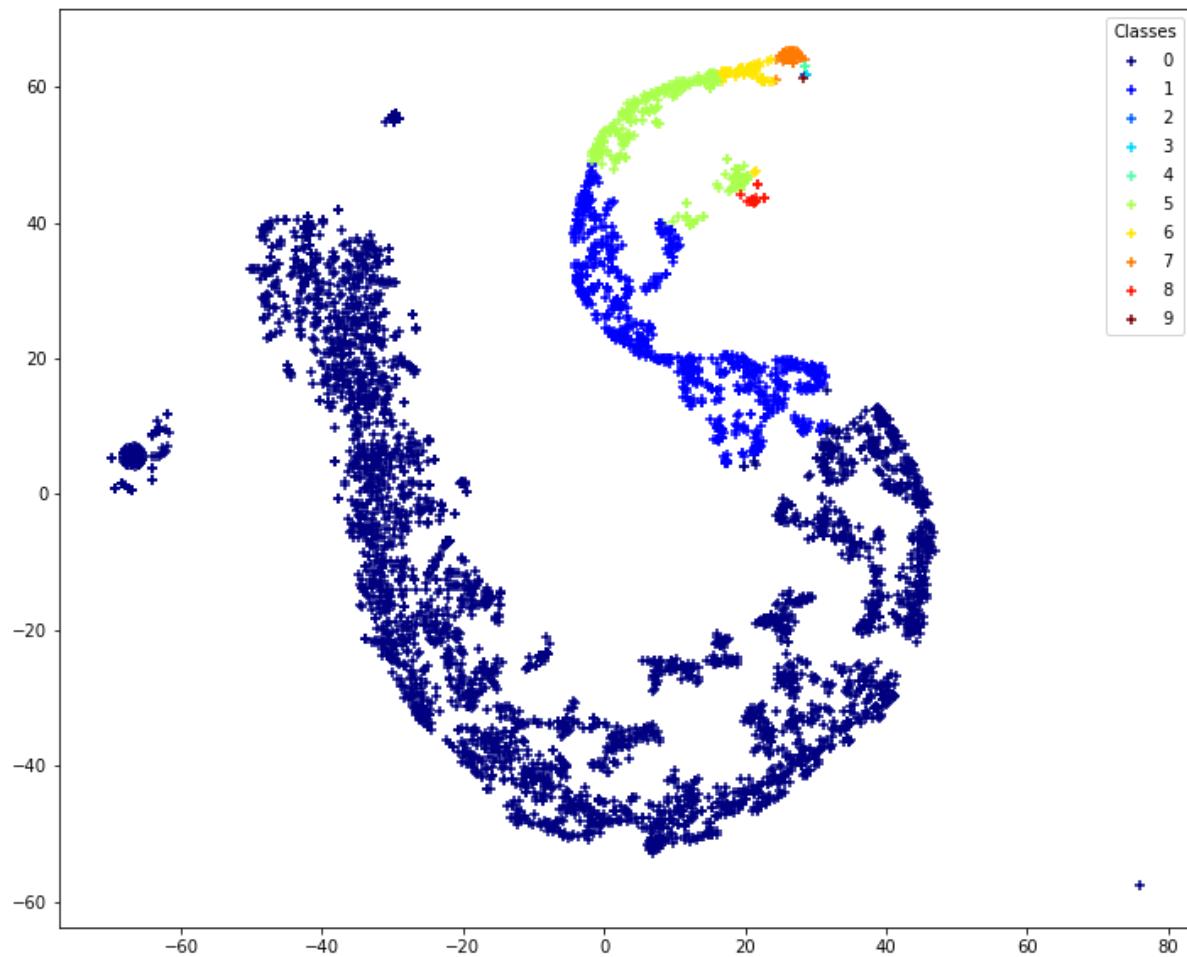
(3) t-SNE Visualization

t-SNE Visualization is a dimensionality reduction method particularly suitable for visualization of high-dimensional data. Visualize your clustering assignment result of (2) by reducing the dimension to a 2D space. You'll need to submit two graphs here. (10%)

c1:



c2:



(4)

For L2 and L1 distance, are random initialization of K-means using c1.txt better than initialization using c2.txt in terms of cost? Explain your reasoning. (5%)

It is clear to tell from the cost figure in (2) that initialization using c2.txt can reduce the cost rapidly and remains at a lower level for L2 distance. Although the order of lines in figures in (1) is the opposite, it is due to the fact that using L1 performs worse than using L2 as the distance function. The reason why initialization using c2.txt is better is that making origin centroids as far away to each other as possible makes the iteration to update centroids much more easier and quickly. While using random selected centroids may end in letting outliers weight too much to update centroids thus making the converge of cost function slow and leading to inappropriate centroids.

