Assignment 2

K-Means & DBSCAN

COMP257(SEC. 003)

Yoo Sun Song
301091906

1) Retrieve and load the Olivetti faces dataset.

Using the sklearn library to import the Olivetti faces dataset, a collection of face images on a grayscale.

```
From sklearn.datasets import fetch_olivetti_faces
```

Fetches the Olivetti faces dataset, with data shuffled and a random state set for reproducibility.

```
olivetti_faces = fetch_olivetti_faces(shuffle=True, random_state=96)
```

Extracts the images into variable X.

```
X = olivetti_faces.images
```

Extract the labels into variable y.

```
y = olivetti_faces.target
```

Creates a 5x5 grid of subplots to display 25 images.

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(5, 5, figsize=(10, 10))
axes = axes.ravel()

for i in range(25):
    axes[i].imshow(X[i], cmap='gray')
    axes[i].set_title(f"Label: {y[i]}")
    axes[i].axis('off')

plt.tight_layout()
plt.show()
```

The above picture is output.

Label: 21
Label: 27
Label: 11
Label: 0
Label: 37
Label: 19
Label: 8
Label: 12
Label: 13
Label: 1
Label: 0
Label: 1
Label: 19
Label: 34
Label: 28
Label: 17
Label: 32
Label: 22
Label: 3
Label: 17
Label: 28
Label: 25
Label: 16
Label: 37
Label: 18

2) Split the training set, the validation set, and the test set using stratified sampling to ensure the same number of images per person in each set. Provide your rationale for the split ratio.

Split the Olivetti face dataset into training, validation, and test sets using stratified sampling. The goal is to ensure that each set represents each individual's image equally.

Import the function for splitting datasets.

```
from sklearn.model_selection import train_test_split
```

Split the dataset into an 80% temporary set and a 20% test set while preserving the label distribution.

```
train_test_split(X, y, test_size=0.2, stratify=y, random_state=96)
```

Split the temporary set into 60% training and 20% validation sets, again preserving the label distribution.

```
train_test_split(X_temp, y_temp, test_size=0.25, stratify=y_temp, random_state=96)
```

'stratify' is used to ensure that each set has a balanced class, which is important for fair evaluation.

```
print(f"Temporary set: {X_temp.shape}, {y_temp.shape}")
```

- Temporary set: (320, 64, 64), (320,)
- there are 320 samples, each of 64x64 pixels.
- there are 320 labels corresponding to the 320 samples.

```
print(f"Training set: {X_train.shape}, {y_train.shape}")
```

- Training set: (240, 64, 64), (240,)
- there are 240 samplesfor training, each of 64x64 pixels.
- there are 240 labels for the training set.

```
print(f"Validation set: {X_val.shape}, {y_val.shape}")
```

- Validation set: (80, 64, 64), (80,)
- there are 80 samples in validation set, each of 64x64 pixels.
- there are 80 labels for the validation set.

```
print(f"Test set: {X_test.shape}, {y_test.shape}")
```

- Test set: (80, 64, 64), (80,)
- there are 80 samples in the test set, each of 64x64 pixels.
- there are 80 labels for the test set.

3) Using k-fold cross validation, train a classifier to predict which person is represented in each picture, and evaluate it on the validation set.

Reshape the Olivetti faces data and trains the Support Vector Machine (SVM) classifier using k-fold cross-validation. It then evaluates the model on a validation set and plots the cross-validation scores.

Reshape each set of images to a 2D array for compatibility with the classifier.

```
X_train = X_train.reshape((X_train.shape[0], -1))
X_val = X_val.reshape((X_val.shape[0], -1))
X_test = X_test.reshape((X_test.shape[0], -1))
```

Initialize the SVM classifier with a linear kernel and normalization parameter (C) of 1.

```
clf = SVC(kernel='linear', C=1)
```

Performs 5-fold cross-validation on the training set, raising an exception if an error occurs.

```
cv_scores = cross_val_score(clf, X_train, y_train, cv=5, error_score='raise')

print(f"Cross Validation Scores: {cv_scores}")
```

```
Cross Validation Scores: [0.875 0.95833333 0.89583333 0.95833333 0.9375 ]
```

→ This value indicates how well the model performed on each fold of the 5-fold cross-validation. Values are between 0 and 1, and 1 indicates perfect accuracy.

```
print(f"Average 5 Fold CV Score: {np.mean(cv_scores)}")
```

```
Average 5 Fold CV Score: 0.925
```

→ 0.925 is the average of the 5 cross-validation scores.

Fit the model to the complete training set.

