Rahmatur Ramadhan

1301184468

IF-42-INT

Clustering of Kendaraan Dataset using K-Means Algorithm

I. Problem Formulation

A. Task Description

Build a model to clustering kendaraan dataset by exploration and preprocessing the data. Then evaluate the model that has been build and perform various experiments involving the previous steps.

B. Dataset

This Dataset is about an Insurance company that has provided Health Insurance to its customers now where the task is to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the company. The attributes within the dataset according to the description that we can summarize are as follows:

1) id: Unique ID for the customer
2) Jenis_Kelamin: Gender of the customer.
3) Umur: Age of the customer.
4) SIM: Driving license of the customer.
5) Kode_Daerah: Unique code for the region of the customer.
6) Sudah_Asuransi: Vehicle Insurance of the customer.
7) Umur_Kendaraan: Age of the Vehicle
8) Kendaraan_Rusak: Customer got the vehicle damaged in the past.
9) Premi: The amount customer needs to pay as premium in the year.
10) Kanal_Penjualan: Anonymized Code for the channel of outreaching to the customer ie. Different Agents, Over Mail, Over Phone, In Person, etc.
11) Lama_Berlangganan: Number of Days, Customer has been associated with the company.
12) Tertarik: Customer is interested or not.

The label according to the description is ‘Tertarik’ feature (0 & 1). Total data count of 47,640 for test dataset and 285,831 for train dataset.

II. Data Exploration and Preprocessing

A. Data Exploration

Based on my exploration of the kendaraan dataset. I have found that several features within the dataset have missing values and duplicated rows which is shown in the figure below.

```
# drop column 'id'
df_pre = df.drop('id', axis=1)

# check the duplicate rows and missing values
print('Duplicated row count: %d' %df_pre.duplicated().sum())
print('Missing values\n%s' %df_pre.isna().sum())
```

✓ 0.3s

```
Duplicated row count: 169
Missing values
Jenis_Kelamin        14440
Umur                 14214
SIM                  14404
Kode_Daerah          14306
Sudah_Asuransi       14229
Umur_Kendaraan       14275
Kendaraan_Rusak      14188
Premi                14569
Kanal_Penjualan      14299
Lama_Berlangganan    13992
Tertarik                 0
```

Then check the datatypes also and has 3 features with *object* datatypes, there are Jeni_Kelamin, Umur_Kendaraan, and Kendaraan_Rusak.

```
# check features datatype
df_pre.dtypes
```

✓ 0.2s

```
Jenis_Kelamin         object
Umur                 float64
SIM                  float64
Kode_Daerah          float64
Sudah_Asuransi       float64
Umur_Kendaraan        object
Kendaraan_Rusak       object
Premi                float64
Kanal_Penjualan      float64
Lama_Berlangganan    float64
Tertarik               int64
```

B. Preprocessing

1) Handling Missing Values and Duplicates

In the kendaraan dataset, since the total data count is 285,831, we can drop rows that are duplicates or have missing values entirely. Which leaves the total data count of 171017.

```
# drop the duplicate rows and missing values
df_pre = df_pre.drop_duplicates()
df_pre = df_pre.dropna()
df_default = df_pre
df_pre
```

✓ 0.3s

	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan
0	Wanita	30.0	1.0	33.0	1.0	< 1 Tahun
1	Pria	48.0	1.0	39.0	0.0	> 2 Tahun
3	Wanita	58.0	1.0	48.0	0.0	1-2 Tahun
5	Pria	21.0	1.0	35.0	1.0	< 1 Tahun
8	Wanita	20.0	1.0	8.0	1.0	< 1 Tahun
...	...	...	...	...	...	...
285826	Wanita	23.0	1.0	4.0	1.0	< 1 Tahun
285827	Wanita	21.0	1.0	46.0	1.0	< 1 Tahun
285828	Wanita	23.0	1.0	50.0	1.0	< 1 Tahun
285829	Pria	68.0	1.0	7.0	1.0	1-2 Tahun
285830	Pria	45.0	1.0	28.0	0.0	1-2 Tahun

