Coffee Classification and Clustering Challenge Report

- Ujwal Kavalipati (kavalipatiujwal@gmail.com)

Contents

Code Challenge- Coffee Classification and Clustering:

This document is mainly written to show the different types of Machine learning algorithms I tried for building the classification and clustering model. I will also compare the results and state the reason for choosing particular models. The link to the Heroku app is https://coffee-prediction.herokuapp.com/ .I have deployed classification model on the server.

If you are directly seeing this file, I would first encourage you to follow the instructions in https://github.com/UjwalKavalipati/Coffee-Classification-Challenge and run the file **Coffee Classification and Clustering. ipynb** in your jupyter notebook.

I have clearly explained all the steps I followed for building the model by using famous libraries in python. Let me re-iterate all the steps I followed briefly so that it will be clear to you.

The steps followed in doing this task are:

- **Loading the dataset**: I used the cleaned data from https://github.com/jldbc/coffee-quality-database/tree/master/data extracted from https://database.coffeeinstitute.org/ by James LeDoux. (https://github.com/jldbc). The main features I used for building the models are by default cleaned in the raw dataset also.
- **Exploratory Data Analysis**: I have done basic EDA by plotting some graphs to understand the data and find the missing values.
- **Feature Engineering**: I tried encoding the categorical features by one-hot encoding and selected the most frequently occurred values. I did this to understand the feature's importance.
- **Feature Selection**: I selected the features with high importance at this stage for our model and removed other features. I have also applied over-sampling to balance the dataset.
- **Model Building and Evaluation**: I tried different algorithms and selected the best ones for both classification and clustering. Then, I computed the performance metrics like Precision, Recall, F1- score, and Matthews Correlation Coefficient.
- **Model Deployment**: I deployed the model on Heroku platform and created a basic front-end framework for testing the model.

Classification Algorithms:

This section talks about the different machine learning algorithms I applied for building a classification model.

1. Logistic Regression:

In [43]:
```
np.random.seed(99)
model = LogisticRegression(max_iter=1500)
model.fit(X_res,y_res)
best_preds = model.predict(X_test)
#Performance metrics
print("Precision = {}".format(precision_score(y_test, best_preds, average='macro')))
print("Recall = {}".format(recall_score(y_test, best_preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(y_test, best_preds)))
print("f1_score = {}".format(f1_score(y_test, best_preds)))
print("Matthews Correlation Coefficient = {}".format(matthews_corrcoef(y_test, best_preds)))

#print the classfication report
print("\n")
print("----------------Classification report---------------------------")
print(classification_report(y_test,best_preds,target_names=['Robusta','Arabica']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test,best_preds))

```

```
Precision = 0.8571428571428572
Recall = 0.9961977186311788
Accuracy = 0.9925373134328358
f1_score = 0.9961832061068702
Matthews Correlation Coefficient = 0.841934607768814


----------------Classification report---------------------------
              precision    recall  f1-score   support

     Robusta       0.71      1.00      0.83         5
     Arabica       1.00      0.99      1.00       263

    accuracy                           0.99       268
   macro avg       0.86      1.00      0.91       268
weighted avg       0.99      0.99      0.99       268


Confusion Matrix:
[[  5   0]
 [  2 261]]
```

The precision of Logistic Regression for Robusta is 0.71 which is low. 2 samples of Robusta are incorrectly classified as Arabica.

