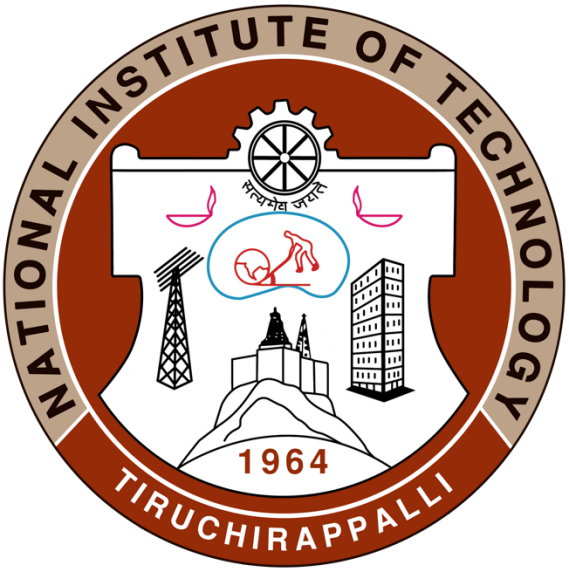
Empirical Earthquake Prediction and Analysis

Programming Assignment – Big Data Analytics (CSOE17)

(7^{th} Semester B-Tech EEE | 2016 – 2020 Batch)

SUBMITTED
BY

Name : Deebthik Ravi
Roll Number : 107116025



DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY TIRUCHIRAPPALLI
TIRUCHIRAPPALLI IN TAMIL NADU, INDIA

TABLE OF CONTENTS

ABSTRACT

Characteristic perils like earthquakes are for the most part the consequence of spreading seismic waves underneath the surface of the earth. Tremors are dangerous absolutely in light of the fact that they're erratic, striking without warning, triggering fires and tsunamis and leading to deaths of countless individuals. If people could be cautioned in weeks or months ahead of time about seismic disturbances, clearing and different arrangements could be made to spare incalculable lives. An early identification and future earthquake prediction can be achieved using machine learning models. Seismic stations continuously gather data without the necessity of the occurrence of an event. The gathered data can be used to distinguish earthquake and non-earthquake prone regions. Machine learning methods can be used for analysing continuous time series data in order to detect earthquakes effectively. The pre-existing linear models applied to earthquake problems have failed to achieve significant amount of efficiency and generate overheads with respect to pre-processing. This assignment exploits parallel processing in Hadoop by using the various frameworks like Pig-Hive optimization, Map Reduce and Impala, in order to mine and analyse earthquake data to propose a model for predicting future earthquakes.

INTRODUCTION

Movement of seismic plates under the surface of the earth which supports life in many form, causes earthquakes which is a natural hazard. Seismometers, which are used to record motion of these plates, are installed at various locations on the planet. These instruments detect vertical motion of the plates to record it on the scale. The earth surface formally called the crust is divided into seven large tectonic plates. These larger plates are further divided into several small sub-plates which are being observed and are noticed to move apart continuously. There are variances of seismic types. These can be stated as divergence, convergence which lead to transformation of plate boundaries. When the plates distance themselves from each other, new boundaries are introduced. In the phenomenon of convergence, plates of different densities tend to approach nearer giving rise to new geographical structures. When these plates slide apart from each other, this type of motion is called transformation. Divergence, convergence and transformations are all together known as faults. A fault in any geological region causes stress. When the stress quantity is large, it is released by earth in the form of earthquakes and sometimes volcanic eruption (stress along with heat). Apart from faults, some other reasons leading to earthquakes include volcanic eruptions, nuclear activities, mine blasts. The point of origin of the earthquake is known as the focus point. Earthquakes are recorded by a modern form of geophones called seismometers. These geophones are very sensitive to even small energy patterns that they can record. They work in efficient way when they are installed in groups and work in a cluster. The cluster of geophones can be deployed to increase the accuracy in measurement of seismic values. Geophones are mainly used for two purposes. Firstly, they increase the accuracy by reducing noise results; and secondly, they record vertical displacements and ignore any kind of horizontal seismic vibrations. Horizontally moving seismic waves are also called ground rolls. They are considered as noise which is caused as a side effect of seismic energy patterns. Vertically propagating waves almost simultaneously strike the seismometers installed in a group and are recorded. All the vertical waves that hit the seismometers at the same time are recorded by the cluster and all others which hit with some delay are ignored. The sum of the propagating waves vertically can be calculated and in the end it can generate time series data for recording. Four stages that are included in the prediction of earthquake in this assignment are:-

i) Pig hive optimisation technique is used in the process as it is faster than Hadoop tools like Zookeeper. Thus the processing of the data becomes faster. Apart from Hive, Impala is also used for queries.

ii) Pre-processing phase would include elaborating the different metrics over which the study would be conducted.

iii) Feature extraction is performed on Hadoop and plots are generated for analysis.

iv) Prediction using ensemble algorithms as well as comparison of different clustering techniques.

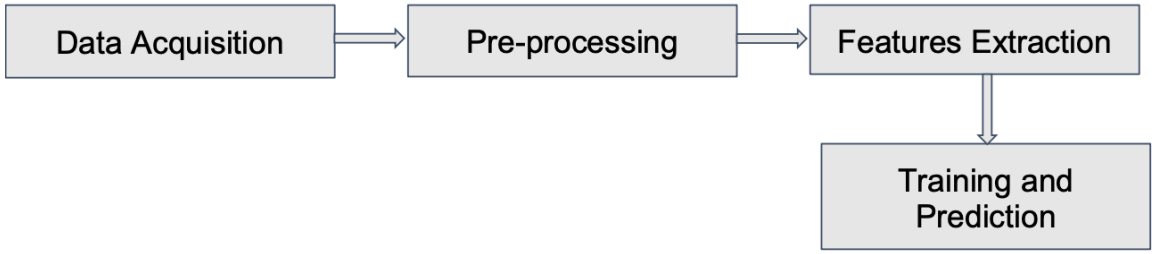


Figure 1: The stages involved in earthquake analysis

ARCHITECTURE

This assignment aims at using parallel processing in order to reduce the overheads that are generated when dealing with the processing and computation of earthquake data. Impala, a massively powerful parallel processing engine is used to perform the computations. This assignment also analyses the earthquake dataset and performs analysis of the various clustering techniques like hierarchical and k-means clustering. The different problems solved by this assignment includes the following:-

1. Predicting the most likely location of the future earthquake using past seismic data using the United States Geological Survey Dataset.
2. Classification of earthquakes based on types, magnitudes, location of occurrences,
3. Data analysis of past earthquakes and visualizations to better understand the factors behind occurrences of them.
4. Finding which regions are affected by earthquakes the most and successfully applying prediction and optimization algorithms on them.

