Assignment Project Report

Gaussian Mixture Models: Bag of Words Representation

Name: Arun Govind
Course: AI and ML
(Batch 4)

- **Problem Statement**

  Implement GMM clustering on a blob of data

- **Prerequisites**

  - Software:
    - Python 3 (Use anaconda as your python distributor as well)

  - Tools:
    - Numpy
    - Pandas
    - Matplotlib
    - Sklearn
    - Scipy

  - Dataset: Analytics Vidya Clustering GMM

- **Method Used**

  In statistics, a mixture model is a probabilistic model for representing the presence of subpopulations within an overall population, without requiring that an observed data set should identify the sub-population to which an individual observation belongs.

  A Gaussian mixture model (GMM) attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset. In the simplest case, GMMs can be used for finding clusters in the same manner as k-means.
  Under the hood, a Gaussian mixture model is very similar to k-means.
  It uses an expectation–maximization approach which qualitatively does the following:
  a. Choose starting guesses for the location and shape
  b. Repeat until converged

- **Implementation:**

1. Load all required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
sns.set()
```

2. After calling dataset, use elbow method for identifying the number of clusters.

```
: wcss=[]
for i in range(1,7):
    kmeans = KMeans(i)
    kmeans.fit(x)
    wcss_iter = kmeans.inertia_
    wcss.append(wcss_iter)

number_clusters = range(1,7)
plt.plot(number_clusters,wcss)
plt.title('The Elbow title')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

3. Implementing K-Means clustering

```
from sklearn.cluster import KMeans
kmeans = KMeans(4)
kmeans.fit(x)
```

```
KMeans(n_clusters=4)
```

```
identified_clusters = kmeans.fit_predict(x)
```

```
data_with_clusters = data.copy()
data_with_clusters['Clusters'] = identified_clusters
plt.scatter(data_with_clusters['Height'],data_with_clusters['Weight'],c=data_with_clusters['Clusters'])
```

```
<matplotlib.collections.PathCollection at 0x7fb1b9280d30>
```

4. Implementing GMM

```
from sklearn.mixture import GaussianMixture as GMM
gmm = GMM(n_components=4).fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')
plt.show()
```

This returns a matrix of size [n_samples, n_clusters] which measures the probability that any point belongs to the given cluster.

5. Graphing GMM clustering Results

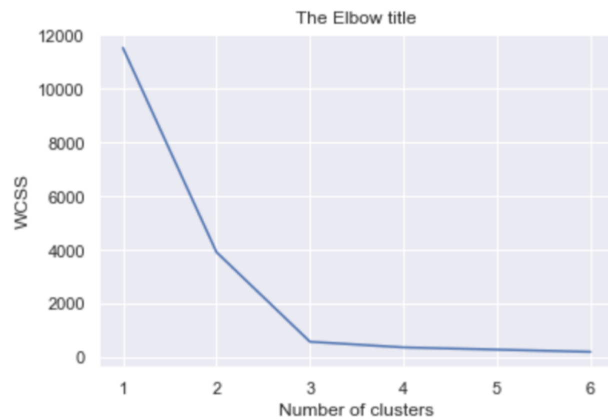
```
from matplotlib.patches import Ellipse

def draw_ellipse(position,covariance,**kwargs):
    ax = plt.gca()
    if covariance.shape == (2,2):
        U,s,Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1,0],U[0,0]))
        width,height = 2* np.sqrt(s)
    else:
        angle =0
        width,height = 2*np.sqrt(covariance)

    for n in range(1,4):
        ax.add_patch(Ellipse(position,n*width,n*height,angle, **kwargs))

def plot_gmm(gmm, X, label=True):
    ax = plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```

```
gmm = GMM(n_components=4, covariance_type='full', random_state=42)
rng = np.random.RandomState(13)
X_stretched = np.dot(X, rng.randn(2, 2))
plot_gmm(gmm, X_stretched)
```

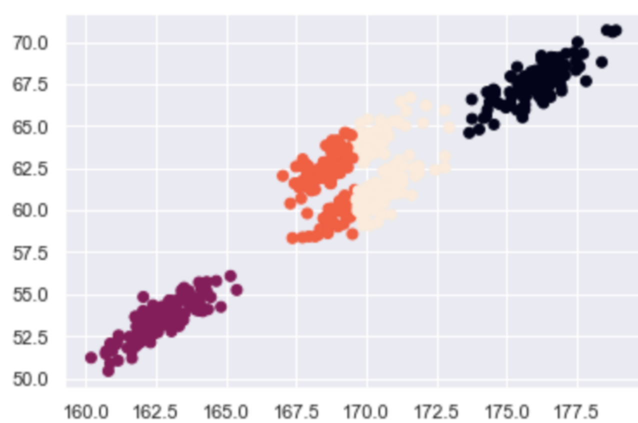
GMM approach to fit our stretched dataset; allowing for a full covariance the model will fit even very oblong, stretched-out clusters.

- **Results:**

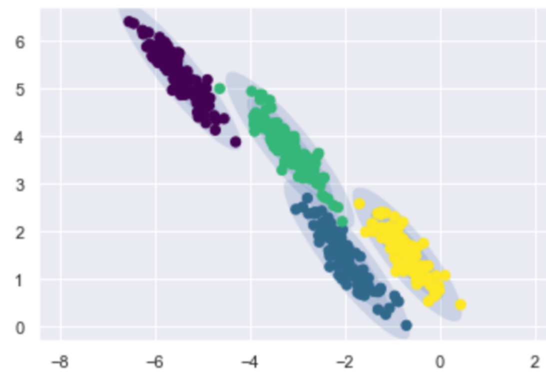
1. Elbow method:



2. Resulting graph for K-Means Clustering:



3. Resulting graph from GMM clustering:


6
5
4
3
2
1
0
−8
−6
−4
−2
0
2