2. Decision Tree Classifier

```
1  np.random.seed(10)
2  model_dec = DecisionTreeClassifier()
3  model_dec.fit(X_res, y_res)
4  best_preds = model_dec.predict(X_test)
5  #Performance metrics
6  print("Precision = {}".format(precision_score(y_test, best_preds, average='macro')))
7  print("Recall = {}".format(recall_score(y_test, best_preds, average='macro')))
8  print("Accuracy = {}".format(accuracy_score(y_test, best_preds)))
9  print("f1_score = {}".format(f1_score(y_test, best_preds)))
10 print("Matthews Correlation Coefficient = {}".format(matthews_corrcoef(y_test, best_preds)))
11
12 #print the classfication report
13 print("\n")
14 print("----------------Classification report---------------------------")
15 print(classification_report(y_test,best_preds,target_names=['Robusta','Arabica']))
16
17 print("\nConfusion Matrix:")
18 print(confusion_matrix(y_test,best_preds))
19
```

```
Precision = 0.8125
Recall = 0.9942965779467681
Accuracy = 0.9888059701492538
f1_score = 0.994263862332696
Matthews Correlation Coefficient = 0.7860475319174154


----------------Classification report---------------------------
              precision    recall  f1-score   support

     Robusta       0.62      1.00      0.77         5
     Arabica       1.00      0.99      0.99       263

    accuracy                           0.99       268
   macro avg       0.81      0.99      0.88       268
weighted avg       0.99      0.99      0.99       268


Confusion Matrix:
[[  5   0]
 [  3 260]]
```

The Precision of the Decision Tree for Robusta is just 0.62. 3 samples of Robusta are incorrectly classified as Arabica.

3. Random Forest Classifier

```
#declare the model with best features
rfc1=RandomForestClassifier(random_state=93, max_features='auto', criterion='gini')
rfc1.fit(X_res, y_res)
best_preds= rfc1.predict(X_test)
#Performance metrics
print("Precision = {}".format(precision_score(y_test, best_preds, average='macro')))
print("Recall = {}".format(recall_score(y_test, best_preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(y_test, best_preds)))
print("f1_score = {}".format(f1_score(y_test, best_preds)))
print("Matthews Correlation Coefficient = {}".format(matthews_corrcoef(y_test, best_preds)))

#print the classfication report
print("\n")
print("----------------Classification report--------------------------")
print(classification_report(y_test,best_preds,target_names=['Robusta','Arabica']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test,best_preds))

```

```
Precision = 0.8571428571428572
Recall = 0.9961977186311788
Accuracy = 0.9925373134328358
f1_score = 0.9961832061068702
Matthews Correlation Coefficient = 0.841934607768814


----------------Classification report--------------------------
              precision    recall  f1-score   support

     Robusta       0.71      1.00      0.83         5
     Arabica       1.00      0.99      1.00       263

    accuracy                           0.99       268
   macro avg       0.86      1.00      0.91       268
weighted avg       0.99      0.99      0.99       268


Confusion Matrix:
[[  5   0]
 [  2 261]]
```

The precision of Random Forest for Robusta is 0.71 which is the same as Logistic Regression. 2 samples of Robusta are incorrectly classified as Arabica.

4. Passive Aggressive Classifier:

```
np.random.seed(10)
model_dec = DecisionTreeClassifier()
model_dec.fit(X_res, y_res)
best_preds = model_dec.predict(X_test)
```

```
#Performance metrics
print("Precision = {}".format(precision_score(y_test, best_preds, average='macro')))
print("Recall = {}".format(recall_score(y_test, best_preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(y_test, best_preds)))
print("f1_score = {}".format(f1_score(y_test, best_preds)))
print("Matthews Correlation Coefficient = {}".format(matthews_corrcoef(y_test, best_preds)))

#print the classfication report
print("\n")
print("----------------Classification report--------------------------")
print(classification_report(y_test,best_preds,target_names=['Robusta','Arabica']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test,best_preds))

```

```
Precision = 0.8125
Recall = 0.9942965779467681
Accuracy = 0.9888059701492538
f1_score = 0.994263862332696
Matthews Correlation Coefficient = 0.7860475319174154


----------------Classification report--------------------------
              precision    recall  f1-score   support

     Robusta       0.62      1.00      0.77         5
     Arabica       1.00      0.99      0.99       263

    accuracy                           0.99       268
   macro avg       0.81      0.99      0.88       268
weighted avg       0.99      0.99      0.99       268


Confusion Matrix:
[[  5   0]
 [  3 260]]
```

The Precision of Passive Aggressive Classifier for Robusta is just 0.62 which is the same as the Decision Tree. 3 samples of Robusta are incorrectly classified as Arabica.

