DSM010 Big Data Analysis - Coursework 1

Q1) Find the descriptive statistics for temperature of each day of a given month for the year 2007.

In this CW the 200707hourly.txt was used as a dataset.

Below is the python 3 implementation for the mapper:

In [ ]:
```
#!/usr/bin/env python
import sys
x = "-"

for line in sys.stdin:
    # grabbing date and dry bulb  temp strings from raw data by splitting line by co
    line = line.split(",")
    date = line[1]
    temp = line[8]
    ## remove empty/wrong date entries
    if date.isdigit() == False:
        continue
    # remove empty dry bulb temp values
    if temp == x:
        continue
    # printing date and temp seperated by comma
    print( '%s,%s' % (date ,temp))
```

In pseudo code:

1. Split each line of the txt.file by commas
2. Get date and dry bulb temp data by index
3. Remove missing values
4. Print the variables date and temp seperated by comma

In pseudo code,the mapper has to grab the date and the dry bulb temperature for each line of the text file and print it out, seperated by a comma.

In the case above this is achieved by splitting each line of the text file by commas and then grabbing date and temp by their indices. At last, empty values have been removed and for every line in the txt file, the date and the dry bulb temp have been printed.

Below the python 3 code for the reducer:

In [ ]:
```
#!/usr/bin/env python
import sys

# function for calculating median
def median(lst):
    quotient, remainder = divmod(len(lst), 2)
    if remainder:
        return sorted(lst)[quotient]
    return sum(sorted(lst)[quotient - 1:quotient + 1]) / 2

# open empty dicitionary
```

```
key_values = {}

for line in sys.stdin:

    #getting key and values
    k, v = line.split(',')
    v = v.replace("\n", "")

    # transforming k and v into integers
    try:
        v = int(v)
        k = int(k)
    except ValueError:
        continue
    # adding new key with empty list if key not already in dict
    if not k in key_values:
        key_values[k]=[]
    # appending values to each key
    key_values[k].append(v)


# looping over dict to calcualte basic statistics and printing them
for k in key_values:
    v = key_values[k]
    median_value = median(v)
    mean = sum(v)/float(len(v))
    max_v=max(v)
    min_v=min(v)
    sumSq = 0
    for a in v:
        sumSq += a*a
    daily_var = sumSq / len(v) - mean * mean
    print('Date: %s\t Max Dry Bulb Temp: %s\t Min Dry Bulb Temp: %s\t Daily mean: %s
```

Above is the python 3 script for the reducer. The reducer gets the input from the mapper after it was sorted.

Pseudo code:

1. For each line get key and value variable
2. Open dicitonary adding the key and an empty list
3. For each key add the according values into the list
4. Now we have each unique key and a list of values that are belonging to the key
5. Calculate basic statistics on the list for each key
6. Print the results

In pseudo code, the reducer has to add all values for one unique key in a list and then do some basic statistic operations on that list for every unique key.

In the code above, a dicitionary is used to insert the unique keys and a list of values for the accoding key.

Then for each key in the dict, statistics like mean max etc. are calculated and printed out.

How to run this code as a layperson

1. Save each code cell seperatly as mapperCW1.py and reducerCW1.py

2. Upload the dataset and the python scripts to the sever by : scp filename(e.g. mapperCW1.py) user@lena.doc.gold.ac.uk:
3. Enter password
4. Upload dataset and if wanted other files to hadoop filesystem: hadoop fs -copyFromLocal /path/in/linux(e.g../BigData/200707hourly.txt) /hdfs/path(e.g. ./CW1)
5. Execute map and reduce task with following command: hadoop jar /opt/hadoop/current/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file mapperCW1.py -mapper mapperCW1.py -file reducerCW1.py -reducer reducerCW1.py -input 200707hourly.txt - output outputCW1
6. Copy the result file to Local: hadoop fs - copyToLocal outputCW1
7. Now there is a folder called outputCW1 which includes a file called "part-00000" which has the result

Date: 20070701 Max Dry Bulb Temp: 115 Min Dry Bulb Temp: 32 Daily mean: 70.71862785135056 Daily median: 72.0 Daily variance: 136.1566969329133

Date: 20070702 Max Dry Bulb Temp: 115 Min Dry Bulb Temp: 32 Daily mean: 70.9395780051151 Daily median: 72.0 Daily variance: 130.5191636442869

Date: 20070703 Max Dry Bulb Temp: 115 Min Dry Bulb Temp: 33 Daily mean: 72.7748518572048 Daily median: 73 Daily variance: 116.1037525976526

Date: 20070704 Max Dry Bulb Temp: 120 Min Dry Bulb Temp: 29 Daily mean: 73.9162407063197 Daily median: 74.0 Daily variance: 113.22126040302737