The problems with Big Data are not only the size. There are three V's associated with Big Data that makes it difficult to process on a single machine.

1. Volume: The size of the data being generated.
2. Velocity : The rate at which the data is being generated.
3. Variety : the different sources and formats in which the data is coming.

The Hadoop map reduce algorithm is used to perform the processing. A number of other big data processing techniques have been used. Impala, a query processing variant in the big data environment is used. This assignment also includes a performance comparison of Hive and Impala.

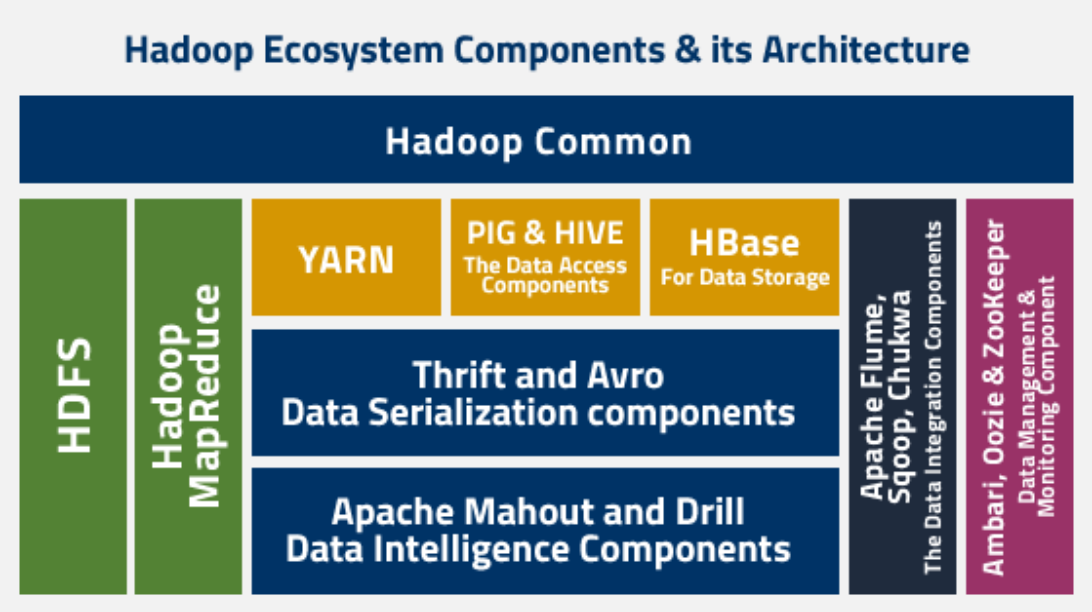


Figure 2: The Hadoop Architecture

The Pig Hive optimization technique is used. Pig is used here rather than other Hadoop tools like Hive or Zookeeper as it is fast and it processes the data faster. Just like the literal meaning of the pig, an animal who eats garbage, similarly here also the pig automatically processes the data(arranges) irrespective of the arranged or not. After the data is processed and cleaned, predictions on it is performed using K-Means. The K-Means algorithm is inefficient for large scale processing thus the this assignment also studied other areas such as evolutionary computing with emphasis on Swarm Particle Optimization to find a highly efficient algorithm that can be used. This would be based on the attributes defined in table below.

DATA COLLECTION

This assignment uses the earthquake dataset of the United States Geological Survey.

Data Attributes for earthquake prediction	
Date	The day when it occurred
Time	The particular time it started
Latitude and Longitude	Location tracing
Type	Natural Earthquake / Nuclear Explosion
Depth	Epicentre Location
Magnitude	Intensity of Earthquake Magnitude Type: MB, MW,ML etc.
Status	It is the place of origin or it received from somewhere

Table 1: The Dataset Features

IMPLEMENTATION

This assignment compares the different Hadoop data processing technologies. The USGS dataset is used for data acquisition phase. The data is processed using Hadoop distributed computing algorithm. The techniques of Hive and Impala are used for the data pre-processing. The cleansed dataset is then fed to the different algorithms in order to observe insights and patterns. The first phase uses the K-means clustering algorithm to form the clusters based on the different locations of the earthquake. The hierarchical clustering and k-means clustering algorithm divide the earthquakes into clusters of Northern Hemisphere, Southern Hemisphere and the ones near the equator.

THE PARTITION CLUSTER ALGORITHM

Step-0: Start
Step-1: Load the dataset into the environment
Step-2: Using Mapper, breakdown the tasks into individual clusters.
Step 3: Put the data in key and value pairs.
Step 4: Map the consequent values of a cluster through synchronised search into key pairs.
Step 5: Randomise the value to reduces through shuffle and sort
Step 6: Transfer the key pairs to reducer
Step 7 : Partition the data and store in HDFS
Step 8 : Compute the particle's closeness in free space and compute the clusters
Step 9: End

In stages of development of the algorithm in cloud-era environment, it was seen to take a lot of time in processing. Since a number of tasks were to be divided into mappers and reducers, the overhead and complexity increased enormously. Therefore, the data visualisation was carried out in the Python environment. Figure 4 shows the values obtained from the k-means clustering algorithm.

Techniques	Description
K-Means	Simple clustering algorithm
Hierarchical Clustering	A more complex clustering algorithm

Table 2: The Clustering Technique Used

The figure 3(b) and 4(b) given below shows the results of the different clustering performed in the respective hemispheres. The elbow method is used to check the optimum number of clusters by checking the critical points on the graph. The most optimum number of clusters obtained here are 3.

