Introduction to K-Means Clustering

Machine learning algorithms can be broadly classified into two categories - supervised and unsupervised learning. There are other categories also like semi-supervised learning and reinforcement learning. But, most of the algorithms are classified as supervised or unsupervised learning. The difference between them happens because of presence of target variable. In unsupervised learning, there is no target variable. The dataset only has input variables which describe the data. This is called unsupervised learning.

K-Means clustering is the most popular unsupervised learning algorithm. It is used when we have unlabelled data which is data without defined categories or groups. The algorithm follows an easy or simple way to classify a given data set through a certain number of clusters, fixed apriori. K-Means algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

K-Means Clustering intuition

K-Means clustering is used to find intrinsic groups within the unlabelled dataset and draw inferences from them. It is based on centroid-based clustering.

Centroid - A centroid is a data point at the centre of a cluster. In centroid-based clustering, clusters are represented by a centroid. It is an iterative algorithm in which the notion of similarity is derived by how close a data point is to the centroid of the cluster. K-Means clustering works as follows:- The K-Means clustering algorithm uses an iterative procedure to deliver a final result. The algorithm requires number of clusters K and the data set as input. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the K centroids. The algorithm then iterates between two steps:-

1. Data assignment step

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, which is based on the squared Euclidean distance. So, if ci is the collection of centroids in set C, then each data point is assigned to a cluster based on minimum Euclidean distance.

1. Centroid update step

In this step, the centroids are recomputed and updated. This is done by taking the mean of all data points assigned to that centroid’s cluster.

The algorithm then iterates between step 1 and step 2 until a stopping criteria is met. Stopping criteria means no data points change the clusters, the sum of the distances is minimized or some maximum number of iterations is reached. This algorithm is guaranteed to converge to a result. The result may be a local optimum meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

Choosing the value of K

The K-Means algorithm depends upon finding the number of clusters and data labels for a pre-defined value of K. To find the number of clusters in the data, we need to run the K-Means clustering algorithm for different values of K and compare the results. So, the performance of K-Means algorithm depends upon the value of K. We should choose the optimal value of K that gives us best performance. There are different techniques available to find the optimal value of K. The most common technique is the elbow method which is described below.

The elbow method

The elbow method is used to determine the optimal number of clusters in K-means clustering. The elbow method plots the value of the cost function produced by different values of K.

If K increases, average distortion will decrease. Then each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids. However, the improvements in average distortion will decline as K increases. The value of K at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters.

The problem statement

In this project, We implement K-Means clustering with Python and Scikit-Learn. As mentioned earlier, K-Means clustering is used to find intrinsic groups within the unlabelled dataset and draw inferences from them. I have used Facebook Live Sellers in Thailand Dataset for this project. I implement K-Means clustering to find intrinsic groups within this dataset that display the same status_type behaviour. The status_type behaviour variable consists of posts of a different nature (video, photos, statuses and links).

Dataset description

In this project, I have used Facebook Live Sellers in Thailand Dataset, downloaded from the UCI Machine Learning repository. The dataset can be found at the following url-

https://archive.ics.uci.edu/ml/datasets/Facebook+Live+Sellers+in+Thailand

The dataset consists of Facebook pages of 10 Thai fashion and cosmetics retail sellers. The status_type behaviour variable consists of posts of a different nature (video, photos, statuses and links). It also contains engagement metrics of comments, shares and reactions.

Import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Ignore warnings

```
import warnings

warnings.filterwarnings('ignore')
```

Import dataset

```
dataset = pd.read_csv('Live_20210128.csv')
dataset.head()
```

```
   status_id status_type status_published  num_reactions  num_comments  \
0          1       video   4/22/2018 6:00            529           512   
1          2       photo  4/21/2018 22:45            150             0   
2          3       video   4/21/2018 6:17            227           236   
3          4       photo   4/21/2018 2:29            111             0   
4          5       photo   4/18/2018 3:22            213             0   

   num_shares  num_likes  num_loves  num_wows  num_hahas  num_sads  \
0         262        432         92         3          1         1   
1           0        150          0         0          0         0   
2          57        204         21         1          1         0   
3           0        111          0         0          0         0   
4           0        204          9         0          0         0   

   num_angrys  Column1  Column2  Column3  Column4  
0           0      NaN      NaN      NaN      NaN  
1           0      NaN      NaN      NaN      NaN  
2           0      NaN      NaN      NaN      NaN  
3           0      NaN      NaN      NaN      NaN  
4           0      NaN      NaN      NaN      NaN  
```