(5)

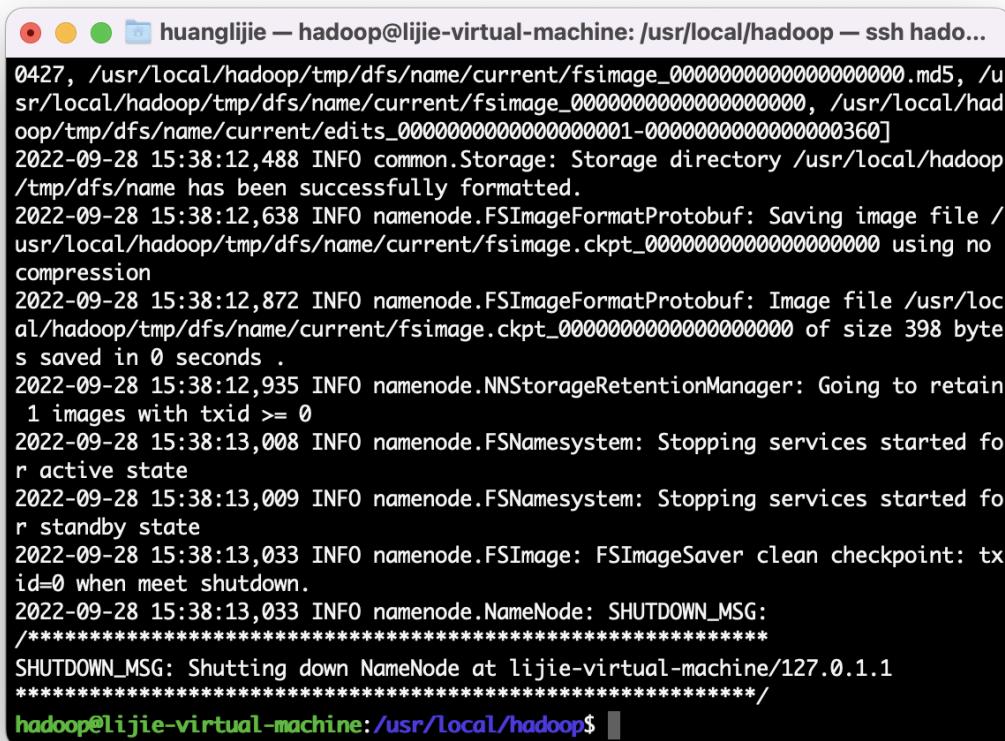
What is the time complexity of the iterative K-means? (5%)

The time complexity is $O(i * k * n * d)$, where i is the number of iterations, k is the number of centroids, n is the total number of all points, d is the dimension.

2. Question2

(1) Download and setup Hadoop

Set up the namenode



hadoop@lijie-virtual-machine: /usr/local/hadoop — ssh hado...

```
0427, /usr/local/hadoop/tmp/dfs/name/current/fsimage_00000000000000000000.mds, /usr/local/hadoop/tmp/dfs/name/current/fsimage_00000000000000000000, /usr/local/hadoop/tmp/dfs/name/current/edits_00000000000000000001-000000000000000360]
2022-09-28 15:38:12,488 INFO common.Storage: Storage directory /usr/local/hadoop/tmp/dfs/name has been successfully formatted.
2022-09-28 15:38:12,638 INFO namenode.FSImageFormatProtobuf: Saving image file /usr/local/hadoop/tmp/dfs/name/current/fsimage.ckpt_00000000000000000000 using no compression
2022-09-28 15:38:12,872 INFO namenode.FSImageFormatProtobuf: Image file /usr/local/hadoop/tmp/dfs/name/current/fsimage.ckpt_00000000000000000000 of size 398 bytes saved in 0 seconds .
2022-09-28 15:38:12,935 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2022-09-28 15:38:13,008 INFO namenode.FSNamesystem: Stopping services started for active state
2022-09-28 15:38:13,009 INFO namenode.FSNamesystem: Stopping services started for standby state
2022-09-28 15:38:13,033 INFO namenode.FSImage: FSImageSaver clean checkpoint: tx id=0 when meet shutdown.
2022-09-28 15:38:13,033 INFO namenode.NameNode: SHUTDOWN_MSG:
***** SHUTDOWN_MSG: Shutting down NameNode at lijie-virtual-machine/127.0.1.1
*****
```

hadoop@lijie-virtual-machine:/usr/local/hadoop\$

verify hadoop dfs

```
huanglijie — hadoop@lijie-virtual-machine: /usr/local/hadoop — ssh hado...
[hadoop@lijie-virtual-machine:/usr/local/hadoop$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [lijie-virtual-machine]
[hadoop@lijie-virtual-machine:/usr/local/hadoop$ ]
```

verify yarn scrpit

```
huanglijie — hadoop@lijie-virtual-machine: /usr/local/hadoop — ssh hado...
Starting secondary namenodes [lijie-virtual-machine]
[hadoop@lijie-virtual-machine:/usr/local/hadoop$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
[hadoop@lijie-virtual-machine:/usr/local/hadoop$ ]
```

(2) HDFS metrics monitoring

- Monitor HDFS metrics through HTTP API. Provide a screenshot.