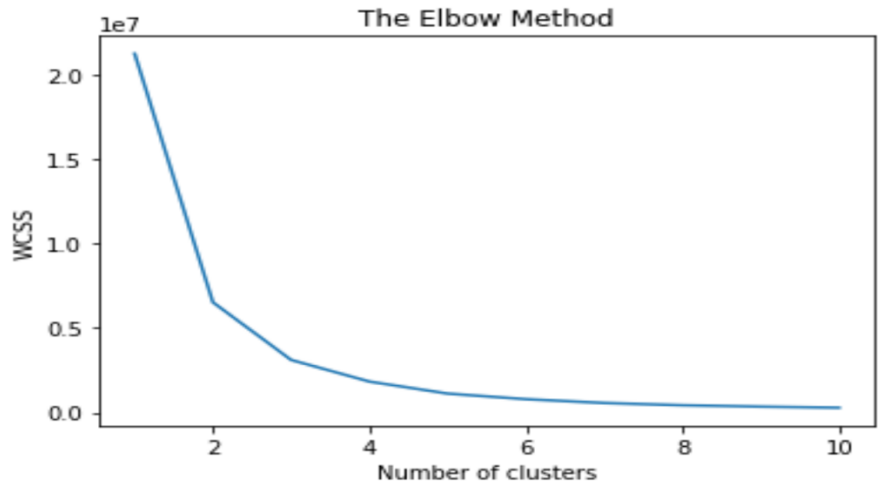


Figure 3(a): Elbow Method

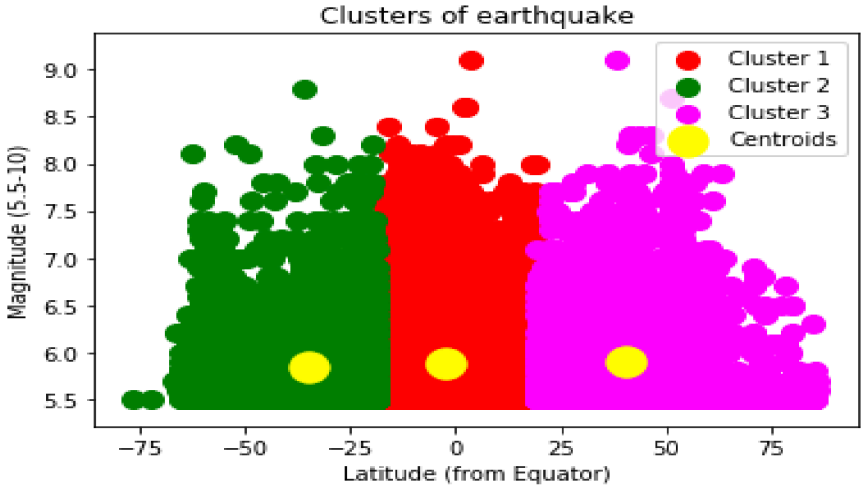


Figure 3(b): K-means Clustering

It was found that the distance between the clusters was more in K-means algorithm. Thus, hierarchical clustering was used as counter measure.

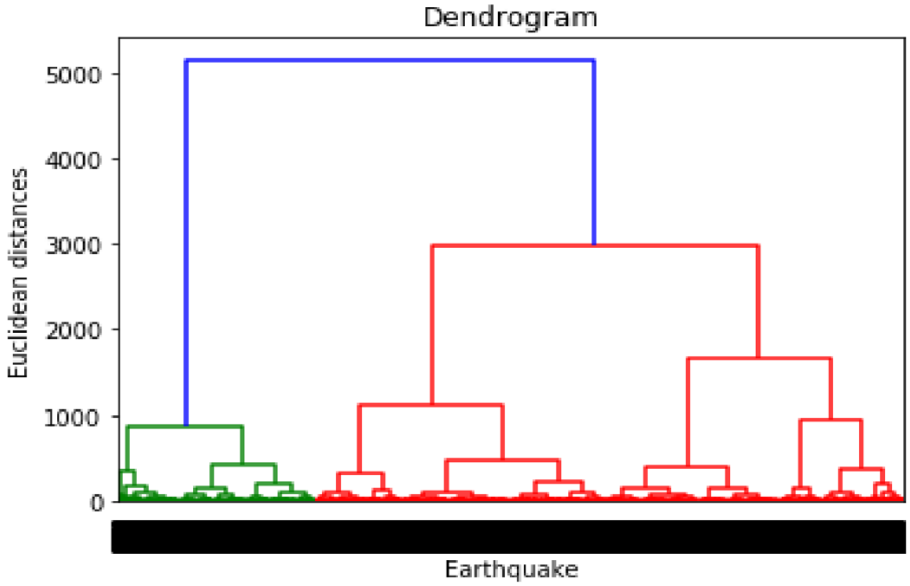


Figure 4(a): Dendrogram

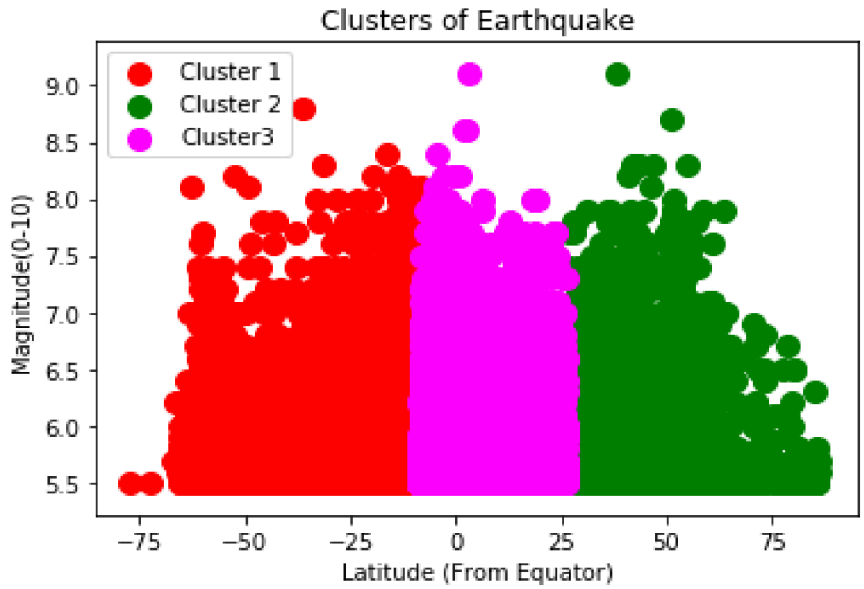


Figure 4(b): Hierarchical Clustering

The dendrogram in figure 4(a) shows the Euclidean distance between the clusters and groups them based on the least distance between the points. The clustering methods are used to obtain the centroid of the latitudes and longitudes and then the most prone locations to earthquakes are found, depending on the distance of the point, the clusters can be mapped as shown in figure 5(b).