```
clf.fit(X_train, y_train)
```

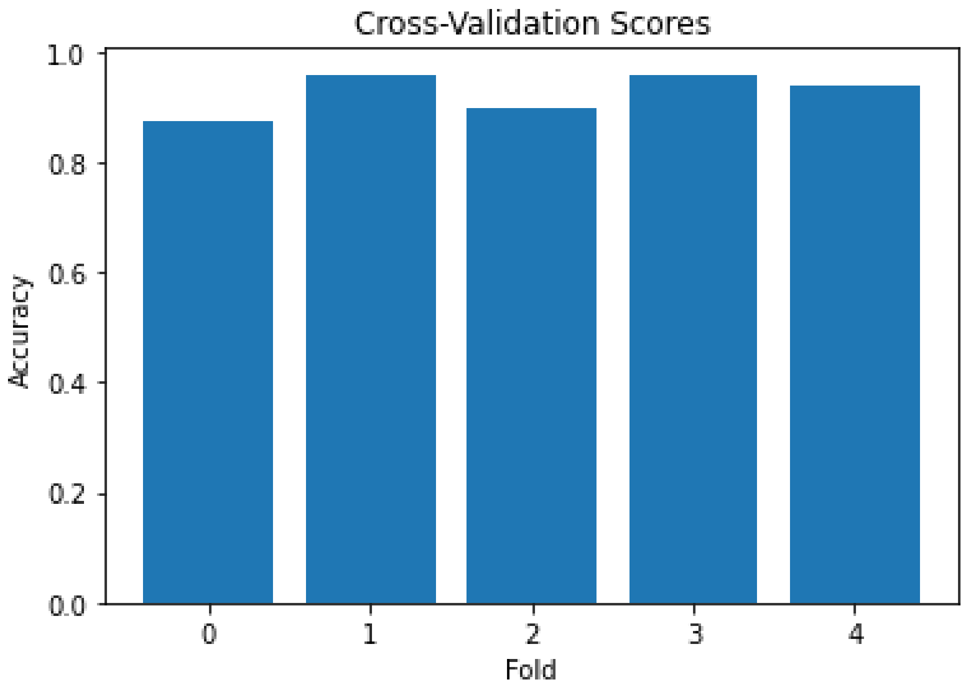
Evaluate the model on the validation set.

```
val_score = clf.score(X_val, y_val)
print(f"Validation Set Score: {val_score}")
```

```
Validation Set Score: 0.9125
```

plot the cross-validation scores .

```
plt.bar(range(len(cv_scores)), cv_scores)
plt.title("Cross-Validation Scores")
plt.xlabel("Fold")
plt.ylabel("Accuracy")
plt.show()
```


Cross-Validation Scores
Accuracy
Fold
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4

4) Use K-Means to reduce the dimensionality of the set. Provide your rationale for the similarity measure used to perform the clustering. Use the silhouette score approach to choose the number of clusters.

The code applies K-Means clustering to reduce the dimensionality of the training, validation, and test sets. It uses silhouette scores to identify the optimal number of clusters.

Define the range of clusters.

```
n_clusters_list = range(2, 15)
```

Initialized to 0, this variable will eventually hold the number of clusters (n_clusters) that gives the highest silhouette score.

```
best_n_clusters = 0
```

Initialized to -1, this variable will store the highest silhouette score achieved during the loop that tests various numbers of clusters.

