BIG DATA ANALYSIS: MAPREDUCE

Q1. Descriptive Statistics

MapReduce in Hadoop was used to calculate the following statistics: maximum, minimum, mean, median, variance and standard deviation of hourly 'Dry Bulb' temperatures. These statistics were calculated for the month of April 2007 for all weather stations.

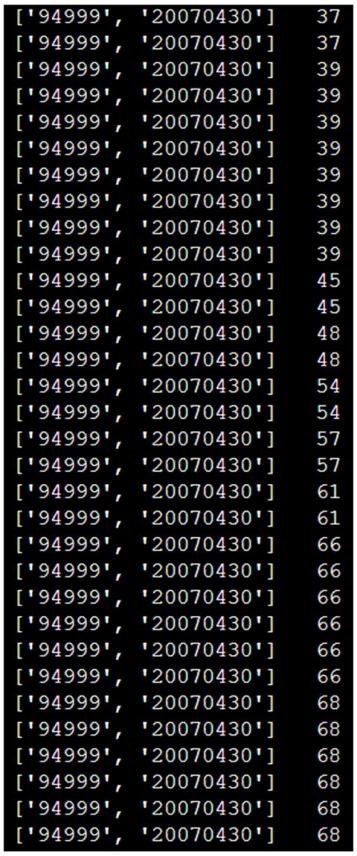
One mapper file and five reducer files were created to output the various statistics. The output of this mapper file, mapper_temp.py formed the input for all five reducer files. Pseudocodes for all six files have also been included in the appendix.

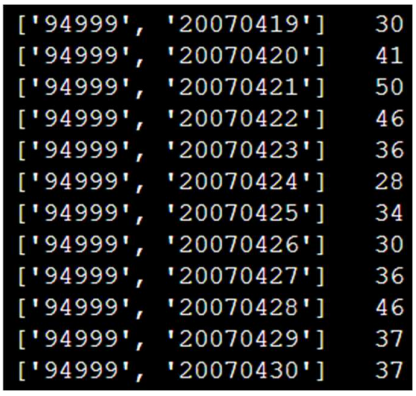
The mapper pseudo code explains how to output year, value pairs for all weather stations for the month of April 2007.

mapper_temp.py

- create a mapper file in Python that will output all daily temperatures by hour for different weather stations, April 2007
- make it possible to execute a script file with Python
- import the sys module in Python
- iterate through the input file until end.
- a copy of the string with the trailing characters removed is displayed.
- a list of the words using comma as a delimiter in the string is returned.
- separate the strings into keys and values.
- the key will include the weather station ID (Wban Number) and the date (YearMonthDay)
- the value is the temperature for the key (Dry Temp Bulb)
- create a 'for loop' which will skip over pass the temperature values with the character '-'
- try and except statement to convert values to integer format

The example below shows the mapper output for the hourly minimum temperatures for each station on the left and the reducer output from this mapper on the right. Data is sorted by weather station, date, and temperature. The mapper was tested separately, then the mapper and reducer tested together as seen below.





Q2. Cluster Analysis using Apache Mahout.

K-Means clustering is a machine learning algorithm that partitions data into k clusters, it's simple and is an unsupervised method of learning. For this dataset, four distance measures were used to compare the inter and intra cluster scores. K clusters of 3, 5, 8, 10, 12, 15, 20, 25 and 30 were implemented and compared.

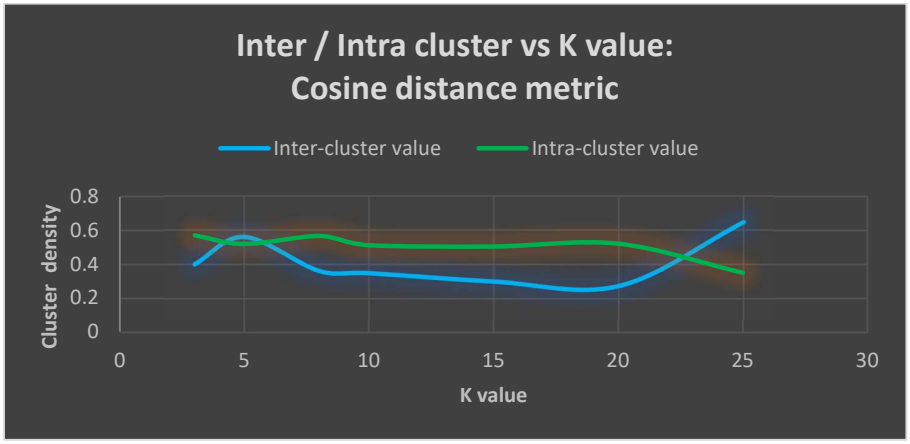
A useful means of determining cluster quality is the inter-cluster density. The further apart the clusters are from each other, the more distinct and the higher the quality of the cluster. The inter-cluster centroids should therefore have a low density or be highly spread apart. Centroids that are further apart indicate better quality clusters.

