**Customer Segmentation**

**Using Python**

**Declaration**

This is to certify that the project work entitled **“CUSTOMER SEGMENTATION”** is being submitted to **EXPOSYS DATA LABS** I hold the responsibility for the originality of the work incorporated into this thesis.

Signature of the candidate

Name: P. Ramakrishna

Date: 11-02-2021

**ABSTRACT**

 Today's business run based on such innovation having the ability to enthrall the customers with the products, but with such a large raft of products leave the customers confounded, what to buy and what to not and the companies are nonplussed about what section of customers to target to sell their products. This is where machine learning comes into play, various algorithms are applied for unraveling the hidden patterns in the data for better decision making for the future. This elude concept of which segment to target is made unequivocal by applying segmentation. The process of segmenting the customers with similar behaviors into the same segment and with different patterns into different segments is called customer segmentation. In this paper, clustering algorithms (k-Means) are been implemented to segment the customers and finally compare the results of clusters obtained from the algorithms. A python program has been developed and the program is been trained by applying standard scaler onto a dataset having two features of 200 training samples taken from the local retail shop. Both the features are the mean of the amount of shopping by customers and average of the customer's visit into the shop annually. By applying clustering, 5 segments of a cluster have been formed labeled as Careless, Careful, Standard, Target, and Sensible customers. However, two new clusters emerged on applying mean shift clustering labeled as High buyers and frequent visitors and High buyers and occasional visitors.

**Table of Contents**

**1. Introduction…………………………………………………………………. 5**

**2. Existing Method……………………………………………………………... 6**

**3. Proposed method with Architecture………………………………………... 8**

**4. Methodology………………………………………………………………….. 9**

**5. Implementation………………………………………………………………. 11**

**6. Conclusion……………………………………………………………………. 22**

1. **Introduction**

Today's business run based on such innovation having the ability to enthrall the customers with the products, but with such a large raft of products leave the customers confounded, what to buy and what to not and the companies are nonplussed about what section of customers to target to sell their products. This is where machine learning comes into play, various algorithms are applied for unraveling the hidden patterns in the data for better decision making for the future. This elude concept of which segment to target is made unequivocal by applying segmentation. The process of segmenting the customers with similar behaviors into the same segment and with different patterns into different segments is called customer segmentation. In this paper, clustering algorithms (k-Means) are been implemented to segment the customers and finally compare the results of clusters obtained from the algorithms. A python program has been developed and the program is been trained by applying standard scaler onto a dataset having two features of 200 training samples taken from a local retail shop. Both the features are the mean of the amount of shopping by customers and average of the customer's visit into the shop annually. By applying clustering, 5 segments of a cluster have been formed labeled as Careless, Careful, Standard, Target, and Sensible customers. However, two new clusters emerged on applying mean shift clustering labeled as High buyers and frequent visitors and High buyers and occasional visitors.

**K-Means:**

Clustering algorithms are unsupervised algorithms but are like Classification algorithms, but the basis is different.

In Clustering, you do not know what you are looking for, and you are trying to identify some segments or clusters in your data. When you use clustering algorithms in your dataset, unexpected things can suddenly pop-up like structures, clusters, and groupings you would have never thought otherwise.

**K-Means clustering**

**K-Means clustering** algorithm is an unsupervised algorithm, and it is used to segment the interest area from the background. It clusters or partitions the given data into K-clusters or parts based on the K-centroids.

The algorithm is used when you have unlabeled data (i.e., data without defined categories or groups). The goal is to find certain groups based on similarity in the data with the number of groups represented by K.

**2. Existing Method**

**Elbow Method**

The Elbow method gives us an idea of what a good *k* number of clusters would be based on the sum of squared distance (SSE) between data points and their assigned clusters’ centroids. We pick *k* at the spot where SSE starts to flatten out and forming an elbow. We will use the geyser dataset and evaluate SSE for different values of *k* and see where the curve might form an elbow and flatten out.