```
best_silhouette = -1
```

Iteratively perform K-Means clustering on the training set for varying numbers of clusters, calculate the silhouette score for each configuration, and update the best number of clusters and corresponding silhouette score. It also stores all the silhouette scores in a list for further analysis or plotting.

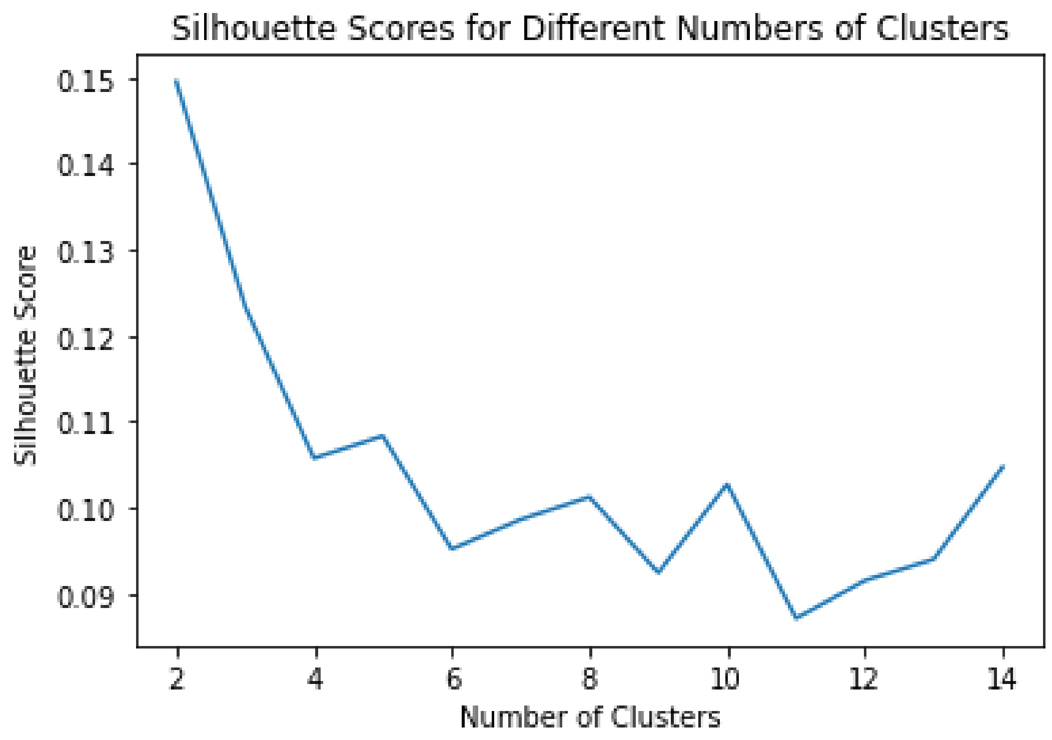
```
silhouette_scores = []
for n_clusters in n_clusters_list:
    kmeans = KMeans(n_clusters=n_clusters, random_state=96)
    cluster_labels = kmeans.fit_predict(X_train)
    silhouette_avg = silhouette_score(X_train, cluster_labels)
    print(f"For n_clusters = {n_clusters}, Silhouette score is {silhouette_avg}")
    silhouette_scores.append(silhouette_avg)
    if silhouette_avg > best_silhouette:
        best_n_clusters = n_clusters
        best_silhouette = silhouette_avg
```

Result:

```
For n_clusters = 2, Silhouette score is 0.14954762160778046
For n_clusters = 3, Silhouette score is 0.12343879044055939
For n_clusters = 4, Silhouette score is 0.1057407557964325
For n_clusters = 5, Silhouette score is 0.10832604765892029
For n_clusters = 6, Silhouette score is 0.0951652005314827
For n_clusters = 7, Silhouette score is 0.09861566871404648
For n_clusters = 8, Silhouette score is 0.1012277603149414
For n_clusters = 9, Silhouette score is 0.09243087470531464
For n_clusters = 10, Silhouette score is 0.10272692143917084
For n_clusters = 11, Silhouette score is 0.08710966259241104
For n_clusters = 12, Silhouette score is 0.09152248501777649
For n_clusters = 13, Silhouette score is 0.09399876743555069
For n_clusters = 14, Silhouette score is 0.10473056137561798
```

Plot the silhouette scores for different numbers of clusters

```
plt.plot(n_clusters_list, silhouette_scores)
plt.title("Silhouette Scores for Different Numbers of Clusters")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.show()
```



Fit the KMeans model with the best number of clusters

```
kmeans = KMeans(n_clusters=best_n_clusters, random_state=96)
kmeans.fit(X_train)
```

Transform the data points to their corresponding cluster centers

```
X_train_reduced = kmeans.transform(X_train)
X_val_reduced = kmeans.transform(X_val)
X_test_reduced = kmeans.transform(X_test)

print(f"Optimal number of clusters: {best_n_clusters}")
```

Optimal number of clusters: 2

5) Use the set from step (4) to train a classifier as in step (3)

The code segment trains a Support Vector Classifier (SVC) on the dimensionality-reduced dataset obtained from K-Means clustering. It also evaluates the model using 5-fold cross-validation and on a reduced validation set.

New SVC (clf_reduced) is initialized with linear kernel and C=1, similar to the original classifier.

```
clf_reduced = SVC(kernel='linear', C=1)
```

5-fold cross-validation is performed on the reduced training set.