Exploring Data

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   status_id         7050 non-null   int64  
 1   status_type       7050 non-null   object 
 2   status_published  7050 non-null   object 
 3   num_reactions     7050 non-null   int64  
 4   num_comments      7050 non-null   int64  
 5   num_shares        7050 non-null   int64  
 6   num_likes         7050 non-null   int64  
 7   num_loves         7050 non-null   int64  
 8   num_wows          7050 non-null   int64  
 9   num_hahas         7050 non-null   int64  
 10  num_sads          7050 non-null   int64  
 11  num_angrys        7050 non-null   int64  
 12  Column1           0 non-null      float64
 13  Column2           0 non-null      float64
 14  Column3           0 non-null      float64
 15  Column4           0 non-null      float64
dtypes: float64(4), int64(10), object(2)
memory usage: 881.4+ KB
```

```
dataset.describe()
```

```
         status_id  num_reactions  num_comments   num_shares 
num_likes  \
count  7050.000000    7050.000000   7050.000000  7050.000000 
7050.000000   
mean   3525.500000     230.117163    224.356028    40.022553 
215.043121   
std    2035.304031     462.625309    889.636820   131.599965 
449.472357   
min       1.000000       0.000000      0.000000     0.000000 
0.000000   
25%    1763.250000      17.000000      0.000000     0.000000 
17.000000   
50%    3525.500000      59.500000      4.000000     0.000000 
58.000000   
75%    5287.750000     219.000000     23.000000     4.000000 
184.750000   
max    7050.000000    4710.000000  20990.000000  3424.000000 
4710.000000   

         num_loves     num_wows    num_hahas     num_sads   num_angrys 
\
count  7050.000000  7050.000000  7050.000000  7050.000000  7050.000000 

mean     12.728652     1.289362     0.696454     0.243688     0.113191 
```

```
std      39.972930     8.719650     3.957183     1.597156     0.726812

min       0.000000     0.000000     0.000000     0.000000     0.000000

25%       0.000000     0.000000     0.000000     0.000000     0.000000

50%       0.000000     0.000000     0.000000     0.000000     0.000000

75%       3.000000     0.000000     0.000000     0.000000     0.000000

max     657.000000   278.000000   157.000000    51.000000    31.000000
```

```
       Column1  Column2  Column3  Column4
count      0.0      0.0      0.0      0.0
mean       NaN      NaN      NaN      NaN
std        NaN      NaN      NaN      NaN
min        NaN      NaN      NaN      NaN
25%        NaN      NaN      NaN      NaN
50%        NaN      NaN      NaN      NaN
75%        NaN      NaN      NaN      NaN
max        NaN      NaN      NaN      NaN
```

```
dataset.shape
```

```
(7050, 16)
```

Checking For Missing Values

```
dataset.isnull().sum()
```

```
status_id              0
status_type            0
status_published       0
num_reactions          0
num_comments           0
num_shares             0
num_likes              0
num_loves              0
num_wows               0
num_hahas              0
num_sads               0
num_angrys             0
Column1             7050
Column2             7050
Column3             7050
Column4             7050
dtype: int64
```

Drop Unused Columns

```
dataset.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1,
inplace=True)

dataset.head()

   status_id status_type status_published  num_reactions  num_comments
\
0          1       video   4/22/2018 6:00            529           512

1          2       photo  4/21/2018 22:45            150             0

2          3       video   4/21/2018 6:17            227           236

3          4       photo   4/21/2018 2:29            111             0

4          5       photo   4/18/2018 3:22            213             0


   num_shares  num_likes  num_loves  num_wows  num_hahas  num_sads
num_angrys
0         262        432         92         3          1         1
0
1           0        150          0         0          0         0
0
2          57        204         21         1          1         0
0
3           0        111          0         0          0         0
0
4           0        204          9         0          0         0
0

dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   status_id         7050 non-null   int64
 1   status_type       7050 non-null   object
 2   status_published  7050 non-null   object
 3   num_reactions     7050 non-null   int64
 4   num_comments      7050 non-null   int64
 5   num_shares        7050 non-null   int64
 6   num_likes         7050 non-null   int64
 7   num_loves         7050 non-null   int64
 8   num_wows          7050 non-null   int64
 9   num_hahas         7050 non-null   int64
 10  num_sads          7050 non-null   int64
```

```
 11  num_angrys        7050 non-null   int64
dtypes: int64(10), object(2)
memory usage: 661.1+ KB
```