171017 rows × 11 columns

2) Encoding

For kendaraan datasets, I have implemented label encoding. Label encoding is an encoding technique where each label within the dataset is assigned with unique integer based on defined ordering or by default alphabetical ordering. I already implementing it for feature with object datatypes. The labels that are encode is shown in below.

```
# Encoding the data in column 'Jenis_Kelamin', 'Umur_Kendaraan', and 'Kendaraan_Rusak'
object_columns = ['Jenis_Kelamin', 'Umur_Kendaraan', 'Kendaraan_Rusak']
for column in object_columns:
    df_pre[column] = LabelEncoder().fit_transform(df_pre[column])
df_pre
```

✓ 0.1s

	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak
0	1	30.0	1.0	33.0	1.0	1	1
1	0	48.0	1.0	39.0	0.0	2	0
3	1	58.0	1.0	48.0	0.0	0	1
5	0	21.0	1.0	35.0	1.0	1	1
8	1	20.0	1.0	8.0	1.0	1	1
...	...	...	...	...	...	...	...
285826	1	23.0	1.0	4.0	1.0	1	1
285827	1	21.0	1.0	46.0	1.0	1	1
285828	1	23.0	1.0	50.0	1.0	1	1
285829	0	68.0	1.0	7.0	1.0	0	1
285830	0	45.0	1.0	28.0	0.0	0	0

171017 rows × 11 columns

3) Scaling

To prevent the feature with a higher value range starts dominating when calculating distances so I using standard scaler to standardization the dataset. Standard scales each input variable separately by subtracting the mean (called centering) and dividing by the standard deviation to shift the distribution to have a mean of zero and a standard deviation of one.

```
# Scaler the data with standard scaler
df_pre = StandardScaler().fit_transform(df_pre)
df_pre = pd.DataFrame(df_pre)
df_pre
```

✓ 0.1s

	0	1	2	3	4	5
0	1.084823	-0.567763	0.045544	0.496806	1.083930	0.834866
1	-0.921809	0.591112	0.045544	0.949222	-0.922569	2.566305
2	1.084823	1.234931	0.045544	1.627846	-0.922569	-0.896574
3	-0.921809	-1.147200	0.045544	0.647611	1.083930	0.834866
4	1.084823	-1.211582	0.045544	-1.388262	1.083930	0.834866
...	...	...	...	...	...	...
171012	1.084823	-1.018436	0.045544	-1.689872	1.083930	0.834866
171013	1.084823	-1.147200	0.045544	1.477041	1.083930	0.834866
171014	1.084823	-1.018436	0.045544	1.778652	1.083930	0.834866
171015	-0.921809	1.878750	0.045544	-1.463664	1.083930	-0.896574
171016	-0.921809	0.397966	0.045544	0.119792	-0.922569	-0.896574

171017 rows × 11 columns

4) Dimensionality Reduction

Because kendaraan dataset has multiple column to reduce the dimensionality, I using PCA method. By using PCA I convert the dataset into 2 columns.

```
# convert the multiple columns into two columns with PCA
pca = PCA(n_components=2)
df_pre = pca.fit_transform(df_pre)
df_pre = pd.DataFrame(df_pre)
df_pre
```

✓ 0.5s Python

	0	1
0	-2.044763	0.081549
1	0.884965	-0.802505
2	0.460075	0.691800
3	-2.006719	0.054067
4	-2.457453	-0.290917
...	...	...
171012	-2.345175	-0.202274
171013	-2.208928	-0.009156
171014	-2.131359	0.083778
171015	0.056628	2.122961
171016	2.057072	0.547299

171017 rows × 2 columns

III. Modelling

A) Initialize Centroid

Initialize the centroid by generate centroid randomly. For the random itself using np.random.random with the range 0 until total row in the data.

```
# use randomly initilialize the centroids
def initialize_centroids(self, data):
  init_centroids = []
  idx = []
  for _ in range(self.k):
    idx.append(np.random.randint(0,data.shape[0]))
  init_centroids = (X[idx])
  return init_centroids
```

B) Euclidean Distance

Calculate the distance of each data to the centroid. It is using Euclidean distance. and then define the members of each cluster based on the shortest distance. For the Euclidean distance I using np.linalg.norm(x1-x2) with x1 is the centroid and x2 is the data.

```
# calculate the distance using euclidean distance method
def euclidean_dist(x1,x2):
  return np.linalg.norm(x1-x2)
```