```
try:
    cv_scores_reduced = cross_val_score(clf_reduced, X_train_reduced, y_train, cv=5, error_score='raise')
    print(f"Cross Validation Scores on reduced set: {cv_scores_reduced}")
    print(f"Average 5 Fold CV Score on reduced set: {np.mean(cv_scores_reduced)}")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Cross Validation Scores on reduced set: [0.22916667 0.20833333 0.29166667 0.16666667
0.29166667]
Average 5 Fold CV Score on reduced set: 0.2375
```

Fit the model to the complete training set.

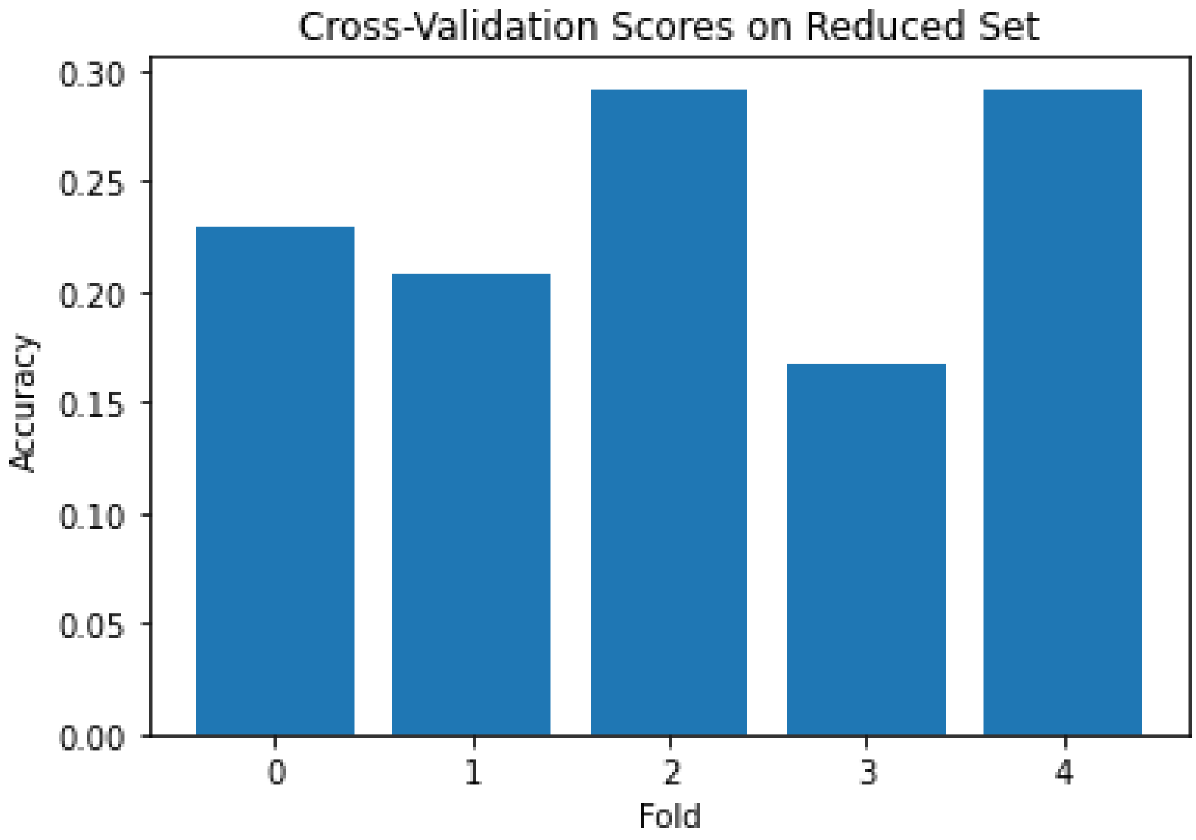
```
clf_reduced.fit(X_train_reduced, y_train)
```

Evaluate the model on the reduced validation set.

```
val_score_reduced = clf_reduced.score(X_val_reduced, y_val)
print(f"Validation Set Score on reduced set: {val_score_reduced}")
```

Plot the cross-validation scores for the reduced dataset.