```
dataset.describe()
```

```
         status_id  num_reactions  num_comments   num_shares
num_likes  \
count  7050.000000    7050.000000   7050.000000  7050.000000
7050.000000
mean   3525.500000     230.117163    224.356028    40.022553
215.043121
std    2035.304031     462.625309    889.636820   131.599965
449.472357
min       1.000000       0.000000      0.000000     0.000000
0.000000
25%    1763.250000      17.000000      0.000000     0.000000
17.000000
50%    3525.500000      59.500000      4.000000     0.000000
58.000000
75%    5287.750000     219.000000     23.000000     4.000000
184.750000
max    7050.000000    4710.000000  20990.000000  3424.000000
4710.000000

         num_loves     num_wows    num_hahas     num_sads   num_angrys

count  7050.000000  7050.000000  7050.000000  7050.000000  7050.000000

mean     12.728652     1.289362     0.696454     0.243688     0.113191

std      39.972930     8.719650     3.957183     1.597156     0.726812

min       0.000000     0.000000     0.000000     0.000000     0.000000

25%       0.000000     0.000000     0.000000     0.000000     0.000000

50%       0.000000     0.000000     0.000000     0.000000     0.000000

75%       3.000000     0.000000     0.000000     0.000000     0.000000

max     657.000000   278.000000   157.000000    51.000000    31.000000
```

Explore status_id variable

```
# view the labels in the variable
dataset['status_id'].unique()
```

```
array([   1,    2,    3, ..., 7048, 7049, 7050], dtype=int64)
```

```
# view how many different types of variables are there
len(dataset['status_id'].unique())
```

```
7050
```

We can see that there are 6997 unique labels in the status_id variable. The total number of instances in the dataset is 7050. So, it is approximately a unique identifier for each of the instances. Thus this is not a variable that we can use. Hence, I will drop it.

Explore status_published variable

```
# view the labels in the variable

dataset['status_published'].unique()
```

```
array(['4/22/2018 6:00', '4/21/2018 22:45', '4/21/2018 6:17', ...,
       '9/21/2016 23:03', '9/20/2016 0:43', '9/10/2016 10:30'],
      dtype=object)
```

```
# view how many different types of variables are there

len(dataset['status_published'].unique())
```

```
6913
```

Again, we can see that there are 6913 unique labels in the status_published variable. The total number of instances in the dataset is 7050. So, it is also a approximately a unique identifier for each of the instances. Thus this is not a variable that we can use. Hence, I will drop it also.

Explore status_type variable

```
# view the labels in the variable

dataset['status_type'].unique()
```

```
array(['video', 'photo', 'link', 'status'], dtype=object)
```

```
# view how many different types of variables are there

len(dataset['status_type'].unique())
```

```
4
```

We can see that there are 4 categories of labels in the status_type variable.

Drop status_id and status_published variable from the dataset

```
dataset.drop(['status_id', 'status_published'], axis=1, inplace=True)
```

View the summary of dataset again

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   status_type    7050 non-null   object
 1   num_reactions  7050 non-null   int64 
 2   num_comments   7050 non-null   int64 
 3   num_shares     7050 non-null   int64 
 4   num_likes      7050 non-null   int64 
 5   num_loves      7050 non-null   int64 
 6   num_wows       7050 non-null   int64 
 7   num_hahas      7050 non-null   int64 
 8   num_sads       7050 non-null   int64 
 9   num_angrys     7050 non-null   int64 
dtypes: int64(9), object(1)
memory usage: 550.9+ KB
```

```
dataset.describe()
```

```
       num_reactions  num_comments   num_shares    num_likes
num_loves  \
count    7050.000000   7050.000000  7050.000000  7050.000000
7050.000000
mean      230.117163    224.356028    40.022553   215.043121
12.728652
std       462.625309    889.636820   131.599965   449.472357
39.972930
min         0.000000      0.000000     0.000000     0.000000
0.000000
25%        17.000000      0.000000     0.000000    17.000000
0.000000
50%        59.500000      4.000000     0.000000    58.000000
0.000000
75%       219.000000     23.000000     4.000000   184.750000
3.000000
max      4710.000000  20990.000000  3424.000000  4710.000000
657.000000

          num_wows    num_hahas     num_sads   num_angrys
count  7050.000000  7050.000000  7050.000000  7050.000000
mean      1.289362     0.696454     0.243688     0.113191
std       8.719650     3.957183     1.597156     0.726812
min       0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000
```

```
75%       0.000000     0.000000     0.000000     0.000000
max     278.000000   157.000000    51.000000    31.000000
```