Date: 20070705 Max Dry Bulb Temp: 120 Min Dry Bulb Temp: 33 Daily mean: 74.60091011854729 Daily median: 74 Daily variance: 117.12680593090772

Date: 20070706 Max Dry Bulb Temp: 133 Min Dry Bulb Temp: 38 Daily mean: 75.04550619082302 Daily median: 75 Daily variance: 125.2744331924232

Date: 20070707 Max Dry Bulb Temp: 116 Min Dry Bulb Temp: 32 Daily mean: 75.8927368787738 Daily median: 77.0 Daily variance: 132.60533740034498

Date: 20070708 Max Dry Bulb Temp: 111 Min Dry Bulb Temp: 36 Daily mean: 76.51774047053998 Daily median: 77 Daily variance: 124.48372471199491

Date: 20070709 Max Dry Bulb Temp: 110 Min Dry Bulb Temp: 36 Daily mean: 75.99281122150789 Daily median: 77.0 Daily variance: 120.98218093397008

Date: 20070710 Max Dry Bulb Temp: 115 Min Dry Bulb Temp: 35 Daily mean: 74.95989140905509 Daily median: 75 Daily variance: 118.15567302437557

Date: 20070711 Max Dry Bulb Temp: 115 Min Dry Bulb Temp: 33 Daily mean: 72.97240273932682 Daily median: 73 Daily variance: 119.79078727653086

Date: 20070712 Max Dry Bulb Temp: 111 Min Dry Bulb Temp: 3 Daily mean: 72.12499271094525 Daily median: 73.0 Daily variance: 127.91880448563006

Date: 20070713 Max Dry Bulb Temp: 111 Min Dry Bulb Temp: 35 Daily mean: 71.83831640058055 Daily median: 72.0 Daily variance: 127.54494694778623

Date: 20070714 Max Dry Bulb Temp: 110 Min Dry Bulb Temp: 40 Daily mean: 73.13970374673251 Daily median: 73.0 Daily variance: 120.40865596799904

Date: 20070715 Max Dry Bulb Temp: 111 Min Dry Bulb Temp: 36 Daily mean: 73.85910952339462 Daily median: 73 Daily variance: 124.76865441892369

Date: 20070716 Max Dry Bulb Temp: 114 Min Dry Bulb Temp: 38 Daily mean: 74.33973407814727 Daily median: 74 Daily variance: 125.14386910969915

Date: 20070717 Max Dry Bulb Temp: 114 Min Dry Bulb Temp: 39 Daily mean: 74.87995037694436 Daily median: 75 Daily variance: 116.84056788887574

Date: 20070718 Max Dry Bulb Temp: 113 Min Dry Bulb Temp: 39 Daily mean: 75.74674272836116 Daily median: 75.0 Daily variance: 107.500900545896

Date: 20070719 Max Dry Bulb Temp: 90 Min Dry Bulb Temp: 74 Daily mean: 80.97402597402598 Daily median: 81 Daily variance: 16.492831843481326

Discussion on MapReduce methodology

MapReduce is a programming model or pattern within the Hadoop framework that is used to access big data stored in the Hadoop File System (HDFS). It is a core component, integral to the functioning of the Hadoop framework.

MapReduce facilitates concurrent processing by splitting petabytes of data into smaller chunks, and processing them in parallel on Hadoop commodity servers. In the end, it aggregates all the data from multiple servers to return a consolidated output back to the application.

At the crux of MapReduce are two functions: Map and Reduce. They are sequenced one after the other.

The Map function takes input from the disk as <key,value> pairs, processes them, and produces another set of intermediate <key,value> pairs as output. The Reduce function also takes inputs as <key,value> pairs, and produces <key,value> pairs as output.

The types of keys and values differ based on the use case. All inputs and outputs are stored in the HDFS. While the map is a mandatory step to filter and sort the initial data, the reduce function is optional.

<k1, v1> -> Map() -> list(<k2, v2>)

<k2, list(v2)> -> Reduce() -> list(<k3, v3>)

Mappers and Reducers are the Hadoop servers that run the Map and Reduce functions respectively. It doesn't matter if these are the same or different servers.

Map:

The input data is first split into smaller blocks. Each block is then assigned to a mapper for processing.

For example, if a file has 100 records to be processed, 100 mappers can run together to process one record each. Or maybe 50 mappers can run together to process two records each. The Hadoop framework decides how many mappers to use, based on the size of the data to be processed and the memory block available on each mapper server.

Reduce:

After all the mappers complete processing, the framework shuffles and sorts the results before passing them on to the reducers. A reducer cannot start while a mapper is still in progress. All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key.

Combine and Partition:

There are two intermediate steps between Map and Reduce.

Combine is an optional process. The combiner is a reducer that runs individually on each mapper server. It reduces the data on each mapper further to a simplified form before passing it downstream.