Intra-cluster density should also be maximised if possible so that distances between samples and their centroids are lower. Intra-cluster density gives an indication of how well similar samples are brought together within a cluster.

Cosine results and graphs:

Distance metric	K-value	Inter-cluster value	Intra-cluster value
Cosine	3	0.402260512	0.570773306
	5	0.562165302	0.520273926
	8	0.363533355	0.567282408
	10	0.351055284	0.513338531
	15	0.302377862	0.505415292
	20	0.276322128	0.521819051
	25	0.648412201	0.350125121

Tabulated results show that a K-value of 20 gives the highest inter-cluster density score for the Cosine distance metric. It's corresponding intra-cluster score is 0.521. A low inter-cluster density score indicates that the clusters are of high quality. Higher intra-cluster density scores give an indication of samples being more similar within the clusters. At around K = 20, the following graph shows a significant increase in inter-cluster density and a corresponding decrease in intra-cluster density. When running K-means on K >= 27, intra-cluster density produced NaN (not a number) values; this means that the K value must be less than the number of input files. The number of input files in this clustering process is 27.



Quality clustering is also influenced by the distance measure chosen. Cosine, Tanimoto (also known as extended Jaccard measure), squared Euclidean and Manhattan were used to compare the inter and intra-cluster distance measures. Distance measures should be chosen in relation to the data to be analysed. 27 books with long texts were analysed in this clustering process.

Cosine similarity is a measure best suited to finding duplication, whereas Jaccard similarity looks at uniqueness. Cosine uses inverse document frequency (IDF) and term frequency (TF) in its approach when generating its vectors. This can be seen in the K-means algorithm.

Cosine distance has been found to be a better distance measure for analysing text. Grouping documents by common words with the highest weighting is a feature of the Cosine distance. Vectors weighted using TF-IDF giver a higher weight to more important words, which allows a higher average weight to be given to the centroid vector. Lower weight is subsequently given for less important words or stop-words. Cosine distance is also known as cosine similarity.

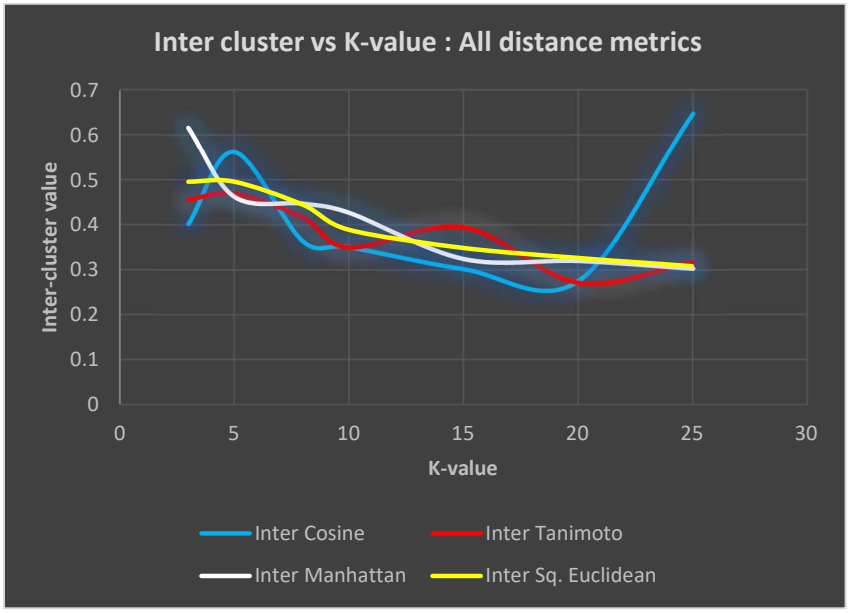
In comparison, distance values for the Euclidean measure are larger than Cosine distance measure. Comparing Euclidean and Cosine measures is only useful if both are scaled to the same level or normalized, in this case they are essentially the same. Centroid pairs that are scaled by the inter-cluster average are a more reliable measure of determining the quality of clusters. Euclidean distance is a measure that calculates the shortest distance between any two points.