5. XGBoost Classifier:

```
#convert the train and test datasets into DMatrix
D_train = xgb.DMatrix(X_res, label=y_res)
D_test = xgb.DMatrix(X_test, label=y_test)
```

```
param = {
    'eta': 0.3,
    'max_depth': 3,
    'objective': 'multi:softprob',
    'num_class': 2,
    'random_state':4
     }

steps = 20  # The number of training iterations
```

```
#train the model
model = xgb.train(param, D_train, steps)
```

```python
#test the model
preds = model.predict(D_test)
best_preds = np.asarray([np.argmax(line) for line in preds])

#Performance metrics
print("Precision = {}".format(precision_score(y_test, best_preds, average='macro')))
print("Recall = {}".format(recall_score(y_test, best_preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(y_test, best_preds)))
print("f1_score = {}".format(f1_score(y_test, best_preds)))
print("Matthews Correlation Coefficient = {}".format(matthews_corrcoef(y_test, best_preds)))

#print the classfication report
print("\n")
print("----------------Classification report--------------------------")
print(classification_report(y_test,best_preds,target_names=['Robusta','Arabica']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test,best_preds))
```

```
Precision = 0.9
Recall = 0.9981060606060606
Accuracy = 0.996268656716418
f1_score = 0.9981024667931689
Matthews Correlation Coefficient = 0.892731592904439


----------------Classification report--------------------------
              precision    recall  f1-score   support

     Robusta       0.80      1.00      0.89         4
     Arabica       1.00      1.00      1.00       264

    accuracy                           1.00       268
   macro avg       0.90      1.00      0.94       268
weighted avg       1.00      1.00      1.00       268


Confusion Matrix:
[[  4   0]
 [  1 263]]
```

The Precision of XGBoost for the Robusta class is 0.80 which is higher compared to all other algorithms. Only 1 sample of Robusta is incorrectly classified as Arabica. Hence, this model is selected and shown in the notebook and deployed into the server. The Precision, Recall, f1-score for Robusta from the classification report are summarised in the below table. The true performance of a classifier in the case of an imbalanced dataset is measured by How good is the classifier predicting the minority class(Robusta).

Model Name	Precision	Recall	F1-score
Logistic Regression	0.71	1.00	0.83
Decision Tree	0.62	1.00	0.77
Random Forest	0.71	1.00	0.83

Passive Aggressive	0.62	1.00	0.77
XGBoost	0.80	1.00	0.89

Hence, XGBoost is selected as it has high precision and f1-score compared to other classification algorithms.

Clustering algorithms:

This section talks about the various clustering algorithms I applied for clustering our coffee dataset.

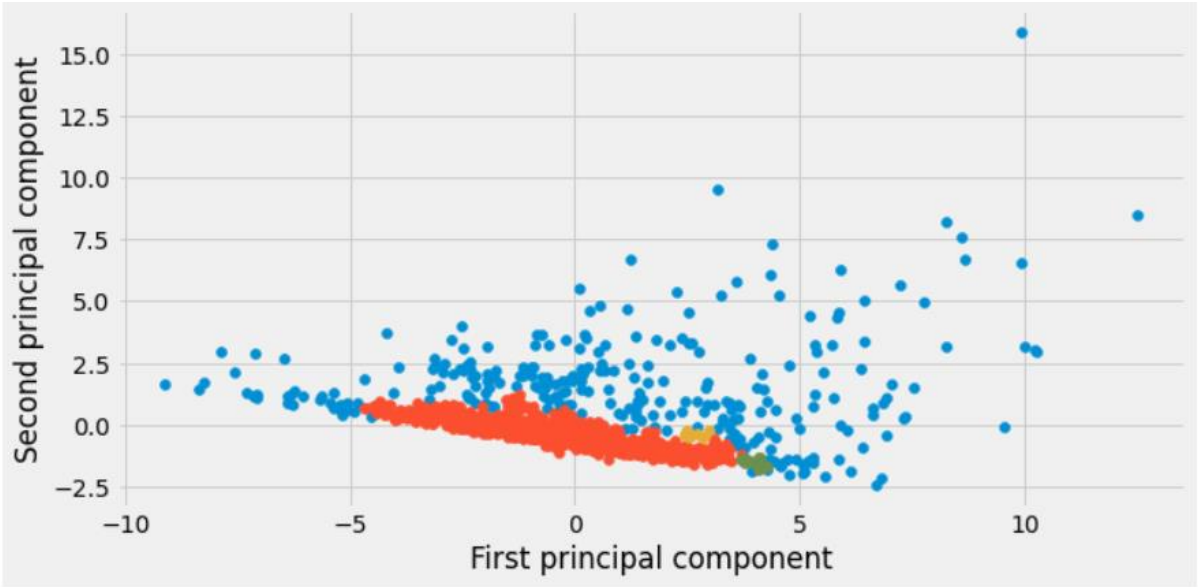
1. DBScan:

Density-based clustering determines cluster assignments based on the density of data points in a region. Clusters are assigned where there are high densities of data points separated by low-density regions.

```
dbscan = DBSCAN(eps=0.3,min_samples=9)
y_dbscan = dbscan.fit_predict(x_pca)
# retrieve unique clusters
clusters = np.unique(y_dbscan)

plt.figure(figsize=(10,5))
# create scatter plot for samples from each cluster
for cluster in clusters:
    # get row indexes for samples with this cluster
    row_ix = np.where(y_dbscan == cluster)
    # create scatter of these samples
    plt.scatter(x_pca[row_ix, 0], x_pca[row_ix, 1])

# show the plot
plt.show()
```




```
dbscan_silhouette = silhouette_score(x_pca, y_dbscan).round(2)
print(dbscan_silhouette)
```

```
0.14
```

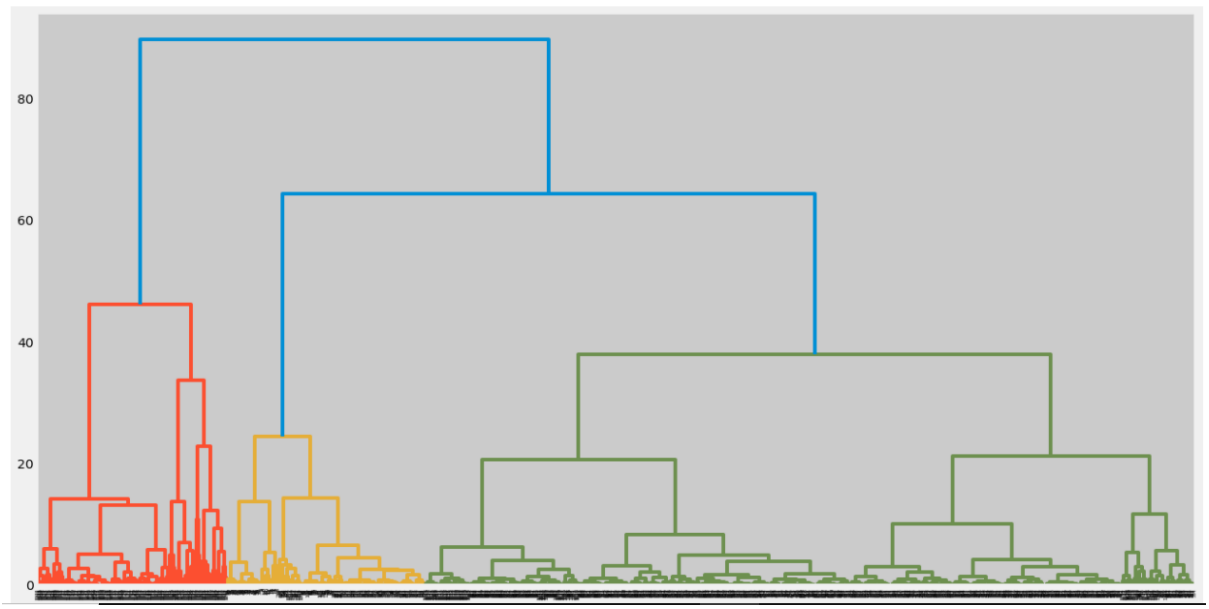
DBScan made 4 clusters out of which 2 have very few data points. The silhouette score for DBScan is 0.14.