This makes shuffling and sorting easier as there is less data to work with. Often, the combiner class is set to the reducer class itself, due to the cumulative and associative functions in the reduce function. However, if needed, the combiner can be a separate class as well.

Partition is the process that translates the <key, value> pairs resulting from mappers to another set of <key, value> pairs to feed into the reducer. It decides how the data has to be presented to the reducer and also assigns it to a particular reducer.

The default partitioner determines the hash value for the key, resulting from the mapper, and assigns a partition based on this hash value. There are as many partitions as there are reducers. So, once the partitioning is complete, the data from each partition is sent to a specific reducer.

This methodology was used on the dataset above and resulted in precise outcomes like statistics about the daily weather for one month.

Discussion on K-means algorithms methodology

Clustering:

Clustering is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a

similarity measure such as euclidean-based distance or correlation-based distance. The decision of which similarity measure to use is application-specific.

Clustering analysis can be done on the basis of features where we try to find subgroups of samples based on features or on the basis of samples where we try to find subgroups of features based on samples. We'll cover here clustering based on features. Clustering is used in market segmentation; where we try to find customers that are similar to each other whether in terms of behaviors or attributes, image segmentation/compression; where we try to group similar regions together, document clustering based on topics, etc.

Unlike supervised learning, clustering is considered an unsupervised learning method since we don't have the ground truth to compare the output of the clustering algorithm to the true labels to evaluate its performance. We only want to try to investigate the structure of the data by grouping the data points into distinct subgroups. In this post, we will cover only Kmeans which is considered as one of the most used clustering algorithms due to its simplicity.

Kmeans Algorithm:

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster. The way kmeans algorithm works is as follows:

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
4. Compute the sum of the squared distance between data points and all centroids.
5. Assign each data point to the closest cluster (centroid).
6. Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

Applications:

kmeans algorithm is very popular and used in a variety of applications such as market segmentation, document clustering, image segmentation and image compression, etc. The goal usually when we undergo a cluster analysis is either:

1. Get a meaningful intuition of the structure of the data we're dealing with.
2. Cluster-then-predict where different models will be built for different subgroups if we believe there is a wide variation in the behaviors of different subgroups.

Discussion about Kmeans parameters

Considering another distance measure for comparison

There is no best distance measure. There is only a best distance measure for a given data-set. The choice of the distance measure WILL influence the clustering, but it depends on the data-set and on the objective, which distance measure is most adequate for your particular application.

Below is a table found online with advantages and disatvantages for different kmeans applications:

Distance Measure	Equation	Time complexity	Advantages	Disadvantages	Applications
Euclidean Distance	$d_{euc} = \left[\sum_{i=1}^{n}(x_i - y_i)^2\right]^{\frac{1}{2}}$	O(n)	Very common, easy to compute and works well with datasets with compact or isolated clusters [27,31].	Sensitive to outliers [27,31].	*K*-means algorithm, Fuzzy *c*-means algorithm [38].
Average Distance	$d_{ave} = \left(\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2\right)^{\frac{1}{2}}$	O(n)	Better than Euclidean distance [35] at handling outliers.	Variables contribute independently to the measure of distance. Redundant values could dominate the similarity between data points [37].	*K*-means algorithm
Weighted Euclidean	$d_{we} = \left(\sum_{i=1}^{n}w_i(x_i - y_i)^2\right)^{\frac{1}{2}}$	O(n)	The weight matrix allows to increase the effect of more important data points than less important one [37].	Same as Average Distance.	Fuzzy *c*-means algorithm [38]
Chord	$d_{chord} = \left(2 - 2\frac{\sum_{i=1}^{n}x_i y_i}{\|x\|_2\|y\|_2}\right)^{\frac{1}{2}}$	O(3n)	Can work with un-normalized data [27].	It is not invariant to linear transformation [33].	Ecological resemblance detection [35].
Mahalanobis	$d_{mah} = \sqrt{(x-y)S^{-1}(x-y)^T}$	O(3n)	Mahalanobis is a data-driven measure that can ease the distance distortion caused by a linear combination of attributes [35].	It can be expensive in terms of computation [33]	Hyperellipsoidal clustering algorithm [30].
Cosine Measure	$\mathrm{Cosine}(x,y) = \frac{\sum_{i=1}^{n}x_i y_i}{\|x\|_2\|y\|_2}$	O(3n)	Independent of vector length and invariant to rotation [33].	It is not invariant to linear transformation [33].	Mostly used in document similarity applications [28,33].
Manhattan	$d_{man} = \sum_{i=1}^{n}(x_i - y_i)$	O(n)	Is common and like other Minkowski-driven distances it works well with datasets with compact or isolated clusters [27].	Sensitive to the outliers. [27,31]	*K*-means algorithm
Mean Character Difference	$d_{MCD} = \frac{1}{n}\sum_{i=1}^{n}\|x_i - y_i\|$	O(n)	*Results in accurate outcomes using the K-medoids algorithm.	*Low accuracy for high-dimensional datasets using K-means.	Partitioning and hierarchical clustering algorithms.
Index of Association	$d_{IOA} = \frac{1}{n}\sum_{i=1}^{n}\left\|\frac{x_i}{\sum_{i=1}^{n}x_i} - \frac{y_i}{\sum_{i=1}^{n}y_i}\right\|$	O(3n)	-	*Low accuracy using K-means and K-medoids algorithms.	Partitioning and hierarchical clustering algorithms.
Canberra Metric	$d_{canb} = \sum_{i=1}^{n}\frac{\|x_i - y_i\|}{(x_i + y_i)}$	O(n)	*Results in accurate outcomes for high-dimensional datasets using the K-medoids algorithm.	-	Partitioning and hierarchical clustering algorithms.
Czekanowski Coefficient	$d_{czekan} = 1 - \frac{2\sum_{i=1}^{n}min(x_i, y_i)}{\sum_{i=1}^{n}(x_i + y_i)}$	O(2n)	*Results in accurate outcomes for medium-dimensional datasets using the K-means algorithm.	-	Partitioning and hierarchical clustering algorithms.
Coefficient of Divergence	$d_{canb} = \left(\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - y_i}{x_i + y_i}\right)^2\right)^{\frac{1}{2}}$	O(n)	*Results in accurate outcomes using the K-means algorithm.	-	Partitioning and hierarchical clustering algorithms.
Pearson coefficient	$Pearson(x,y) = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}}$	O(2n)	*Results in accurate outcomes using the hierarchical single-link algorithm for high dimensional datasets.	-	Partitioning and hierarchical clustering algorithms.