**Code**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset=pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:,[3,4]].values

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('Sum of squared distance')

plt.show()

Chart, line chart

Description automatically generated

Fig: The Elbow Method

The graph above shows that k=5 is not a bad choice. Sometimes it is still hard to figure out a good number of clusters to use because the curve is monotonically decreasing and may not show any elbow or has an obvious point where the curve starts flattening out.

**3.Proposed method with Architecture**

**Proposed method with Architecture**

**Elbow Method**

The Elbow method gives us an idea of what a good *k* number of clusters would be based on the sum of squared distance (SSE) between data points and their assigned clusters’ centroids. We pick *k* at the spot where SSE starts to flatten out and forming an elbow. We will use the geyser dataset and evaluate SSE for different values of *k* and see where the curve might form an elbow and flatten out.

Elbow Method is Finding the Number of Cluster’s on Data set. But That cluster’s is We are Validating the Silhouette Analysis Method it is Best K choosing.

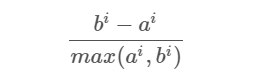
**Silhouette Analysis Validating Method**

Silhouette analysis can be used to determine the degree of separation between clusters. For each sample:

Compute the average distance from all data points in the same cluster (ai).

Compute the average distance from all data points in the closest cluster (bi).

Compute the coefficient:



The coefficient can take values in the interval [-1, 1].

If it is 0 –> the sample is very close to the neighboring clusters.

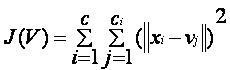
It is 1 –> the sample is far away from the neighboring clusters.

If it is -1 –> the sample is assigned to the wrong clusters.

Therefore, we want the coefficients to be as big as possible and close to 1 to have a good cluster. We will use here geyser dataset again because it is cheaper to run the silhouette analysis and it is obvious that there are most likely only two groups of data points.

**4. Methodology**

**k-means** is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriority. The main idea is to define k centers, one for each cluster. These centers should be placed cunningly because of different location causes a different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest canter. When no point is pending, the first step is completed, and an early group age is done. At this point, we need to re-calculate k new centroids as barycentre of the clusters resulting from the previous step. After we have these k new centroids, a new binding must be done between the same data set points and the nearest new canter. A loop has been generated. As a result of this loop, we may notice that the k centers change their location step by step until no more changes are done or in other words, centers do not move anymore. Finally, this algorithm aims at minimizing an objective function know as the squared error function given by:



were, *‘||xi - vj||’* is the Euclidean distance between *xi* and *VJ.* *‘ci’* is the number of data points in the *ith* cluster.

*‘c’* is the number of cluster centers.

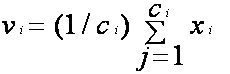
**Algorithmic steps for k-means clustering.**

Let X = {x1, x2, x3, ……... ,xn} be the set of data points and V = {v1,v2,…….,vc} be the set of centres.

1) Randomly select *‘c’* cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster canter whose distance from the cluster canter is the minimum of all the cluster centers.

4) Recalculate the new cluster canter using:

where *‘ci’* represents the number of data points in the *ith* cluster.

5) Recalculate the distance between each data point and new obtained cluster centres.

6) If no data point was reassigned then stop, otherwise repeat from step 3).

**Silhouette Method**

K-Means Clustering Algorithm Best method is the Elbow method The method we are validating using one of the methods is Silhouette.

Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like some clusters visually. This measure has a range of [-1, 1].

Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

In this example, the silhouette analysis is used to choose an optimal value for n\_clusters. The silhouette plot shows that the n\_clusters value of 3, 5, and 6 are a bad pick for the given data due to the presence of clusters with below-average silhouette scores and due to wide fluctuations in the size of the silhouette plots. Silhouette's analysis is more ambivalent in deciding between 2 and 4.

Also, from the thickness of the silhouette plot the cluster size can be visualized. The silhouette plot for cluster 0 when n\_clusters is equal to 2, is bigger owing to the grouping of the 3 sub-clusters into one big cluster. However, when the n\_clusters is equal to 4, all the plots are more or less of similar thickness and hence are of similar sizes as can be also verified from the labeled scatter plot on the right.