2.Agglomerative Clustering:

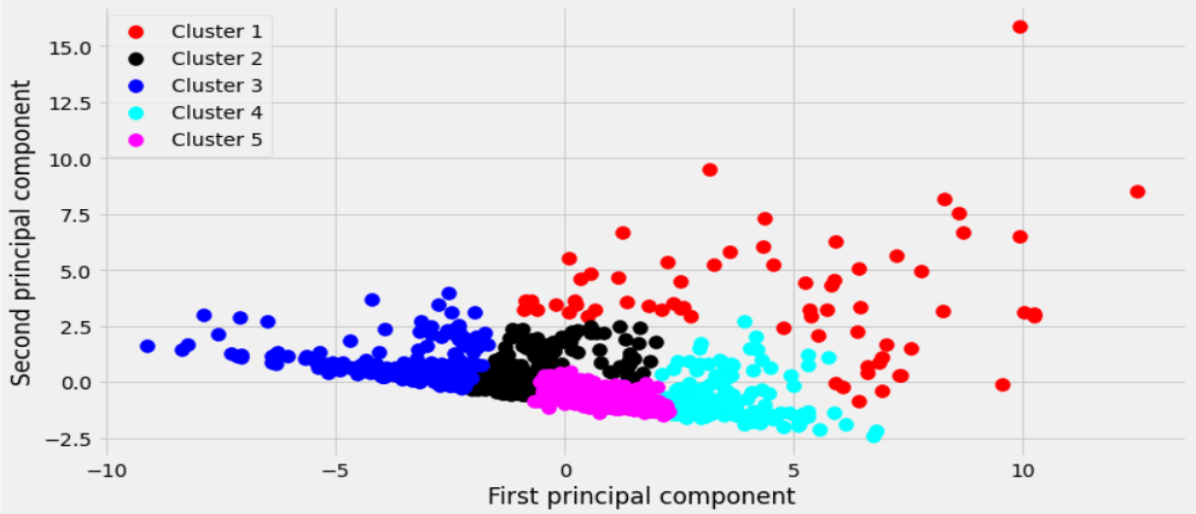
It is a part of Hierarchical clustering. Agglomerative clustering is the bottom-up approach. It merges the two points that are the most similar until all points have been merged into a single cluster.

```
# import hierarchical clustering libraries
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
```

```
# create dendrogram
dendrogram = sch.dendrogram(sch.linkage(x_pca, method='ward'))
# create clusters
hc = AgglomerativeClustering(n_clusters=4, affinity = 'euclidean', linkage = 'ward')
# save clusters for chart
y_hc = hc.fit_predict(x_pca)
```




```
plt.scatter(x_pca[y_hc ==0,0], x_pca[y_hc == 0,1], s=100, c='red',label='Cluster 1')
plt.scatter(x_pca[y_hc==1,0], x_pca[y_hc == 1,1], s=100, c='black',label='Cluster 2')
plt.scatter(x_pca[y_hc ==2,0], x_pca[y_hc == 2,1], s=100, c='blue',label='Cluster 3')
plt.scatter(x_pca[y_hc ==3,0], x_pca[y_hc == 3,1], s=100, c='cyan',label='Cluster 4')
plt.scatter(x_pca[y_hc==4,0],x_pca[y_hc==4,1],s=100,c='magenta',label='Cluster 5')
plt.xlabel('First principal component')
plt.ylabel('Second principal component')
plt.legend()
plt.show()
```