Manhattan distance is used for regression analysis, compressed sensing and frequency distribution. It is a special case of Minkowski distance.

At a K-value of 20 when comparing the inter-cluster density of the four-distance metrics, it appears that Cosine or Tanimoto have slightly lower values than either squared Euclidean or Manhattan. Values for Tanimoto, Squared Euclidean and Manhattan are listed below.

Distance metric	K-value	Inter-cluster value	Intra-cluster value
Tanimoto (Jaccard)	3	0.456236655	0.578354115
	5	0.469595624	0.61912456
	8	0.416644984	0.591163294
	10	0.350242263	0.564543229
	15	0.394720928	0.530100971
	20	0.27193208	0.495688783
	25	0.318290743	0.593992876
Squared Euclidean	3	0.496517277	0.585662215
	5	0.496517277	0.585662215
	8	0.445224964	0.640070675
	10	0.389373091	0.596006811
	15	0.348822645	0.518160206
	20	0.32611464	0.56807503
	25	0.307985326	0.483333333
Manhattan	3	0.616491543	0.592744095
	5	0.462610686	0.560357039
	8	0.44752124	0.509245168
	10	0.427214782	0.560350618
	15	0.324902748	0.515349767
	20	0.320618876	0.516748952
	25	0.303457482	0.483333333

There is a general trend in all distance metrics that with an increase in K, there is a corresponding decrease in inter-cluster density (or higher distance). After around K = 20 there is no significant decrease in inter-cluster density. Cosine shows the lowest inter-cluster density at a value just below K = 20. The lowest inter-cluster density for either squared Euclidean or Manhattan measures indicate that a K value of 25 would be more appropriate to obtain the lowest inter-distance density. When running the clustering using the Squared Euclidean metric, it was necessary to alter the convergence threshold from the default of 0.5 to a value of 1.0. For the other metrics a value of 0.1 was used.



Automating the clustering procedure:

To partially automate the clustering process by distance metric, two script files were created in Python and run in the terminal window.

The file was made executable using the command: **chmod +x clustering_script.py**

The two script files for cosine distance metric with **k = 20** can be found below. If the wildcard option were to be used, it would be possible to fully automate the clustering process. This process was attempted but was unsuccessful.

Because of this partial automation and the fact that the output file from the k-means clustering changes randomly, the command: **hadoop fs -ls docs-kmeans-clusters** needs to be run between the two scripts to determine the input file for the clusterdump command. The file name will therefore need to be altered if it has changed from the previous run. The output file can also be changed, if need be, though this is not a necessity. If not altered, the original file will be overwritten. It was decided to change this file name with each run.

The output displays the inter and intra-cluster density values at the tail-end (last few lines) of text file. If a change in either distance metric or k-value is required, the scrip file will need to be altered to incorporate this. An example of two script files and a summary of the commands can be found in the appendix.

CONCLUSION:

MapReduce is a simple algorithm used to analyse Big Data. The map task takes input data sets and converts them to the reducer as shuffled and sorted key-value pairs. The reducer then uses these output data sets as an input and combines them into key-value pairs. MapReduce can be used to calculate statistics and for clustering in a distributed, independent manner. Finding similarities between items using distance metrics is easily achieved by the K-means algorithm. After altering the K-value and the distance metrics, results indicated that Cosine was the better metric in terms of inter-cluster density values. The choice of distance metric affects cluster quality, so this should be tailored to the input data. In the case of text, Cosine or Tanimoto proved to be more effective measures in comparison to either Euclidean or Manhattan. Although MapReduce is an effective means of processing huge

datasets on disk, some limitations include the face that it is not able to effectively compute iterative processing or for Machine Learning or streaming. MapReduce also relies on data sharing

BIBLIOGRAPHY:

Brownlee, J. (202, August 19). *4 Distance Measures for Machine Learning*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/distance-measures-for-machine-learning/

Chowdam Sreedhar, N. K. (2017). Clustering large datasets using K-means modified inter and intra clustering (KM-12C) in Hadoop. *Journal of Big Data*, 1 - 19.

K.Thirumoorthy, K. M. (2018). Unstructured Text Document Clustering Text mining methods. *International Journal of Pure and Applied Mathematics*, 8.