C) Fit

To find the optimal centroid find the member of the cluster. The following step is in this picture below.

```
def fit(self, data):
  self.clusters = {}
  #initialize the centroids randomly
  self.centroids = self.initialize_centroids(data)
  #assign each row to a centroid and recalculate centroids until there is no change or we reach the most iterations we wanna do
  iter = 0
  while iter < self.max_iter:
    #create the clusters (empty in the beginning)
    self.clusters.clear()

    #fill the clusters by adding the appropriate row to the cluster associated with the closest centroid
    for row in data:
      dist = []
      for i in range(len(self.centroids)):
        dist.append(self.euclidean_dist(self.centroids[i],row))
      idx = dist.index(min(dist))
      self.clusters.setdefault(idx,[]).append(list(row))

    #store the previous centroids
    old_centroids = self.centroids.copy()

    #recalculate the new centroids
    for centroid in range(len(self.centroids)):
      self.centroids[centroid] = np.average(self.clusters[centroid],axis=0)

    #check if the centroids have moved according to the amount of slack
    diff = []
    for centroid in range(len(self.centroids)):
      old_centroid = old_centroids[centroid]
      diff.append(np.sum(abs((self.centroids[centroid]-old_centroid))))

    if sum(diff) <= self.tol:
      break

    #increment number of iterations
    iter += 1

  print("Iterations:", iter)
  for k in range(self.k):
      self.clusters[k] = np.array(self.clusters[k])
```

D) Inertia

To calculate the sum of errors with measure the distance between the new centroid and members of the clusters.

```python
# to find the sum of square errors
def inertia(self):
    errors = []
    for i in range(len(self.centroids)):
        cluster_error = 0
        for j in range(len(self.clusters[i])):
            cluster_error += self.euclidean_dist(self.centroids[i],self.clusters[i][j])
        errors.append(cluster_error)
    return sum(errors)
```

IV. Evaluation

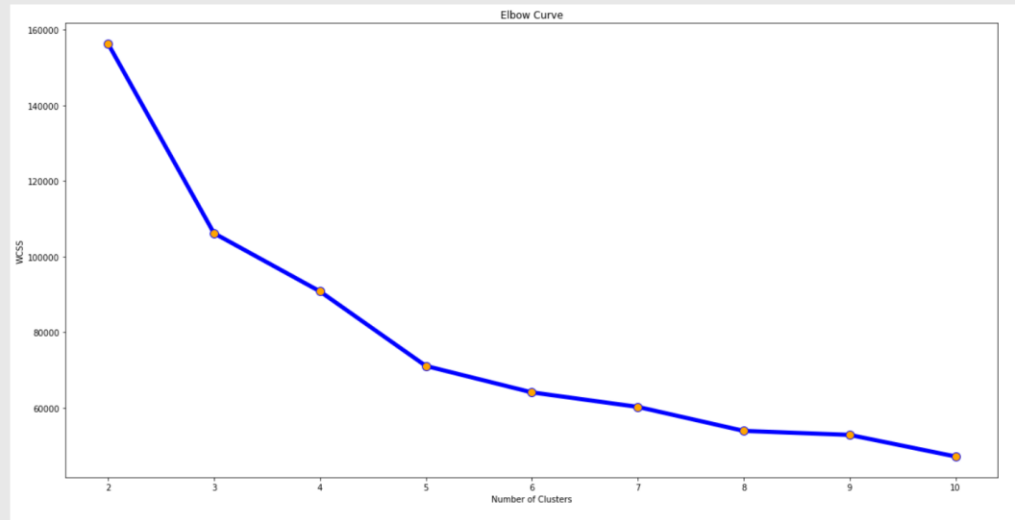
For evaluation process here are the steps:

1) Find the optimal K using elbow method with range 2-10

```python
k_models = []
k_range = range(2,11)
for i in k_range:
    print("K =",i)
    model = KMeans(k=i,max_iter=100,tol=0.001)
    model.fit(X)
    k_models.append(model)
```

```
K = 2
Iterations: 6
K = 3
Iterations: 8
K = 4
Iterations: 15
K = 5
Iterations: 14
K = 6
Iterations: 20
K = 7
Iterations: 16
K = 8
Iterations: 27
K = 9
Iterations: 23
K = 10
Iterations: 40
```

```python
sse = [k.inertia() for k in k_models]
plot_elbow(k_range,sse)
```