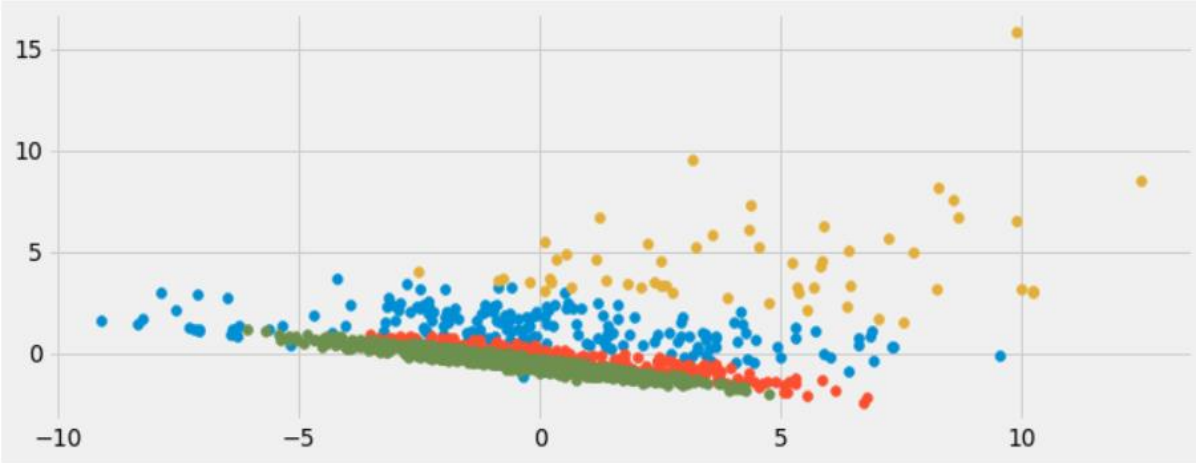
```
In [62]:  1  silhouette_score(x_pca,y_hc)
Out[62]: 0.40383893680363786
```

5 clusters are made with the Agglomerative Clustering model and the silhouette score is 0.40.

3.Gaussian Mixture Model

Gaussian Mixture Models (GMMs) assume that there are a certain number of Gaussian distributions, and each of these distributions represents a cluster. Hence, a Gaussian Mixture Model tends to group the data points belonging to a single distribution together.

```
1  from sklearn.mixture import GaussianMixture
2  plt.figure(figsize=(11,4))
3  model = GaussianMixture(n_components=4)
4  # fit the model
5  model.fit(x_pca)
6  # assign a cluster to each example
7  yhat = model.predict(x_pca)
8  # retrieve unique clusters
9  clusters = np.unique(yhat)
10 # create scatter plot for samples from each cluster
11 for cluster in clusters:
12     # get row indexes for samples with this cluster
13     row_ix = np.where(yhat == cluster)
14     # create scatter of these samples
15     plt.scatter(x_pca[row_ix, 0], x_pca[row_ix, 1])
16 # show the plot
17 plt.xlabel('First principal component')
18 plt.ylabel('Second principal component')
19 plt.show()
```




```
In [94]:  1  silhouette_score(x_pca,yhat)
Out[94]: 0.14217683695098654
```

GMM didn't perform that well with this data. The silhouette score is just 0.14 which is a very poor score.

4. KMeans clustering:

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group.

```
#call KMeans
kmeans = KMeans( n_clusters = 5, init='k-means++',random_state=13)

#Fit the model on our data
kmeans.fit(x_pca)

#get the predictions
y_kmeans = kmeans.predict(x_pca)
```