Declare feature vector and target variable

```
X = dataset

y = dataset['status_type']
```

Convert categorical variable into integers

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X['status_type'] = le.fit_transform(X['status_type'])

y = le.transform(y)
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   status_type    7050 non-null   int32
 1   num_reactions  7050 non-null   int64
 2   num_comments   7050 non-null   int64
 3   num_shares     7050 non-null   int64
 4   num_likes      7050 non-null   int64
 5   num_loves      7050 non-null   int64
 6   num_wows       7050 non-null   int64
 7   num_hahas      7050 non-null   int64
 8   num_sads       7050 non-null   int64
 9   num_angrys     7050 non-null   int64
dtypes: int32(1), int64(9)
memory usage: 523.4 KB
```

```
X.head()
```

```
   status_type  num_reactions  num_comments  num_shares  num_likes
num_loves  \
0            3            529           512         262        432
92
1            1            150             0           0        150
0
2            3            227           236          57        204
21
3            1            111             0           0        111
```

```
0
4             1             213              0            0         204
9

   num_wows  num_hahas  num_sads  num_angrys
0         3          1         1           0
1         0          0         0           0
2         1          1         0           0
3         0          0         0           0
4         0          0         0           0
```

Feature Scaling

```
cols = X.columns

from sklearn.preprocessing import MinMaxScaler

ms = MinMaxScaler()

X = ms.fit_transform(X)

X = pd.DataFrame(X, columns=[cols])

X.head()

  status_type num_reactions num_comments num_shares num_likes
num_loves  \
0    1.000000      0.112314     0.024393   0.076519  0.091720
0.140030
1    0.333333      0.031847     0.000000   0.000000  0.031847
0.000000
2    1.000000      0.048195     0.011243   0.016647  0.043312
0.031963
3    0.333333      0.023567     0.000000   0.000000  0.023567
0.000000
4    0.333333      0.045223     0.000000   0.000000  0.043312
0.013699

   num_wows num_hahas  num_sads num_angrys
0  0.010791  0.006369  0.019608        0.0
1  0.000000  0.000000  0.000000        0.0
2  0.003597  0.006369  0.000000        0.0
3  0.000000  0.000000  0.000000        0.0
4  0.000000  0.000000  0.000000        0.0
```

K-Means model with two clusters

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0)
```

```
kmeans.fit(X)
```

```
KMeans(n_clusters=2, random_state=0)
```

K-Means model parameters study

```
kmeans.cluster_centers_
```

```
array([[3.28506857e-01, 3.90710874e-02, 7.54854864e-04, 7.53667113e-
04,
        3.85438884e-02, 2.17448568e-03, 2.43721364e-03, 1.20039760e-
03,
        2.75348016e-03, 1.45313276e-03],
       [9.54921576e-01, 6.46330441e-02, 2.67028654e-02, 2.93171709e-
02,
        5.71231462e-02, 4.71007076e-02, 8.18581889e-03, 9.65207685e-
03,
        8.04219428e-03, 7.19501847e-03]])
```

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variances, minimizing a criterion known as inertia, or within-cluster sum-of-squares Inertia, or the within-cluster sum of squares criterion, can be recognized as a measure of how internally coherent clusters are. The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean j of the samples in the cluster. The means are commonly called the cluster centroids. The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum of squared criterion. Inertia Inertia is not a normalized metric.

The lower values of inertia are better and zero is optimal.

But in very high-dimensional spaces, euclidean distances tend to become inflated (this is an instance of curse of dimensionality).

Running a dimensionality reduction algorithm such as PCA prior to k-means clustering can alleviate this problem and speed up the computations.

We can calculate model inertia as follows:-

```
kmeans.inertia_
```

```
237.7572640441955
```

Check quality of weak classification by the model

```
labels = kmeans.labels_

# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)
```

```
print("Result: %d out of %d samples were correctly labeled." %
(correct_labels, y.size))
```

```
Result: 63 out of 7050 samples were correctly labeled.
```

