Obesity Data Categorization Using Single Linkage Divisive (Top-Down) Clustering Technique

Gajendra Singh

22ID60R20 [Project Code: OCHC-DS]

❖ Introduction

Clustered the dataset includes data for estimating obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition, into an optimal number of clusters using KMean clustering for different k and calculated the silhouette coefficient and find the optimal k. Done Hierarchical clustering (Divisive clustering) and Calculated Jaccard Similarity between both clusters formed by the KMean cluster and Divisive cluster.

❖ Explanation

- First, Data Pre-processing has been done on a given dataset. Drop the row containing the NA and duplicate values from the dataset. Then columns which contain categorical data are converted to numeric data using dummies for more than two categorical values and replace functions for two categorical values.

```
df=df.drop_duplicates() # Delete all the duplicate rows considering all column wise.
print("Final shape of data after preprocessing", df.shape, '\n')   # final shape of datadata
```

```
df['CAEC'].value_counts()                                    # converting categorical data to numeric data
df = pd.get_dummies(df, columns=['CAEC'])

df['Gender'].value_counts()                                  # converting categorical data to numeric data
df['Gender'].replace(['Male', 'Female'],[1, 0], inplace=True)
```

❖ KMean Clustering

- To perform KMean clustering, a class is defined. A function (fit_predict) is defined inside the class, taking X as an attribute. Assign cluster, move centroid and check finish, these 3 tasks are performed inside this function.

```
class KMeans:
    # variable assign
    def __init__(self, n_clusters, max_iter = 20):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.centroids = None
```

```
# function to make cluster of data
def fit_predict(self,X):
    random_index = random.sample(range(0,X.shape[0]), self.n_clusters)
    self.centroids = X[random_index]

    for i in range(self.max_iter):
        #assign cluster
        cluster_group = self.assign_cluster(X)
        old_centroids = self.centroids

        #move centroid
        self.centroids = self.move_centroids(X,cluster_group)


        #check finish
        if (old_centroids == self.centroids).all():
            break

    return cluster_group
```

- Function (assign cluster) makes the cluster on the basis of cosine similarity.

```
# Assign cluster according to cosine similarity
def assign_cluster(self,X):
```

- Function (move_centroids) takes the mean of the cluster and shift the centroid of the cluster.

```
# Assign new centroid after calculating mean of clustered data
def move_centroids(self,X,cluster_group):
```

- Function (euclidean_distance) is define to calculate distance between two data points.

```
# function to calculate euclidean distance
def euclidean_distance(self,x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
```

- Function (silhouette_coeficient) is defined to calculate the goodness of a clustering technique. Its value ranges from -1 to 1.
  1: Means clusters are well apart from each other and clearly distinguished.

  0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.

  -1: Means clusters are assigned in the wrong way.

  Silhouette Score = (b-a)/max(a,b)

```
# function to calculate silhouete coefficient of clustered data
def silhouette_coefficient(self,X, cluster_group):
```

- Clustered the data for k = 3,4,5,6. Silhouette coefficient is calculated for the same k.
- K, for which the maximum silhouette coefficient is taken as optimal k, which is 3 for this given data set.
- Store the final cluster information into a file name (kmeans_k.txt) for different values of k.

```
fp = open(f"kmeans_k{t}.txt", "w")        # file to store the clustered data

for cluster in clusters_KMean:

    s = ",".join(str(i) for i in cluster) + "\n"
    fp.write(s)                            # store the clustered data into file

fp.close()
```