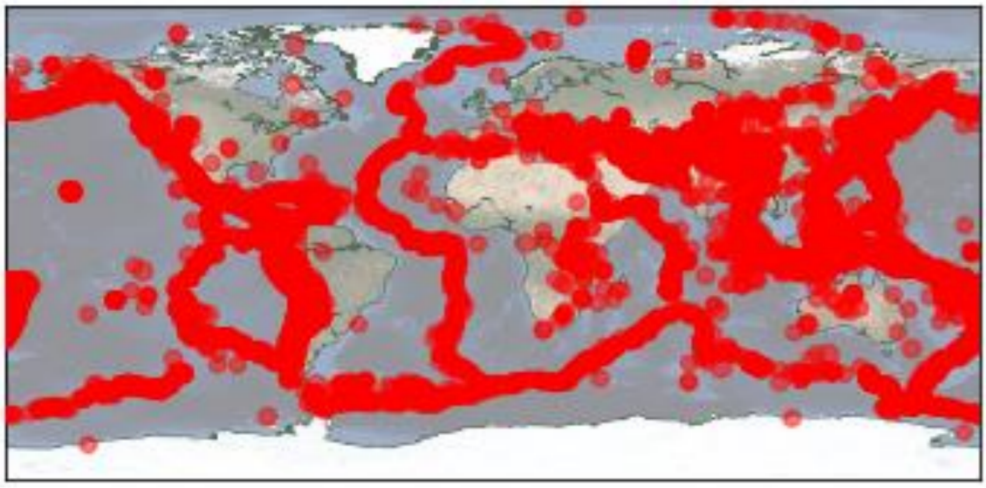


Figure 5(a): The plotting of earthquake dataset

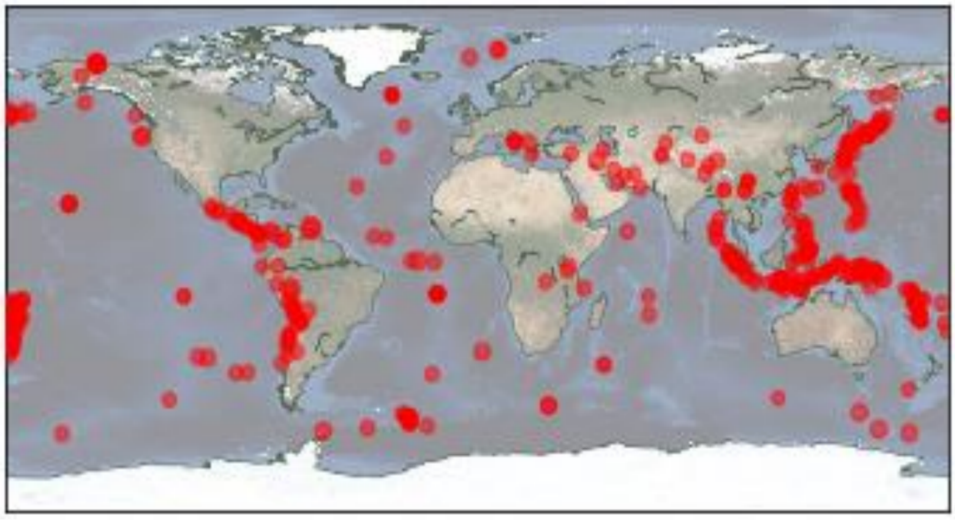


Figure 5(b): The predicted earthquake prone regions

CODE

PLOTTING

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap


DATA_URL = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.csv'

print("Downloading", DATA_URL)
df = pd.read_csv(DATA_URL)

fig, ax = plt.subplots()
earth = Basemap(ax=ax)
earth.drawcoastlines(color='#556655', linewidth=0.5)
ax.scatter(df['longitude'], df['latitude'], df['mag'] ** 2,
           c='red', alpha=0.5, zorder=10)
ax.set_xlabel("This month's earthquakes")
fig.savefig('usgs-monthly.png')
```

K-MEANS CLUSTERING

```
import math
import random
import argparse
from data import *
import turtle
import sys

# constants for the k-means clustering algorithm
# if you change these in your experimentation, you will need to look at
# all parts of the code that refer to them, as there is some dependence
# on them (such as number of colors used in plotting clusters)
#

NO_OF_CLUSTERS = 6
NO_OF_ITERATIONS = 7

class Data_centen:
    """
        Uses list of specific data & provided functions for mean, median
        Args:
            eq_dict: dictionary of lists, each contained list represents an EQ even
t
            data_list: list of specific data to calculate central tendency
        Outputs:
            Statistics in a list
        """
    def __init__(self, data_list):
        self.data_list = data_list

    def statistics(self):
        mean = data_mean(self.data_list)
        median = data_median(self.data_list)

        return (mean, median)
```

```
34.
35. class Data_disp:
36.     """
37.       Uses list of specific data & provided functions for variance & standard devat
    ion
38.         Args:
39.             eq_dict: dictionary of lists, each contained list represents an EQ even
    t
40.             data_list: list of specific data to calculate dispersion statistics
41.         Outputs:
42.             Statistics in a list
43.         """
44.     def __init__(self, data_list):
45.         self.data_list = data_list
46.
47.     def stdev(self):
48.         variance = data_mean_variance(self.data_list)
49.         stdev = math.sqrt(variance[1])
50.         return variance[1], stdev
51.
52.
53. class Data_iso:
54.     """
55.     Isolates columns of data, e.g. for magnitude it isolates the third column in th
    e file,
56.     extracts them, and puts them into a list.
57.     Args:
58.         eq_dict: dictionary of lists, each contained list represents an EQ event
59.         column_number: location of the column. e.g. magnitude == column (2)
60.     Outputs:
61.         A list of the set of specific data, sorted.
62.     """
63.     def __init__(self, column_number, eq_dict):
64.         self.data_type = column_number
65.         self.eq_dict = eq_dict
66.
67.     def isolator(self):
68.         iso_data = []
69.         for key in self.eq_dict:
70.             item = self.eq_dict[key]
71.             iso_data.append(item[self.data_type])
72.         iso_data.sort()
73.         return iso_data
74.
75.     def xy_isolator(self):
76.         iso_data = []
77.         for key in self.eq_dict:
78.             item = self.eq_dict[key]
79.             xy_item = xy_calculate(item[0], item[1])
80.             iso_data.append([xy_item, item[self.data_type]])
81.         iso_data.sort()
82.         return iso_data
83.
84.
85. def euclid_distance(point1, point2):
86.     """
87.     computes the euclidean distance between two points
88.     Args:
89.         point1: list of floats, index 0 is longitude, index 1 is latitude
90.         point2: list of floats, index 0 is longitude, index 1 is latitude
91.     Returns:
92.         float, sqrt((x1-x2)**2 + (y1-y2)**2)
93.     """
94.
95.     total = 0
96.     for index in range(2):
```

```
97.         diff = point1[index] - point2[index]
98.         total += diff * diff
99.
100.            return math.sqrt(total)
101.
102.        def create_centroids(k, datadict):
103.            """
104.            randomly selects 'k' points from 'datadict' as the starting
105.                centroids for the k-means clustering algorithm
106.            Args:
107.                k: int, number of clusters desired
108.                datadict: list of lists, each contained list represents an EQ event
109.            Returns:
110.                list of lists, each contained list is an event to act as the centroi
    d
111.            """
112.            centroids = []
113.            count = 0
114.            centroid_keys = []
115.
116.            while count < k:
117.                rkey = random.randint(1, len(datadict))
118.                if rkey not in centroid_keys:
119.                    centroids.append(datadict[rkey])
120.                    centroid_keys.append(rkey)
121.                    count += 1
122.
123.            return centroids
124.
125.        def create_clusters(k, centroids, datadict, iterations):
126.            """
127.            k-means clustering algorithm - implementation taken from page 249 of
128.                ranum and miller text, with some modifications
129.            Args:
130.                k: integer, number of clusters
131.                centroids: list of events, each event is the centroid of its cluster