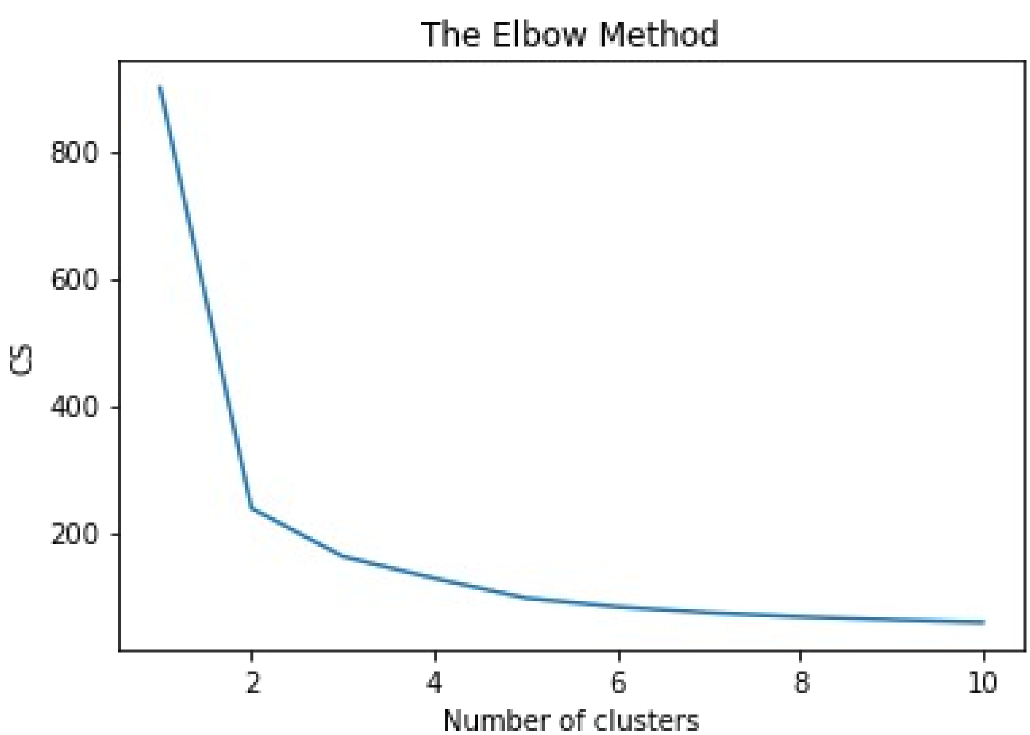
```
print('Accuracy score: {0:0.2f}'.
format(correct_labels/float(y.size)))
```

```
Accuracy score: 0.01
```

We have achieved a weak classification accuracy of 1% by our unsupervised model.

Use elbow method to find optimal number of clusters

```
from sklearn.cluster import KMeans
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter =
300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```



By the above plot, we can see that there is a kink at k=2.

Hence k=2 can be considered a good number of the cluster to cluster this data.

But, we have seen that I have achieved a weak classification accuracy of 1% with k=2.

I will write the required code with k=2 again for convinience.

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2,random_state=0)

kmeans.fit(X)

labels = kmeans.labels_

# check how many of the samples were correctly labeled

correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." %
(correct_labels, y.size))

print('Accuracy score: {0:0.2f}'.
format(correct_labels/float(y.size)))
```

```
Result: 63 out of 7050 samples were correctly labeled.
Accuracy score: 0.01
```

So, our weak unsupervised classification model achieved a very weak classification accuracy of 1%.

I will check the model accuracy with different number of clusters.

K-Means model with different clusters

K-Means model with 3 clusters

```
kmeans = KMeans(n_clusters=3, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." %
(correct_labels, y.size))
print('Accuracy score: {0:0.2f}'.
format(correct_labels/float(y.size)))
```

```
Result: 6 out of 7050 samples were correctly labeled.
Accuracy score: 0.00
```

K-Means model with 4 clusters

```
kmeans = KMeans(n_clusters=4, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." %
(correct_labels, y.size))
print('Accuracy score: {0:0.2f}'.
format(correct_labels/float(y.size)))
```

```
Result: 51 out of 7050 samples were correctly labeled.
Accuracy score: 0.01
```

We have achieved a relatively high accuracy of 62% with k=4.

1. Results and conclusion In this project, I have implemented the most popular unsupervised clustering technique called K-Means Clustering.

I have applied the elbow method and find that k=2 (k is number of clusters) can be considered a good number of cluster to cluster this data.

I have find that the model has very high inertia of 237.7572. So, this is not a good model fit to the data.

I have achieved a weak classification accuracy of 1% with k=2 by our unsupervised model.

So, I have changed the value of k and find relatively higher classification accuracy of 62% with k=4.

Hence, we can conclude that k=4 being the optimal number of clusters.