Overview 'localhost:9000' (✓active)

Started:	Wed Sep 28 15:43:55 -0400 2022
Version:	3.3.1, ra3b9c37a397ad4188041dd80621bdeefc46885f2
Compiled:	Tue Jun 15 01:13:00 -0400 2021 by ubuntu from (HEAD detached at release-3.3.1-RC3)
Cluster ID:	CID-1562cf11-596e-47e1-bd63-8d7425baa327
Block Pool ID:	BP-931567434-127.0.1.1-1664394220965

Summary

Security is off.
Safemode is off.
1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).
Heap Memory used 83.95 MB of 228.5 MB Heap Memory. Max Heap Memory is 869.5 MB.
Non Heap Memory used 47.06 MB of 48.71 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	19.56 GB
Configured Remote Capacity:	0 B
DFS Used:	24 KB (0%)

Datanode Information

Datanode usage histogram

In operation

DataNode State	All	Show 25 entries	Search:						
Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Version
✓/default-rack/jie-virtual-machine:9866 (127.0.0.1:9866)	http://jie-virtual-machine:9864	2s	0m	24 KB	8.48 GB	19.56 GB	0	24 KB (0%)	3.3.1

- Select five metrics that you think are most important. Explain their usages and why they are important.

- 1) CapacityRemaining. it represents the total available capacity remaining across the entire HDFS cluster. Running out of disk space in HDFS will cause bad things to happen. Any running jobs which write out of memory data may fail due to lack of capacity.
- 2) MissingBlocks. A missing block is far worse than a corrupt block, because a missing block cannot be recovered by copying a replica. A missing block represents a block for which no known copy exists in the cluster. That's not to say that the block no longer exists—if a series of DataNodes were taken offline for maintenance, missing blocks could be reported until they are brought back up.
- 3) VolumeFailureTotal. Though a failed volume will not bring your cluster to grinding halt, you most likely want to know when hardware failures occur, if only so that you can replace the failed hardware.
- 4) NumDeadDataNodes. The death of a DataNode causes a flurry of network activity, as the NameNode initiates replication of blocks lost on the dead nodes. Though the loss of a single DataNode may not impact performance much, losing multiple DataNodes will start to be very taxing on cluster resources, and could result in data loss.
- 5) Remaining: If left uncertified, a single DataNode running out of space could quickly cascade into failures across the entire cluster as data is written to an increasingly-shrinking pool of available DataNodes. Tracking this metric over time is essential to maintain a healthy cluster

(3) MapReduce counters monitoring

- Collect MapReduce counters related information through the web UI.
Provide a screenshot.

MapReduce Job
job_1664394282358_0001

Job Overview

Job Name:	QuasiMonteCarlo
User Name:	hadoop
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Wed Sep 28 15:48:09 EDT 2022
Started:	Wed Sep 28 15:48:24 EDT 2022
Finished:	Wed Sep 28 15:48:59 EDT 2022
Elapsed:	34sec
Diagnostics:	
Average Map Time	21sec
Average Shuffle Time	7sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Wed Sep 28 15:48:14 EDT 2022	ljiie-virtual-machine:8042	logs

Task Type	Total	Complete
Map	5	5
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	5
Reduces	0	0	1

Counters for job_1664394282358_0001

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	116	116
	FILE: Number of bytes written	1,367,320	273,497	1,640,817
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	1,330	0	1,330
Job Counters	HDFS: Number of bytes read erasure-coded	0	0	0
	HDFS: Number of bytes written	0	215	215
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	20	5	25
	HDFS: Number of write operations	0	3	3
	Total time spent by all map tasks (ms)	0	0	106,351

- Describe how to monitor the execution of MapReduce tasks through related metrics. (5%)

For jobs counters. MILLIS_MAPS/MILLIS_REDUCES tracks the processing time for maps/reduces. NUM_FAILED_MAPS/NUM_FAILED_REDUCES tracks number of failed maps/reduces.