*Points marked by asterisk are compiled based on this article's experimental results.

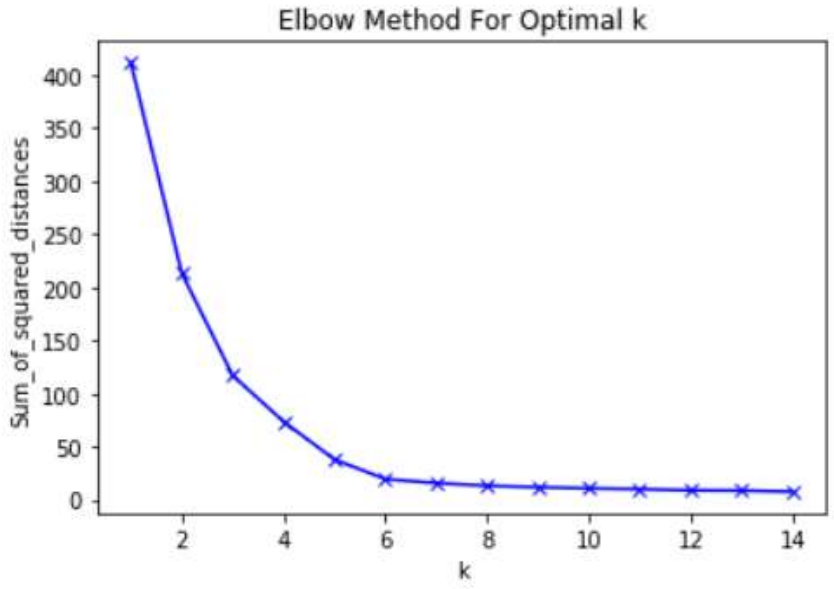
doi:10.1371/journal.pone.0144059.t001

Selecting optimal K for K-means clustering

There are several ways to detect the optimal amount of clusters for k-means clustering but one of the most popular method is the elbow method:

The elbow-point is not a definitive rule but is more of a heuristic method (it works most of the time but not always, so I see it more like is a good rule-of-thumb for choosing a number of clusters to start from). On top of that, the elbow-point cannot always be unambiguously identified.

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In the plot above the elbow is at k=5 indicating the optimal k for this dataset is 5.

References:

https://blog.cambridgespark.com/how-to-determine-the-optimal-number-of-clusters-for-k-means-clustering-14f27070048f

https://stackoverflow.com/questions/59106417/what-would-be-the-best-k-for-this-kmeans-clustering-elbow-point-plot

https://arxiv.org/ftp/arxiv/papers/1405/1405.7471.pdf#:~:text=In%20K%2DMeans%20algorithm%2C

https://www.talend.com/resources/what-is-mapreduce/

https://towardsdatascience.com/log-book-guide-to-distance-measuring-approaches-for-k-means-clustering-f137807e8e21

https://www.sciencedirect.com/science/article/pii/S0031320319301608#sec0023

https://towardsdatascience.com/interpretable-k-means-clusters-feature-importances-7e516eeb8d3c

https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/