```
plt.bar(range(len(cv_scores_reduced)), cv_scores_reduced)
plt.title("Cross-Validation Scores on Reduced Set")
plt.xlabel("Fold")
plt.ylabel("Accuracy")
plt.show()
```



6)Apply DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm to the Olivetti Faces dataset for clustering. Preprocess the images and convert them into feature vectors, then use DBSCAN to group similar images together based on their density. Provide your rationale for the similarity measure used to perform the clustering, considering the nature of facial image data.

Scales the features using StandardScaler.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Initializes DBSCAN with eps=0.5 and min_samples=5.

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
```

Fits the model to the scaled training data.

```
dbscan_labels_train = dbscan.fit_predict(X_train_scaled)
```

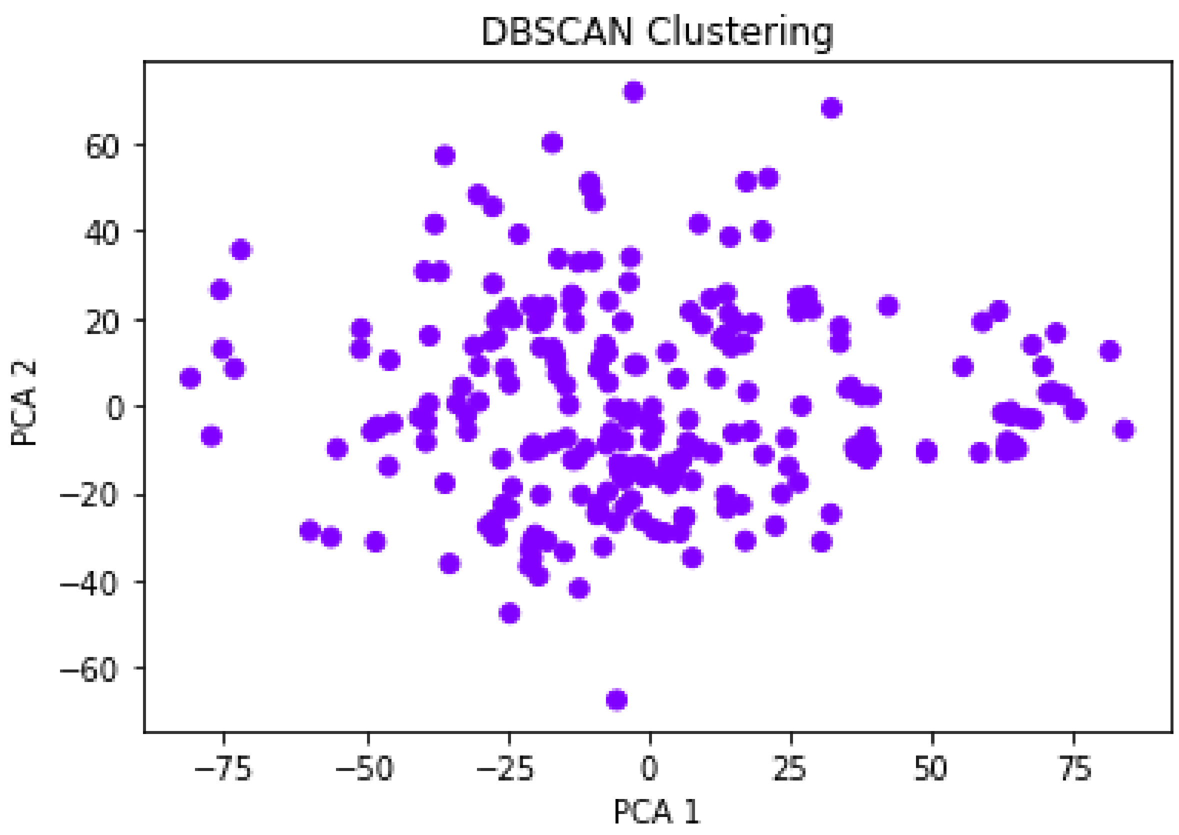
The code identifies the unique cluster labels produced by DBSCAN, calculates the number of clusters, and then computes the silhouette score for these clusters if more than one cluster exists; otherwise, it states that the silhouette score cannot be calculated for a single cluster.

```
unique_labels = np.unique(dbscan_labels_train)
n_clusters = len(unique_labels) - (1 if -1 in unique_labels else 0)
print(f"Number of clusters: {n_clusters}")

if n_clusters > 1:
    silhouette_avg = silhouette_score(X_train_scaled, dbscan_labels_train)
    print(f"Silhouette score is {silhouette_avg}")
else:
    print("Cannot calculate silhouette score for a single cluster.")
```

```
Number of clusters: 0
Cannot calculate silhouette score for a single cluster.
```

A bar chart displays the cross-validation accuracy scores for the 5 folds.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels_train, cmap='rainbow')
plt.title("DBSCAN Clustering")
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.show()
```


DBSCAN Clustering
PCA 2
PCA 1