Manhattan distance [Explained]. (n.d.). Retrieved from iq.opengenus.org: https://iq.opengenus.org/manhattan-distance/

MapReduce in Big Data. (n.d.). Retrieved from HKR Trainings: https://hkrtrainings.com/mapreduce-in-big-data

Navlani, A. (2021, June 4). *Text Similarity Measures: Cosine and Jaccard Similarity*. Retrieved from machine learning geek.com: https://machinelearninggeek.com/text-similarity-measures/

TechVidvan. (n.d.). *Techvidan.com*. Retrieved from K-Means Clustering in Machine Learning: https://techvidvan.com/tutorials/machine-learning-k-means-clustering/

Wei, W. (2020, February 16). *github.com*. Retrieved from github.com: https://github.com/wenkangwei/Distributed-Computing-Hadoop-Pyspark/blob/be0fe02cd8aa39efe55e3569ac1eda8050f1e1d9/hw4/RatingMapperReducers.ipynb

APPENDIX:

Q1: Pseudo Code

Mapper

```
def mapper {
key: text = field 1 and field 2, value: int
emit (key, value)
}
```

Reducers: descriptive statistics (minimum, maximum, mean, median, variance)

Minimum:

```
def reducer {
key: text, value: int
        calculate min (value)
emit (key, min value)
```

```
}
```

The following reducer pseudo code shows how to output the daily minimum temperatures for the data from the mapper. All reducers use the same mapper file to output the required results.

reducer_min_temp.py

- create a reducer file in Python that will output minimum daily temperatures for different weather stations, April 2007
- make it possible to execute a script file with Python
- import the sys module in Python
- assign None (for character) and 0 (for number) to new tuple variables (new_key) and (min_val)
- iterate through the standard input until end of the file is reached (file = mapper_temp.py)
- the input file will be split by the tab delimiter, \t
- create an 'if else' statement to either print the key, value pairs as new tuples or skip them
- keys or values with None or 0 will not form part of the output key value pairs
- minimum values will be calculated for those values forming part of the key that do not contain None or 0
- if the minimum values are calculated, the key values pairs will be printed

Maximum:

```
def reducer {
key: text, value: int
        max (value)
emit (key, max value)
}
```

Mean:

```
def reducer {
key: text, value: int
        sum (value)
        len (value)
        mean = sum/count
emit (key, mean)
}
```

Median:

```
def reducer {
key: text, value: int
        index = len (value) //2
        median = (value[index])
emit (key, median)
}
```

Variance:

```
def reducer {
key: text, value: int
        len = len (value)
        mean = sum(value) / len(value)
        variance = sum[(value – mean)**2] / len
emit (key, variance)
}
```

Mapper_temp.py

```
#!/usr/bin/env python
"""mapper_temp.py"""
import sys
for line in sys.stdin:
    # import data from file: 200704hourly.txt and select columns station and date (1 and 2) for key, column dry bulb (9) for val
    val = line.rstrip('r\n').split(',')
    key, val = (val[0:2], val[8])
    for x in val:
        if x =='-':
            pass
        try:
            val = int(val)
        except ValueError:
            pass
        else:
            print ("%s\t%s" % (key, val))
```

reducer_min_temp.py

```
#!/usr/bin/env python
"""reducer_min_temp.py"""
import sys
(new_key, min_val) = (None, 0)
for line in sys.stdin:
# mapper file mapper_temp.py : Dry Bulb temperatures by station and date for the month of April 2007
    (key, val) = line.strip().split("\t")
    if new_key and new_key != key:
        print ("%s\t%s" % (new_key, min_val))
        # print the maximum value of temperature
        (new_key, min_val) = (key, int(val))
```

```
    else:
        (new_key, min_val) = (key, min(min_val, int(val)))
if new_key:
    print ("%s\t%s" % (new_key, min_val))
```

reducer_max_temp.py

```
#!/usr/bin/env python
"""reducer_max_temp.py"""
import sys
(new_key, max_val) = (None, 0)
for line in sys.stdin:
# mapper file mapper_temp.py : Dry Bulb temperatures by station and date for the month of April 2007
    (key, val) = line.strip().split("\t")
    if new_key and new_key != key:
        print ("%s\t%s" % (new_key, max_val))
        (new_key, max_val) = (key, int(val))
    else:
        # print the maximum value of temperature
        (new_key, max_val) = (key, max(max_val, int(val)))
if new_key:
    print ("%s\t%s" % (new_key, max_val))
```