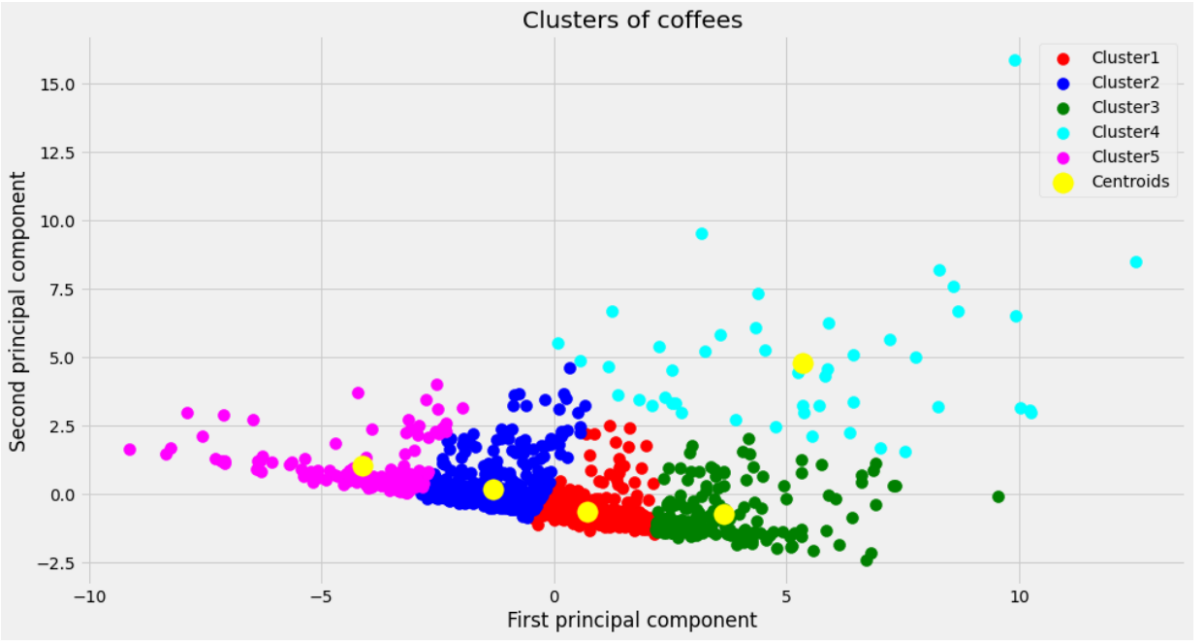
```
#set the figure size
plt.figure(figsize=(15,8))

#plot the scatter plots cluster wise
plt.scatter(x_pca[y_kmeans==0,0],x_pca[y_kmeans==0,1],s=100,c='red',label='Cluster1')
plt.scatter(x_pca[y_kmeans==1,0],x_pca[y_kmeans==1,1],s=100,c='blue',label='Cluster2')
plt.scatter(x_pca[y_kmeans==2,0],x_pca[y_kmeans==2,1],s=100,c='green',label='Cluster3')
plt.scatter(x_pca[y_kmeans==3,0],x_pca[y_kmeans==3,1],s=100,c='cyan',label='Cluster4')
plt.scatter(x_pca[y_kmeans==4,0],x_pca[y_kmeans==4,1],s=100,c='magenta',label='Cluster5')

#plot the centroids
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=300,c='yellow',label='Centroids')

plt.title('Clusters of coffees')
plt.xlabel('First principal component')
plt.ylabel('Second principal component')

#show the plot
plt.legend()
plt.show()
```



In [52]:
```
# Compute the silhouette scores for KMeans algorithm
kmeans_silhouette = silhouette_score(x_pca, kmeans.labels_).round(2)
kmeans_silhouette
```

Out[52]: 0.42

The clusters formed by KMeans are similar to the Agglomerative Clustering model. But, the KMeans algorithm's silhouette score is 0.42 which is the highest compared to all other algorithms.

The Silhouette scores of all algorithms applied are summarised below:

Clustering Model	Silhouette Score
DBScan Clustering	0.14
Agglomerative Clustering	0.40
Gaussian Mixture Model Clustering	0.14
KMeans Clustering	0.42

KMeans clustering model is selected as it has the highest silhouette score and K=5 gave the least SSE.