**5. Implementation**

**Requirements:**

**System Requirements**

| Hardware requirements | |
| --- | --- |
| Item | Requirements |
| Dedicated server | |
| Redundant power | |
| Hard disk space | Disk space storage requirements (approximately 500 GB total, using RAID 1 mirroring)  200GB storage file system for /usr/local/chorus  200GB storage file system for /data/chorus  50GB storage file system for /tmp  50GB storage file system for /home/chorus |
| Processor | Quad-core or higher |
| RAM | Minimum: 48 GB.  Recommended: 48+ GB (more recommended for improved performance). |
| Tested operating systems | |
| Tested operating system | Tested versions[1](https://docs.tibco.com/pub/sfire-dsc/6.5.0/doc/html/TIB_sfire-dsc_sys-req/" \l "fntarg_1) |
| Red Hat Enterprise | 6.2 - 7.6 (64-bit) |
| CentOS | 6.2, 6.5, 7.0, 7.6 (64-bit) |

| Required software | |
| --- | --- |
| Installed on the server | Notes |
| Oracle Java 1.8 (JVM 64-bit) - See the [Oracle download page](http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html). | This JDK is used by chorus users, not all users.  Note: OpenJDK Java is not supported. |
| Bash Unix Shell | If you want to use another shell, contact technical support. |
| Tested third-party and TIBCO tools | |
| Tool | Tested version |
| TIBCO Spotfire Analyst, TIBCO Spotfire Desktop (using the Export to SBDF operator) | 7.x |
| Tableau Server | 10.3 |
| MADlib | 1.15.1 |
| PMML | 1.3 |
| open-source R | 3.5 |

| Supported browsers | |
| --- | --- |
| Browser | Version |
| Google® Chrome | 75.0.3770 |
| Mozilla® Firefox | 68 |
| Microsoft® Internet Explorer | 11 |

**Jupyter Notebooks Requirements**

Ensure your environment meets the following requirements for using Jupyter Notebooks for Team Studio.

These recommendations are based on user count. For example, if you have 10 users, then you get (510MB \* 10) + 2GB RAM, 10GB + 10GB disk space, and (10\*.5) + .5 CPU cores. This projected use computes out to about 8GB of RAM, 20GB of free disk space, and 6 CPU cores.

**With PySpark (Team Studio version 6.2 and later)**

* Memory and disk space required per user: 1GB RAM + 1GB of disk + .5 CPU core.
* Server overhead: 2-4GB or 10% system overhead (whatever is larger), .5 CPU cores, 10GB disk space.
* Port requirements: Port 8000 plus 5 unique, random ports per notebook.

**Without PySpark (Team Studio version 6.0 or 6.1)**

* Memory and disk space required per user: 512MB RAM + 1GB of disk + .5 CPU core.
* Server overhead: 2-4GB or 10% system overhead (whatever is larger), .5 CPU cores, 10GB disk space.
* Port requirements: Port 8000.

**Implementation Code:**

**Packages Importing**

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_samples, silhouette\_score

import matplotlib.pyplot as plt

import matplotlib.cm as cm

import numpy as np

import pandas as pd

**Dataset**

dataset=pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:,[3,4]].values

**Elbow Method In K-Means finding Number of Cluster.**

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

Chart, line chart

Description automatically generated

//We take 5 Clusters Based on Elbow Method

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X)

plt.scatter(X[:,0],X[:,1])

plt.title('Customers data')

plt.xlabel('Annual Income (k--$)')

plt.ylabel('Spending Score (1-100)')

plt.show()

**Chart, scatter chart

Description automatically generated**

**Clustering**

clusterer = KMeans(n\_clusters=5, random\_state=10)

cluster\_labels = clusterer.fit\_predict(X)

print(cluster\_labels)

O/p

[3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3

0 3 0 3 0 3 1 3 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 2 4 2 1 2 4 2 4 2 1 2 4 2 4 2 4 2 4 2 1 2 4 2 4 2

4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4

2 4 2 4 2 4 2 4 2 4 2 4 2 4 2]

**Elbow Method using Clusters Verifying by using Silhouette.**

range\_n\_clusters = [2, 3, 4, 5, 6 ]

for n\_clusters in range\_n\_clusters:

# Create a subplot with 1 row and 2 columns

fig, (ax1, ax2) = plt.subplots(1, 2)

fig.set\_size\_inches(18, 7)

# The 1st subplot is the silhouette plot

# The silhouette coefficient can range from -1, 1 but in this example all

# lie within [-0.1, 1]

ax1.set\_xlim([-0.1, 1])

# The (n\_clusters+1)\*10 is for inserting blank space between silhouette

# plots of individual clusters, to demarcate them clearly.

ax1.set\_ylim([0, len(X) + (n\_clusters + 1) \* 10])

# Initialize the clusterer with n\_clusters value and a random generator

# seed of 10 for reproducibility.

clusterer = KMeans(n\_clusters=n\_clusters, random\_state=10)

cluster\_labels = clusterer.fit\_predict(X)

# The silhouette\_score gives the average value for all the samples.

# This gives a perspective into the density and separation of the formed

# clusters

silhouette\_avg = silhouette\_score(X, cluster\_labels)

print("For n\_clusters =", n\_clusters,

"The average silhouette\_score is :", silhouette\_avg)

# Compute the silhouette scores for each sample

sample\_silhouette\_values = silhouette\_samples(X, cluster\_labels)

y\_lower = 10

for i in range(n\_clusters):

# Aggregate the silhouette scores for samples belonging to

# cluster i, and sort them

ith\_cluster\_silhouette\_values = \

sample\_silhouette\_values[cluster\_labels == i]

ith\_cluster\_silhouette\_values.sort()

size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]

y\_upper = y\_lower + size\_cluster\_i

color = cm.nipy\_spectral(float(i) / n\_clusters)

ax1.fill\_betweenx(np.arange(y\_lower, y\_upper),

0, ith\_cluster\_silhouette\_values,

facecolor=color, edgecolor=color, alpha=0.7)

# Label the silhouette plots with their cluster numbers at the middle

ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))

# Compute the new y\_lower for next plot

y\_lower = y\_upper + 10 # 10 for the 0 samples

ax1.set\_title("The silhouette plot for the various clusters.")

ax1.set\_xlabel("The silhouette coefficient values")

ax1.set\_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values

ax1.axvline(x=silhouette\_avg, color="red", linestyle="--")

ax1.set\_yticks([]) # Clear the yaxis labels / ticks

ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed

colors = cm.nipy\_spectral(cluster\_labels.astype(float) / n\_clusters)

ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,

c=colors, edgecolor='k')

# Labeling the clusters

centers = clusterer.cluster\_centers\_

# Draw white circles at cluster centers

ax2.scatter(centers[:, 0], centers[:, 1], marker='o',

c="white", alpha=1, s=200, edgecolor='k')

for i, c in enumerate(centers):

ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,

s=50, edgecolor='k')

ax2.set\_title("The visualization of the clustered data.")

ax2.set\_xlabel("Feature space for the 1st feature")

ax2.set\_ylabel("Feature space for the 2nd feature")

plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "

"with n\_clusters = %d" % n\_clusters),

fontsize=14, fontweight='bold')

plt.show()

Chart, scatter chart

Description automatically generatedChart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart

Description automatically generated

Chart, scatter chart

Description automatically generated

**Best Model is 5 Cluster Based on Silhouette n\_clusters = 6**

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 70, c = 'red', label = 'Target')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 70, c = 'green', label = 'Motivate')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 70, c = 'blue', label = 'Discount')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 70, c = 'yellow', label = 'Fans')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 70, c = 'pink', label = 'Casual')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 150, c = 'black', label = 'Centroids')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

Chart, scatter chart, bubble chart

Description automatically generated

**6. Conclusion**

**Target Peoples** is Red Colour marking

**Motivate Peoples** is Green we give these peoples Offers Attracts!

**Discount Expecting Peoples** we give those peoples Discount Blue color marking.

**Fans** They are Spending high Amounts that’s the way they Regular and Fans Representing Yellow color marking.

**Casual** they are high more than regular.