reducer_mean_temp.py

```
#!/usr/bin/env python
"""reducer_mean_temp.py"""
import sys
temp_num = 0
temp_total = 0
new_key = None
for line in sys.stdin:
# mapper file mapper_temp.py : Dry Bulb temperatures by station and date for the month of April 2007
    data = line.strip().split('\t')
    if len(data) != 2:
        continue
    key, val = data
    if new_key and new_key != key:
        print ('%s\t%s' % (new_key, temp_total/temp_num))
        new_key = key
        temp_num = 0
```

```
        temp_total = 0
    new_key = key
    temp_num += 1
    temp_total += float(val)
if new_key != None:
    print ('%s\t%s' % (new_key, temp_total/temp_num))
```

reducer_median_temp.py

```
#!/usr/bin/env python
"""reducer_median_temp.py"""
import sys
new_key = None
all_temp = []
for line in sys.stdin:
# mapper file mapper_temp.py : Dry Bulb temperatures by station and date for the month of April 2007
    line = line.strip()
    key, val = line.split("\t", 2)
    try:
        if  len(all_temp)>0 and  new_key != key:
            # Find median of temp(val) list of one key(new_temp)
            index = len(all_temp)//2
            temp_median = all_temp[index]
            print("%s\t%f" % ( new_key, temp_median))
            all_temp.clear()
        val = float(val)
        all_temp.append(val)
        new_key = key
    except ValueError:
        continue
#Check the last key (new_key) in list and print its median (temp_median)
if  len(all_temp)>0:
    index = len(all_temp)//2
    temp_median = all_temp[index]
    print("%s\t%f" % ( new_key, temp_median))
```

reducer_var_temp.py

```
#!/usr/bin/env python
"""reducder_var_temp.py"""
import sys
```

```
import math
new_key = None
new_val = []
for line in sys.stdin:
# mapper file mapper_temp.py : Dry Bulb temperatures by station and date for the month of April 2007
    line = line.strip()
    key, val = line.split("\t", 2)
    try:
        if  len(new_val)>0 and new_key != key:
            # Find variance of temp(val) list of one key(station and date)
            mean_val = sum(new_val)/len(new_val)
            val_ls = [(val - mean_val)**2 for val in new_val]
            std_val = math.sqrt(sum(val_ls)/len(val_ls))
            print("%s\t%f" % ( new_key, val_ls))
            # reset the list of val(temperatures) of current key (new_val)
            new_val.clear()
        val = float(val)
        new_val.append(val)
        new_key = key
    except ValueError:
        continue
#Check the last key (new_key) in list and print its variance
if len(new_val)>0:
    mean_val = sum(new_val)/len(new_val)
    val_ls = [(val - mean_val)**2 for val in new_val]
    std_val = math.sqrt(sum(val_ls)/len(val_ls))
    print("%s\t%f" % ( new_key, val_ls))
    new_val.clear()
```

Code to output the MapReduce mapper code locally:

```
cat 200704hourly.txt |./mapper_temp.py|sort
```

To output the mapper and reducer code locally:

```
cat 200704hourly.txt |./mapper_temp.py|sort |./reducer_min_temp.py
```

Hadoop Code for implementing MapReduce:

```
hadoop jar /opt/hadoop/current/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file mapper_temp.py -mapper mapper_temp.py -file
reducer_min_temp.py -reducer reducer_min_temp.py -input ./CW1/200704hourly.txt -output output_min_temp
```

Q1 layman description of python code

mapper_temp.py

create a mapper file in Python that will output all daily temperatures by hour for different weather stations, April 2007
make it possible to execute a script file with Python
import the sys module in Python
iterate through the input file until end.
a copy of the string with the trailing characters removed is displayed.
a list of the words using comma as a delimiter in the string is returned.
separate the strings into keys and values.
the key will include the weather station ID (Wban Number) and the date (YearMonthDay)
the value is the temperature for the key (Dry Temp Bulb)
create a 'for loop' which will skip over pass the temperature values with the character '-'
try and except statement to convert values to integer format

reducer_min_temp.py

create a reducer file in Python that will output minimum daily temperatures for different weather stations, April 2007
make it possible to execute a script file with Python
import the sys module in Python
assign None (for character) and 0 (for number) to new tuple variables (new_key) and (min_val)
iterate through the standard input until end of the file is reached (file = mapper_temp.py)
the input file will be split by the tab delimiter, \t
create an 'if else' statement to either print the key, value pairs as new tuples or skip them
keys or values with None or 0 will not form part of the output key value pairs
minimum values will be calculated for those values forming part of the key that do not contain None or 0
if the minimum values are calculated the key value pairs will be printed