132.                datadict: dictionary of all EQ events
133.                iterations: int, number of clustering iterations to perform
134.            Returns:
135.                list of lists: each contained list is the set of indices into 'datad
    ict'
136.                   for events that belong to that cluster
137.            """
138.            for iteration in range(iterations):
139.                #print("****Iteration", iteration, "****")
140.                clusters = []
141.                for i in range(k):
142.                    clusters.append([])
143.
144.                for key in datadict:
145.                    distances = []
146.                    for cl_index in range(k):
147.                        dist = euclid_distance(datadict[key], centroids[cl_index])
148.                        distances.append(dist)
149.                    min_dist = min(distances)
150.                    index = distances.index(min_dist)
151.                    clusters[index].append(key)
152.
153.                dimensions = 2
154.                for cl_index in range(k):
155.                    sums = [0]*dimensions
156.                    for key in clusters[cl_index]:
157.                        data_points = datadict[key]
158.                        for ind in range(2):
159.                            sums[ind] = sums[ind] + data_points[ind]
```

```
                    for ind in range(len(sums)):
                        cl_len = len(clusters[cl_index])
                        if cl_len != 0:
                            sums[ind] /= cl_len
                    centroids[cl_index] = sums

                #for c in clusters:
                    #print("CLUSTER")
                    #for key in c:
                        #print(datadict[key], end=" ")
                    #print()

            return clusters

        def read_file(filename):
            """
            read the EQ events from the csv file, 'filename'; any lines starting with
                # are skipped; the longitude, latitude, magnitude, and depth (in miles)
                is extracted from each event record, and stored as a list against its
                record number in a dictionary
            Args:
                filename: string, name of a CSV file containing the EQ data
            Returns:
                dictionary, indexed by integers, each value is a list of floats
                    representing an EQ event
            """
            dict = {}
            key = 0

            fd = open(filename, "r")
            for line in fd:
                if line[0] == '#':
                    continue        # causes the loop to grab another line
                key += 1
                values = line.rstrip('\n').split(',')
                lat = float(values[7])
                lon = float(values[8])
                mag = float(values[1])
                dep = float(values[10])
                dict[key] = [lon, lat, mag, dep]
            fd.close()
            return dict

        # global data for map - if we had ;earmed about classes yet, this would have
        # been hidden in a class instance, and the plot_*() functions would be methods
        # on that class instance.  for now, these are global variables, and the
        # plot functions access them

        eq_turtle = None
        eq_win = None
        # these are the longitudes and latitudes for the Pacific NorthWest map that
        # I have provided to you; do not change them!
        left_lon = -128.608689
        right_lon = -114.084764
        top_lat = 51.248522
        bot_lat = 38.584004
        lon_diff = 0
        lat_diff = 0
        size_x = 0
        size_y = 0
```

```
220.        left_x = 0
221.        bot_y = 0
222.
223.        def prepare_turtle():
224.            """
225.            Prepares the turtle and the window to plot magnitudes, depths, or cluste
    rs
226.            Args:
227.                None
228.            Outputs:
229.                creates turtle, sets window size, defines remainder of global
230.                data needed for plot_routines
231.            """
232.            global eq_turtle, eq_win
233.            global left_lon, right_lon, top_lat, bot_lat
234.            global lon_diff, lat_diff
235.            global size_x, size_y, left_x, bot_y
236.
237.            eq_turtle = turtle.Turtle()
238.            eq_turtle.speed(10)
239.            eq_win = turtle.Screen()
240.            eq_win.screensize(655,808)  # number of pixels in the map I have provide
    d
241.            lon_diff = right_lon - left_lon
242.            lat_diff = top_lat - bot_lat
243.            size_x = eq_win.screensize()[0]
244.            left_x = -size_x/2
245.            size_y = eq_win.screensize()[1]
246.            bot_y = -size_y/2
247.            eq_win.bgpic("PacificNW.gif")   # the map I have provided
248.            eq_turtle.hideturtle()
249.            eq_turtle.up()
250.
251.        def xy_calculate(lon, lat):
252.            """
253.            compute (x, y) given lon[gitude] and lat[itude]
254.            Args:
255.                lon: float, longitude value for point on map
256.                lat: float, latitude value for point on map
257.            Returns:
258.                tuple, corresponding pixel x and y values for use in turtle methods
259.            """
260.            global left_lon, right_lon, top_lat, bot_lat
261.            global lon_diff, lat_diff
262.            global size_x, size_y, left_x, bot_y
263.
264.            x = left_x + (lon - left_lon) / lon_diff * size_x
265.            y = bot_y + (lat - bot_lat) / lat_diff * size_y
266.            return (x, y)
267.
268.        def plot_clusters(eq_clusters, eq_dict):
269.            """
270.            plot the clusters - use turtle.dot() at the appropriate location on the
271.                map for each event; use a different color for the events in each
272.                cluster - e.g. for cluster 0, use 'red', for 1, use 'violet' ...
273.            Args:
274.                eq_clusters: list of lists, each contained list has the indices for
275.                             events in that cluster in eq_dict
276.                eq_dict: dictionary of lists, each contained list represents an EQ e
    vent
277.            Outputs:
278.                plots all events in a particular cluster as dots on the map
279.            """
280.
281.            COLORS = {
282.                    0:'green', 1:'red', 2:'blue', 3:'cyan', 4:'violet',
```

```
                    5:'purple', 6:'brown',7:'yellow', 8:'navy', 9:'light green',}

            global eq_turtle
            count = 0
            final_dict = {}
            for cluster in eq_clusters:
                cluster_list = []
                for number in cluster:
                    item = eq_dict[number]
                    xy_coord = xy_calculate(item[0], item[1])
                    cluster_list.append(xy_coord)
                final_dict[count] = cluster_list
                count += 1

            for key in final_dict:
                item = final_dict[key]
                for xycoord in item:
                    eq_turtle.goto(xycoord)
                    eq_turtle.dot(7.5, COLORS[key])

        def bin_value(value, bounds):
            """
            'bounds' defines a set of bins; this function returns the index of the
                first bin that contains 'value'
            Args:
                value: float, value to place in bin
                bounds: list of floats, bounds[i] is the top value of the bin
                        code assumes that bounds is an increasing set of values
            Returns:
                integer, index of smallest value of bounds[] that is >= value
                    if value > bounds[-1], returns len(bounds)
            """
            for i in range(len(bounds)):
                if value <= bounds[i]:
                    return i
            return len(bounds)

        def plot_magnitudes(eq_dict):
            """
            plot the magnitudes - use turtle.dot() at the appropriate location on th
    e
                map for each event; use a different color and size for magnitude
                equivalence classes - e.g. if magnitude of event is <=1, use small d
    ot
                that is 'violet', if between 1 and 2, use slightly larger dot that i
    s
                'blue', ..., if between 9-10, use very large dot that is 'red'
            Args:
                eq_dict: dictionary of lists, each contained list represents an EQ e
    vent
            Outputs:
                plots magnitude of all events as dots on the map
            """

            global eq_turtle

            extraction_class = Data_iso(2, eq_dict)
            xypoint_list = extraction_class.xy_isolator()

            for point in xypoint_list:
                eq_turtle.goto(point[0])
                if point[1] <= 1.0:
                    eq_turtle.dot(7.5, 'violet')
                if 1.0 < point[1] <= 2.0:
                    eq_turtle.dot(15, 'blue')
                if point[1] > 9.0:
```