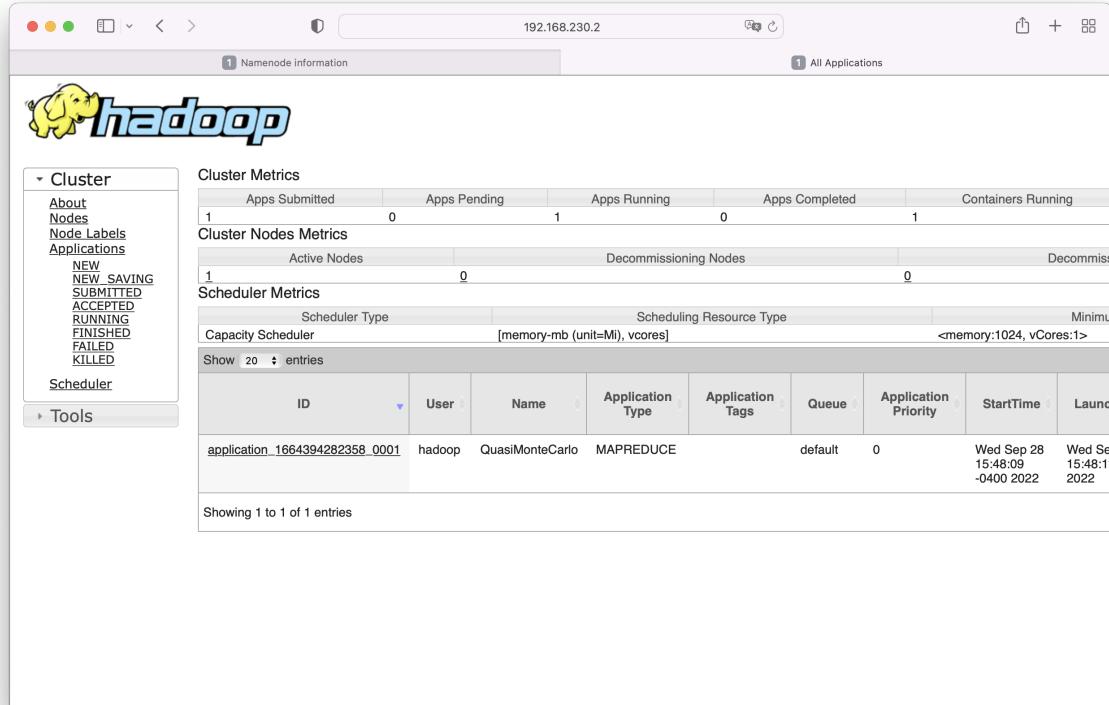
RACK_LOCAL_MAPS/DATA_LOCAL_MAPS/OTHER_LOCAL_MAPS tracks where map tasks were executed.

For task counters. REDUCE_INPUT_RECORDS keeps an eye on the number of input records for reduce tasks. SPILLED_RECORDS is the number of records spilled to disk. GC_TIME_MILLS shows the processing time spend in garbage collection.

For custom counters, Custom counters can be used to track more fine-grained counts, such as counting the number of malformed or missing records.

(4) YARN metrics monitoring

- Monitor YARN metrics through HTTP API. Provide a screenshot in your report.



The screenshot shows the Hadoop Namenode Information web interface. The top navigation bar includes icons for back, forward, and search, along with the IP address 192.168.230.2 and a refresh button. Below the bar, there's a title bar with the text "Namenode Information" and a "All Applications" link. The main content area features a large yellow elephant icon followed by the word "hadoop". On the left, a sidebar menu has "Cluster" expanded, showing "About", "Nodes", "Node Labels", and "Applications" with sub-options: NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, and Scheduler. A "Tools" option is also present. The main panel displays "Cluster Metrics" with tables for "Cluster Metrics", "Cluster Nodes Metrics", and "Scheduler Metrics". The "Scheduler Metrics" table shows a single entry for "Capacity Scheduler" with a "Scheduler Type" of "[memory-mb (unit=Mi), vcores]" and a "Minimum Resource Allocation" of "<memory:1024, vCores:1>". Below these tables is a table titled "Show 20 entries" with columns: ID, User, Name, Application Type, Application Tags, Queue, Application Priority, Start Time, and Launch Time. One row is visible: application_1664394282358_0001, hadoop, QuasiMonteCarlo, MAPREDUCE, default, 0, Wed Sep 28 15:48:09 -0400 2022, and Wed Sep 28 15:48:11 2022. At the bottom, it says "Showing 1 to 1 of 1 entries".

- Select five metrics that you think are most important from the returned json. Explain their usages and why they are important.

- 1) unhealthyNodes. Once a node is marked unhealthy, other nodes have to fill the void, which can cause a cascade of alerts as node after node nears full disk utilization.
- 2) activeNodes/lostNodes. The activeNodes metric tracks the current count of active, or normally operating, nodes. This metric should remain static in the absence of anticipated maintenance. If a NodeManager fails to maintain contact with the ResourceManager, it will eventually be marked as “lost” and its resources will become unavailable for allocation.
- 3) appsFailed. if the percentage of failed map or reduce tasks exceeds a specific threshold the application as a whole will fail.
- 4) progress. tracking progress alongside other metrics can better help you to determine the cause of any performance degradation.
- 5) containersFailed. This tracks the number of containers that failed to launch on that particular NodeManager. If the NodeManager’s disk is full, any container it launches will fail.