reducer_max_temp.py

create a reducer file in Python that will output maximum daily temperatures for different weather stations, April 2007
make it possible to execute a script file with Python
import the sys module in Python
assign None (for character) and 0 (for number) to new tuple variables (new_key) and (max_val)
iterate (repetitions) through the standard input until end of the file is reached (file = mapper_temp.py)
the input file will be split by the tab delimiter, \t
create an 'if else' statement to either print the key, value pairs as new tuples or skip them
keys or values with None or 0 will not form part of the output key value pairs
maximum values will be calculated for those values forming part of the key that do not contain None or 0
if the minimum values are calculated the key value pairs will be printed

reducer_mean_temp.py

create a reducer file in Python that will output mean daily temperatures for different weather stations, April 2007
make it possible to execute a script file with Python
import the sys module in Python
assign None (for character) and 0 (for number) to new list variables new_key, temp_num and temp_total
iterate (repetitions) through the standard input until end of the file is reached (file = mapper_temp.py)
the input file will be split by the tab delimiter, \t
if the length of the input data is not equal to 2 values, then create key, value pairs called data

if the data is not part of the key, temperature values will be printed as 0
else, the temp_num will have 1 record printed as a float
create an 'if else' statement to either print the key, value pairs as new tuples or skip them
mean values will be calculated for those values forming part of the key that do not contain None or 0
if the mean values are calculated the key, value pairs will be printed

reducer_median_temp.py
create a reducer file in Python that will output median daily temperatures for different weather stations, April 2007
make it possible to execute a script file with Python
import the sys module in Python
assign None (for character) new list variable new_key, and an empty list for all_temp
iterate (repetitions) through the standard input until end of the file is reached (file = mapper_temp.py)
the input file will be split by the tab delimiter, \t and will expect at least 2 values.
calculate the median of the values by indexing the values and output into the new list
if the data meets the criteria, update the list with float values of the median temperatures
if there are value errors, continue to the next step
check the last key and output the median temperatures to the list

reducer_variance_temp.py
create a reducer file in Python that will output standard deviation of daily temperatures for different weather stations, April 2007
make it possible to execute a script file with Python
import the sys module in Python
import the math module in Python
assign None (for character) new list variable new_key, and an empty list for all_temp
iterate (repetitions) through the standard input until end of the file is reached (file = mapper_temp.py)
assign None (for character) and 0 (for number) to new list variables new_key, temp_num and temp_total
iterate (repetitions) through the standard input until end of the file is reached (file = mapper_temp.py)
the input file will be split by the tab delimiter, \t
if the length of the input file is not equal to 2 values, then create key, value pairs
if the data is not part of the key, temperature values will be printed as 0
calculate the variance
append the values that meet the criteria to the new_val list.
the values will be in float format. The new_key will equal the input key
print the key with the variance
if there are value errors, the code will continue on to the next step
check the last key and output the variance temperatures to the list

clustering_cosine_k20_script.py
#!/bin/bash.

mahout seqdirectory -i docs -o docs-sequfiles -c UTF-8 -chunk 5
mahout seq2sparse -nv -i docs-seqfiles -o docs-vectors
mahout canopy -i docs-vectors/tfidf-vectors -ow -o docs-vectors/docs-canopy-centroids -dm org.apache.mahout.common.distance.CosineDistanceMeasure -t1 0.5 -t2 0.3
hadoop fs -ls docs-vectors/docs-canopy-centroids
mahout kmeans -i docs-vectors/tfidf-vectors -c docs-canopy-centroids -o hdfs://lena/user/rtyle001/docs-kmeans-clusters -dm org.apache.mahout.common.distance.CosineDistanceMeasure -cl -cd 0.1 -ow -x 20 -k 20

clustering_cosine_k20_script.py
#!/bin/bash.

```
mahout clusterdump -dt sequencefile -d docs-vectors/dictionary.file-* -i docs-kmeans-clusters/clusters-2-final -o
clusters_cosine_k20.txt -b 100 -p docs-kmeans-clusters/clusteredPoints -n 20 --evaluate
tail ./clusters_cosine_k20.txt
```

A summary of the commands in the script files includes:

Script 1:

- Create directories to store the input, sequence file and the clustered output. This is done with the seqdirectory command. The files will need to be copied onto the Hadoop file system before executing this script.
- Vectorised documents are created with seq2sparse command
- Running the canopy clustering algorithm, specify the distance metric – in this case cosine
- Run the k-means clustering algorithm, specify the distance metric (cosine in this example) and the k-value (20 in this example)

Script 2:

- Run the clusterdump command which converts the output into human-readable code; specify the output file.
- View the tail end of the output file to view the inter and intra-cluster values.