```
345.                     eq_turtle.dot(22.5, 'red')
346.
347.         def plot_depths(eq_dict):
348.             """
349.             plot the depths - use turtle.dot() at the appropriate location on the
350.                 map for each event; use a different color and size for depth
351.                 equivalence classes - e.g. if depth of event is <=1 mile, use a larg
    e
352.                 dot that is 'red', if between 1 and 5, use slightly smaller dot that
     is
353.                 'orange', ..., if between 50-100, use a small dot that is 'violet'
354.             Args:
355.                 eq_dict: dictionary of lists, each contained list represents an EQ e
    vent
356.             Outputs:
357.                 plots depth of all events as dots on the map
358.             """
359.             global eq_turtle
360.             extraction_class = Data_iso(3, eq_dict)
361.             xypoint_list = extraction_class.xy_isolator()
362.
363.             for point in xypoint_list:
364.                 eq_turtle.goto(point[0])
365.                 if 50 <= point[1] <= 100:
366.                     eq_turtle.dot(5, 'cyan')
367.                 if 1 <= point[1] <= 5:
368.                     eq_turtle.dot(10, 'blue')
369.                 if point[1] <= 1:
370.                     eq_turtle.dot(15, 'green')
371.
372.
373.         def analyze_depths(eq_dict):
374.             """
375.             Perform statistical analysis on the depth information in the dictionary
376.             Args:
377.                 eq_dict: dictionary of lists, each contained list represents an EQ e
    vent
378.             Outputs:
379.                 mean, median, and standard deviation of depth data
380.                 frequency table for the depth data
381.             """
382.
383.             depth_list = Data_iso(3, eq_dict)
384.             depth_centen = Data_centen(depth_list.isolator())
385.             depth_disp = Data_disp(depth_list.isolator())
386.
387.             centen = depth_centen.statistics() #0 - mean, 1 - median
388.             disp = depth_disp.stdev()          #0 - variance, 1 - standard deviation

389.
390.             frequency = frequency_list(depth_list.isolator())
391.
392.             data_format(centen[0], centen[1], disp[1], 'depth', frequency, 'miles')

393.
394.         def data_format(mean, median, stdev, type_name, frequency, units):
395.             """
396.             Takes the central tendency statistics and prints in in a format
397.             that is viewable for the user. Used for 'analyze magnitude/depths'
398.             command.
399.             Args:
400.                 mean, median, stdev: Results of inputting the data list into the giv
    en
401.                                         data.py functions
402.                 frequency: A frequency table from the 'frequency_list' function
403.                 type_name: The name of the type of data, ie 'magnitude'
```

```
404.                units: Type of number to add to print, ie 'Miles', if none then unit
    s=''
405.            Outputs:
406.                No output, prints all the data for the user.
407.            """
408.            print('Analysis of {} data'.format (type_name))
409.            print('     Mean {} = {:.02} {}'.format(type_name, mean, units))
410.            print('     Median {} = {:.02} {}'.format(type_name, median, units))
411.            print('     Standard Deviation = {:.02f} {}'.format(stdev, units))
412.            print('ITEM    FREQUENCY')
413.            for item in frequency:
414.                print('  {}      {}'.format(item[0], item[1]))
415.
416.        def frequency_list(data_list):
417.            """
418.            Takes a list of data of one catagory (ie magnitude) and counts frequency

419.            of each occurance. Each unique point is stored in a list with its count.

420.            Used for 'analyze depths/magnitudes' command.
421.            Args:
422.                data_list: Data list of one category
423.            Outputs:
424.                List of lists which have two elements each, the unique key and frequ
    ency. ie [[zealot, 2], [tracer, 4]]
425.            """
426.
427.            freq_list = []
428.            checked_list = []
429.
430.            for item in data_list:
431.                if item not in checked_list:
432.                    freq_list.append([item, data_list.count(item)])
433.                    checked_list.append(item)
434.
435.            return freq_list
436.
437.        def analyze_magnitudes(eq_dict):
438.            """
439.            Perform statistical analysis on the magnitude information in the diction
    ary
440.            Args:
441.                eq_dict: dictionary of lists, each contained list represents an EQ e
    vent
442.            Outputs:
443.                mean, median, and standard deviation of magnitude data
444.                frequency table for the magnitude data
445.            """
446.
447.            magnitude_list = Data_iso(2, eq_dict)
448.            magnitude_centen = Data_centen(magnitude_list.isolator())
449.            magnitude_disp = Data_disp(magnitude_list.isolator())
450.
451.            centen = magnitude_centen.statistics() #0 - mean, 1 - median
452.            disp = magnitude_disp.stdev()          #0 - variance, 1 - standard devia
    tion
453.
454.            frequency = frequency_list(magnitude_list.isolator())
455.            units = ''
456.
457.            data_format(centen[0], centen[1], disp[1], 'Magnitude', frequency, units
    )
458.
459.        def analyze_clusters(eq_clusters, eq_dict):
460.            """
461.            Perform statistical analysis on the depth and magnitude information
```