❖ **Single Linkage Divisive (Top-Down) Clustering**

- A function (divisive_clustering) is defined. First, all data is initialized in one cluster.
- Then, Split the largest cluster in two using cosine similarity.
- Return a list of k clusters, each represented as a list of indices, into the data array.
- Store the final cluster information into a file name (divisive_k3.txt), considering the optimal number of clusters is 3.

```python
def divisive_clustering(data, k):

    # Convert data to unit vectors
    norms = np.linalg.norm(data, axis=1, keepdims=True)
    data = data / norms

    # Initialize with all data points in one cluster
    clusters_Hi = [list(range(data.shape[0]))]

    while len(clusters_Hi) < k:
        # Find the cluster with the largest SSE
        sses = [np.sum(np.var(data[cluster], axis=0)) for cluster in clusters_Hi]
        largest_sse_idx = np.argmax(sses)

        # Split the largest cluster in two using cosine similarity
        largest_cluster = np.array(clusters_Hi.pop(largest_sse_idx))
        similarities = np.dot(data[largest_cluster], data[largest_cluster].T)
        np.fill_diagonal(similarities, -1)
        max_similarity_idx = np.argmax(similarities)
        cluster1_idx, cluster2_idx = np.unravel_index(max_similarity_idx, similarities.shape)
        cluster1 = [largest_cluster[cluster1_idx]]
        cluster2 = [largest_cluster[cluster2_idx]]
        for i, point in enumerate(largest_cluster):
            if i not in {cluster1_idx, cluster2_idx}:
                if similarities[i, cluster1_idx] > similarities[i, cluster2_idx]:
                    cluster1.append(point)
                else:
                    cluster2.append(point)

        # Add the new clusters to the list of clusters
        clusters_Hi.append(cluster1)
        clusters_Hi.append(cluster2)

    return clusters_Hi            # Returns a list of k clusters, each represented as a list of indices into the data array.
```

❖ **Jaccard Similarity**

- A function (jaccard_binary) is defined, which takes two arguments, clusters from KMean clustering and clusters from divisive clustering. Map each set of case A to a distinct set of case B (one-to-one and onto mapping) considering the Jaccard similarity.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- print the Jaccard Similarity scores for all the k mappings
- The Jaccard coefficient can be a value between 0 and 1, with 0 indicating no overlap and one complete overlap between the sets.

```python
def jaccard_binary(x,y):
    li = []
    li1 = []
    for i in range(len(x)):
        for j in range(len(y)):
            b = 0
            # intersection = np.logical_and(x[i], y[j])
            # union = np.logical_or(x[i], y[j])
            # similarity = intersection.sum() / float(union.sum())

            intersection = len(list(set(x[i]).intersection(y[j])))
            union = (len(x[i]) + len(y[j])) - intersection
            similarity = float(intersection) / union



            li.append(similarity)
        b = max(li)
        li1.append(b)
    return li1
```

❖ Conclusion

- The optimal number of clusters = **3**

- Analysis of the results about similarity coefficients in steps two and step 4:
  After KMean clustering: -
  - ➢ Silhouette coefficient for k = 3 is 0.18270572106955835
  - ➢ Silhouette coefficient for k = 4 is 0.07464257887902559
  - ➢ Silhouette coefficient for k = 5 is 0.0558536252610166
  - ➢ Silhouette coefficient for k = 6 0.042586674286763916

  - **The optimum k (no of cluster) is 3, having a silhouette coefficient of 0.18270572106955835.**

  - ✓ Silhouette coefficient for optimum cluster (k = 3) from KMean clustering is 0.18270572106955835
  - ✓ Silhouette coefficient after the divisive clustering for optimum clusters (k = 3) is -0.3328860180532363

- ➢ The Jaccard similarity between corresponding sets of both cases is [0.2784313725490196, 0.5089086859688196, 0.5089086859688196].

As the variation is more between silhouette coefficient for k = 3 for Kmean clustering and divisive clustering, Jaccard similarity comes out less due to less overlap between set of different cases.

- Approximate time taken by your program to run all the steps in a reasonable PC configuration is 92.312 seconds

```
Final shape of data after preprocessing (2087, 16)

Silhouette coefficient for k = 3 after KMean clustering, is 0.18270572106955835
Silhouette coefficient for k = 4 after KMean clustering, is 0.07464257887902559
Silhouette coefficient for k = 5 after KMean clustering, is 0.0558536252610166
Silhouette coefficient for k = 6 after KMean clustering, is 0.042586674286763916


Optimum k (no of cluster) is 3, having silhouette coefficient 0.18270572106955835

 silhouette coefficient after divisive clustering for optimum cluseters is -0.3328860180532363.


Jaccard similarity between corresponding sets of both the cases is [0.2784313725490196, 0.5089086859688196, 0.5089086859688196].

[Done] exited with code=0 in 92.312 seconds
```