So, to define the elbow curve, I using kneelocator so the optimal K is 5

```
from kneed import KneeLocator

kl = KneeLocator(k_range,wcss,curve="convex",direction="decreasing")
print(kl.elbow)
opt = kl.elbow
```

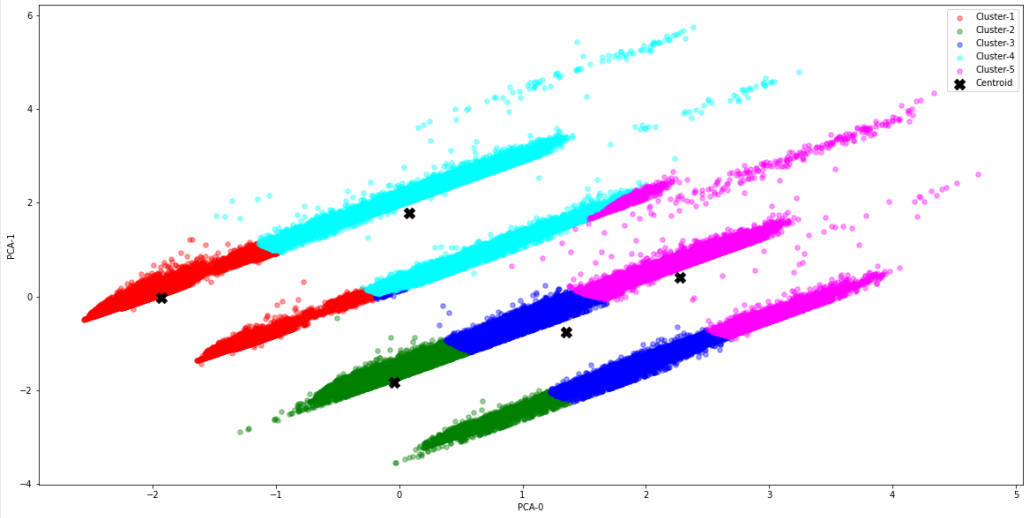
```
5
```

2) Plot the data using scatter plot with the optimal K from elbow method result.

```
# plot the model with optimal k
opt_k = k_models[opt-2]
plot_clusters(opt_k,x1,y1)
print(opt_k.centroids)
```

✓ 342.7s

```
Iterations: 32
[[-1.92571797 -0.02469764]
 [-0.04233433 -1.83391063]
 [ 1.35069964 -0.77074013]
 [ 0.08338331  1.77945783]
 [ 2.27440967  0.40624446]]
```



3) Group the cluster value

I am using ‘groupby’ method to group the cluster value and see the mean value of each of the attributes in the dataset using the ‘mean’ method.

```
# group the cluster value and see the mean value of each of the attributes in the dataset using the 'mean' method
df_default.groupby('Cluster').mean()
```

✓ 0.1s

Cluster	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi
1	0.580316	25.185495	1.000000	25.460701	0.912480	0.952749	0.964842	29938.221241
2	0.533258	27.752307	1.000000	25.913106	0.000700	1.072200	0.007172	28270.082389
3	0.408479	44.158262	1.000000	26.669139	0.000958	0.295606	0.007230	28878.188296
4	0.368538	51.243904	0.996227	27.120143	0.844871	0.020767	0.931837	30906.616246
5	0.317240	55.040807	0.991548	27.633247	0.009322	0.011281	0.010628	35018.026153

V. Conclusion

Based on my evaluation, the conclusion is every time I run the K-Means, the optimal K is always change. Even the same range and dataset.

From the optimal K which is K = 5 and by average data of the customer, we can get a demographic of the customer who interest to buy new vehicle, which are:

Cluster	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan
1	Laki-laki	25	Ada	25	Sudah	< 1 tahun
2	Laki-Laki	28	Ada	26	Belum	1-2 tahun
3	Perempuan	44	Ada	27	Belum	< 1 tahun
4	Perempuan	51	Ada	27	Sudah	< 1 tahun
5	Prempuan	55	Ada	28	Belum	< 1 tahun

Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
Pernah	29938.22	150.8465	154.546	Tidak
Tidak	28270.08	147.9727	153.8709	Tidak
Tidak	28878.19	112.3593	154.4257	Tidak
Pernah	30906.62	78.08678	154.239	Tidak
Tidak	35018.03	43.6685	153.7673	tidak