```
462.                for each cluster
463.            Args:
464.                eq_clusters: list of lists, each contained list has the indices into
                                 eq_dict for events in that cluster
465.                eq_dict: dictionary of lists, each contained list represents an EQ e
    vent
466.            Outputs:
467.                put into a dictionary, for each cluster:
468.                    mean, median, and standard deviation of magnitude data
469.                    mean, median, and standard deviation of depth data
470.            """
471.
472.            counter = 0
473.            for cluster in eq_clusters:
474.                cluster_dict = {}
475.
476.                for number in cluster:
477.                    cluster_dict[number] = eq_dict[number]
478.
479.                #Cluster Statistics for Magnitude
480.                magnitude_list = Data_iso(2, cluster_dict)
481.                magnitude_centen = Data_centen(magnitude_list.isolator())
482.                magnitude_disp = Data_disp(magnitude_list.isolator())
483.                m_centen = magnitude_centen.statistics()  # 0 - mean, 1 - median
484.                m_disp = magnitude_disp.stdev()  # 0 - variance, 1 - standard deviat
    ion
485.
486.                #Cluster Statistics for Depth
487.                depth_list = Data_iso(3, cluster_dict)
488.                depth_centen = Data_centen(depth_list.isolator())
489.                depth_disp = Data_disp(depth_list.isolator())
490.                d_centen = depth_centen.statistics()  # 0 - mean, 1 - median
491.                d_disp = depth_disp.stdev()  # 0 - variance, 1 - standard deviation
492.
493.                #Prints the data for the user, repeats for cluster.
494.                print('Analysis of cluster {}'.format(counter))
495.                print('     Analysis of magnitude data')
496.                print('         Mean magnitude = {:.1f}'.format(m_centen[0]))
497.                print('         Median magnitude = {:.1f}'.format(m_centen[1]))
498.                print('         Standard deviation = {:.02f}'.format(m_disp[1]))
499.                print('     Analysis of depth data')
500.                print('         Mean depth = {:.1f} miles'.format(d_centen[0]))
501.                print('         Median depth = {:.1f} miles'.format(d_centen[1]))
502.                print('         Standard deviation = {:.02f} miles'.format(d_disp[1]
    ))
503.
504.                counter += 1
505.
506.        def main():
507.            """
508.            Interaction if run from the command line.
509.            Usage:  python3 eqanalysis.py eq_data_file.csv command
510.            """
511.            parser = argparse.ArgumentParser(description="Earthquake event file stat
    s")
512.            parser.add_argument('eq_file', type=str,
513.                        help='A csv file containing earthquake events, one per line
    .')
514.            parser.add_argument('command', type=str,
515.                        help='One of the following strings: plot analyze')
516.            parser.add_argument('what', type=str,
517.                        help='One of the following strings: clusters depths magnitu
    des')
518.            args = parser.parse_args()
```

```
            eq_file = args.eq_file
            cmd = args.command
            what = args.what
            if cmd != 'plot' and cmd != 'analyze':
                print('Illegal command: {}; must be "plot" or "analyze"'.format(cmd)
    )
                sys.exit(1)
            if what != 'clusters' and what != 'magnitudes' and what != 'depths':
                print('Can only process clusters, magnitudes, or depths')
                sys.exit(1)
            eq_dict = read_file(eq_file)
            prepare_turtle()
            if what == 'clusters':
                eq_centroids = create_centroids(NO_OF_CLUSTERS, eq_dict)
                eq_clusters = create_clusters(NO_OF_CLUSTERS, eq_centroids, eq_dict,
     NO_OF_ITERATIONS)
            if cmd == 'plot':
                if what == 'clusters':
                    plot_clusters(eq_clusters, eq_dict)
                elif what == 'magnitudes':
                    plot_magnitudes(eq_dict)
                elif what == 'depths':
                    plot_depths(eq_dict)
                print("ALL EVENTS HAVE BEEN PLOTTED")
                eq_win.exitonclick()
            else:
                if what == 'clusters':
                    analyze_clusters(eq_clusters, eq_dict)
                elif what == 'magnitudes':
                    analyze_magnitudes(eq_dict)
                elif what == 'depths':
                    analyze_depths(eq_dict)

        if __name__ == "__main__":
            main()
```

RESULTS AND PERFORMANCE EVALUATION

The different methods of Hadoop like Impala and Hive were compared. In phase 1, the data processing was done and insights on the different kinds of data was observed. The map reduce algorithm although versatile and universal, it fails to match efficiency and fast analytical parallel processing of Impala. Figure 6, shows a comparison for different number of tasks in Hadoop and Impala.

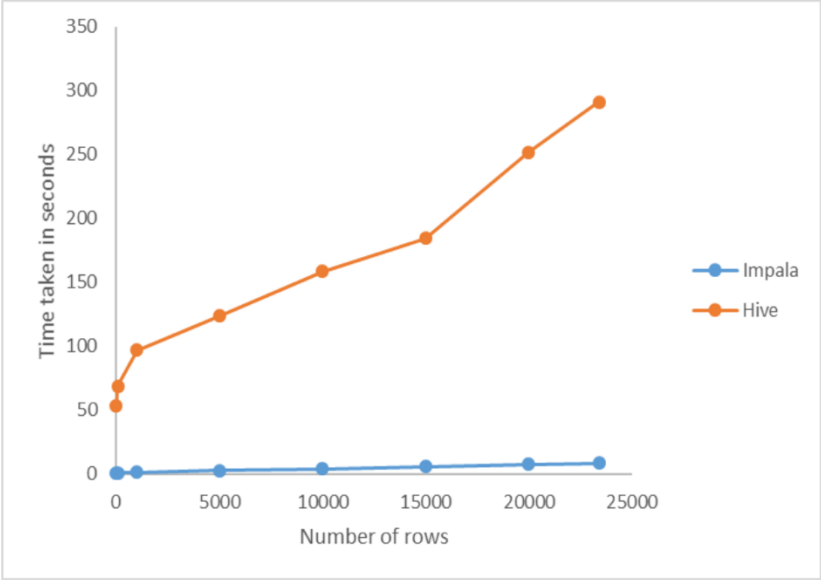


Figure 6: The Job Processing Speed Comparison

From Figure 3 it can be seen that the time taken for processing the same set of tasks in Impala was found to be far less compared to that of Hive. Impala demonstrates the brute force processing power to give lightning fast analytic results. Impala responds quickly through processing whereas Hive translates the given query into MapReduce jobs. Due to this, the overhead increases and thus leads to more processing time. Another advantage with impala is that it avoids start-up overhead as the processes are started at the boot itself, thus always being ready to process a query. Similarly, hive generates overhead during start.

The phase 2 of the this assignment aims at using the cleansed data in order to perform predictions. The train vs test ratio is 95:5. The different clustering techniques were compared and hierarchical clustering was found to give more condensed results. The table 3, compares the values for the different types of clustering techniques used.

No. of rows	K-means (s)	Hierarchical (s)
100	1.72	2.76
1000	2.65	4.11
5000	3.08	6.89
10000	6.11	12.19

Table 3: Comparison of time in different clustering techniques

The hierarchical clustering algorithm takes more processing speed however, further analysis stated that the entropy of hierarchical clustering is less. Entropy refers to the disorder with respect to a given clustering technique. The time overhead is used for making the clusters more precise. Table 4, draws a comparison on the given entropy for K-means and Hierarchical clustering. Similarly the figure 5 draws a line plot to further evaluate the performance between K-means and hierarchical clustering.

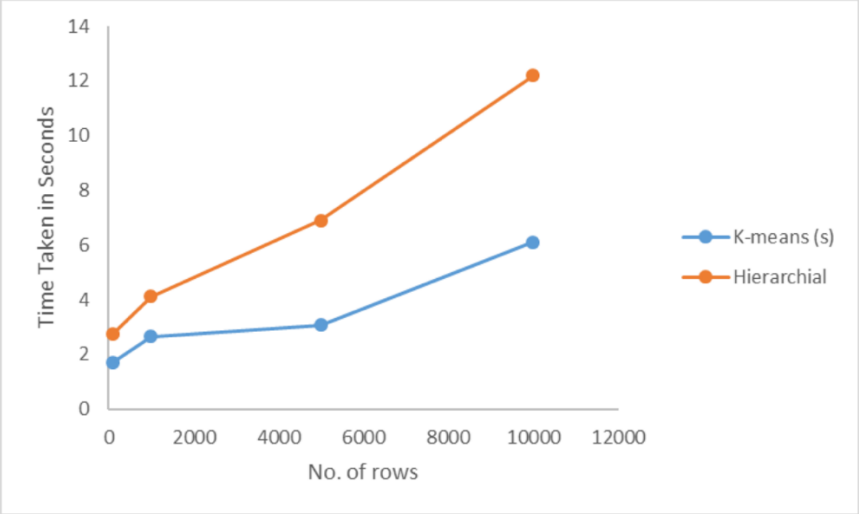


Figure 7: The time comparison between K-means and hierarchical clustering

No. of rows	K-means (s)	Hierarchical (s)
100	0.479	0.217
1000	0.585	0.312
5000	1.633	0.430
10000	2.172	0.768

Table 4: Comparison of K-means and Hierarchical Clustering in terms of entropy

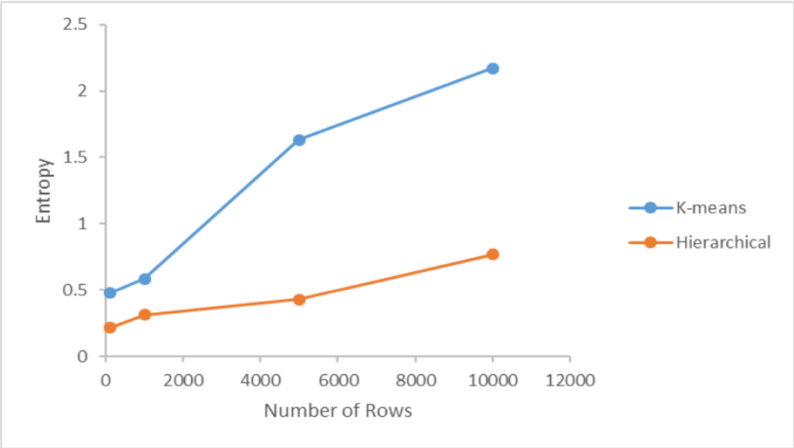


Figure 8: Comparison of Entropy between K-means and Hierarchical Clustering

This assignment also compared other metrics for evaluating the performance of various clustering methods. F-measure and Coefficient of variance were taken into account and compared. The method of exhaustive enumeration was followed in order to obtain the values. The generic algorithm used for analysis was based on the performance of the earthquake dataset. The following algorithm can be used in on other datasets as well for computing the performance. F-measure is used for measuring the accuracy of clustering methods. This value is calculated by weighted average of recall and precision.

$$\text{Recall}(i,j) = \frac{\text{Number of elements of class i in j}}{\text{Number of elements of class i for cluster j}}$$

$$\text{Recall}(i,j) = \frac{\text{Number of elements of class i in j}}{\text{Number of elements of cluster j}}$$

F measure is a result of weighted average of Precision and recall for each class i, and $|i|$ is the given size of the cluster class.

$$F(i,j) = \frac{\sum_i (|i| * F(i))}{\sum_i}$$

The table 4 applies the given formula for evaluating the F measure.

No. of clusters	K-means	Hierarchical
1	0.008	0.01
2	0.0174	0.0312
3	0.0203	0.0389
4	0.0341	0.0412
5	0.0371	0.0562

Table 4: Comparison of F-measure

Figure 9: Comparison of F-measure between K-means and Hierarchical Clustering for different number of clusters

Similarly, the coefficient of variation was found out for the given clustering techniques. The coefficient of variation is obtained from the mean and standard deviation. Table 5 incorporates the coefficient of variance for the given techniques using the same method of exhaustive enumeration.

No. of clusters	K-means	Hierarchical
2	0.372	0.389
3	0.346	0.218
4	0.414	0.357
5	0.407	0.311

Table 5: Comparison of Coefficient of Variance

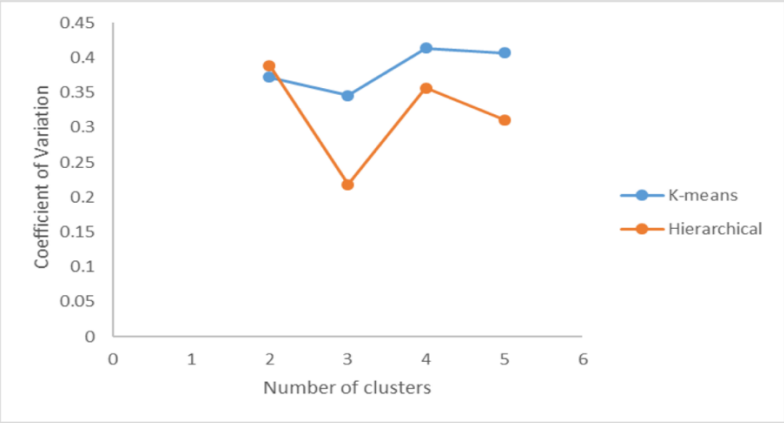


Figure 10: Comparison of Coefficient of variance between K-means and Hierarchical Clustering

CONCLUSION

Thus it can be observed that by using the following algorithmic model for earthquake prediction, proper methods can be implemented for deploying warnings and preparing for earthquakes. The algorithmic model efficiently performs data analysis using Hadoop and can be used for observing insights related to earthquakes. A deep study observed a number of areas that are more prone to earthquakes. Some of these regions include the pacific ring of fire, the Hindukush and the Himalayas, the Japanese coastal spread and the Philippines. It was observed that a number of reasons were responsible for earthquakes, the most dominant were tectonic disturbances followed by nuclear activities. A number of clustering algorithms were used through the course of the research such as K-means and Hierarchical clustering. Hierarchical Clustering was found to be more efficient in terms of entropy but takes more processing time. Similarly, the coefficient of variance of hierarchical clustering was lower than K-means but it was found to have a higher F-measure. Similarly the data analysis was done in the Hadoop environment and techniques like Hive and Impala were compared. However there were a number of drawbacks related to the technique for prediction which can be improved upon.