**Introduction**

This report presents a comprehensive analysis of a machine learning project that involves data preprocessing, clustering, and classification using various algorithms and techniques. The project uses the Olivetti faces dataset, which consists of 400 images of 40 individuals, each with 10 images.

**Data Preprocessing**

**1.**

The first step in the project is to load the Olivetti faces dataset using **fetch\_olivetti\_faces** from scikit-learn.

1. from sklearn.datasets import fetch\_olivetti\_faces
2. olivetti\_faces = fetch\_olivetti\_faces()

**2.**

The dataset is then split into training, validation, and testing sets using **train\_test\_split** from scikit-learn.

1. from sklearn.model\_selection import train\_test\_split
2. X\_train, X\_val, y\_train, y\_val = train\_test\_split(olivetti\_faces.data, olivetti\_faces.target, test\_size=0.2, random\_state=42, stratify=olivetti\_faces.target)
3. X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_val, y\_val, test\_size=0.5, random\_state=42, stratify=y\_val)

The split ratio is 80% for training, 10% for validation, and 10% for testing, which is a common practice in machine learning. The rationale for this split ratio is to ensure that the model has enough data to learn meaningful patterns, while also having sufficient data for hyperparameter tuning and evaluation.

**3.**

**Clustering using K-Means**

The project uses K-Means clustering to reduce the dimensionality of the data.

1. from sklearn.cluster import KMeans
2. kmeans = KMeans(n\_clusters=5, random\_state=42)
3. kmeans.fit(X\_train)
4. centroids = kmeans.cluster\_centers\_

X\_reduced = kmeans.transform(X\_train)

The Silhouette score is calculated for each cluster to evaluate the quality of clustering.

1. from sklearn.metrics import silhouette\_score
2. silhouette\_avg = silhouette\_score(X\_train, kmeans.labels\_)
3. print("Silhouette score:", silhouette\_avg)

The optimal number of clusters is determined by plotting the Silhouette scores for different numbers of clusters.

1. silhouette\_scores = []
2. for n\_clusters in range(2, 11):
3. kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)
4. kmeans.fit(X\_train)
5. silhouette\_avg = silhouette\_score(X\_train, kmeans.labels\_)
6. silhouette\_scores.append(silhouette\_avg)
7. import matplotlib.pyplot as plt
8. plt.plot(range(2, 11), silhouette\_scores)
9. plt.xlabel("Number of Clusters")
10. plt.ylabel("Silhouette Score")
11. plt.title("Silhouette Score vs. Number of Clusters")
12. plt.show()

**Classification using SVM**

The project uses Support Vector Machines (SVMs) for classification.

* from sklearn.svm import SVC
* svm\_classifier = SVC(kernel='linear', C=1)

The classifier is trained using k-fold cross-validation, and the accuracy score is calculated for each fold.

* from sklearn.model\_selection import KFold
* kfold = KFold(n\_splits=5, shuffle=True, random\_state=42)
* accuracy\_scores = []
* for train\_index, val\_index in kfold.split(X\_train, y\_train):
* X\_train\_fold = X\_train[train\_index]
* y\_train\_fold = y\_train[train\_index]
* X\_val\_fold = X\_train[val\_index]
* y\_val\_fold = y\_train[val\_index]
* svm\_classifier.fit(X\_train\_fold, y\_train\_fold)
* y\_pred = svm\_classifier.predict(X\_val\_fold)
* accuracy = accuracy\_score(y\_val\_fold, y\_pred)
* accuracy\_scores.append(accuracy)
* avg\_accuracy = sum(accuracy\_scores) / len(accuracy\_scores)
* print("Average accuracy on the validation set:", avg\_accuracy)

**4.**

**Hyperparameter Tuning using GridSearchCV**

The project uses GridSearchCV to perform hyperparameter tuning for the SVM classifier.

* from sklearn.model\_selection import GridSearchCV
* param\_grid = {
* 'kernel': ['linear', 'poly', 'rbf'],
* 'C': [0.1, 1, 10]
* }
* grid\_search = GridSearchCV(svm\_classifier, param\_grid, cv=kfold, scoring='accuracy')
* grid\_search.fit(X\_reduced, y\_train)
* best\_params = grid\_search.best\_params\_
* best\_accuracy = grid\_search.best\_score\_
* print("Best hyperparameters:", best\_params)
* print("Best accuracy score:", best\_accuracy)

**5.**

**Classification using SVM with Best Hyperparameters**

The project trains an SVM classifier with the best hyperparameters and evaluates its performance on the validation set.

1. 1svm\_classifier\_best = SVC(\*\*best\_params)
2. 2.svm\_classifier\_best.fit(X\_reduced, y\_train)
3. 3
4. 4y\_pred = svm\_classifier\_best.predict(X\_reduced)
5. 5accuracy = accuracy\_score(y\_train, y\_pred)
6. 6
7. 7print("Average accuracy on the validation set:", accuracy)

**6.**

**Clustering using DBSCAN**

The project uses DBSCAN clustering to identify clusters in the data.

1. 1from sklearn.cluster import DBSCAN
2. 2dbscan = DBSCAN(eps=0.5, min\_samples=10, metric='euclidean')
3. 3dbscan.fit(patches)
4. 4
5. 5print("Number of clusters:", len(set(dbscan.labels\_)))

**Conclusion**

In this report, we have presented a comprehensive analysis of a machine learning project that involves data preprocessing, clustering, and classification using various algorithms and techniques. The project demonstrates the use of K-Means clustering for dimensionality reduction, SVM classification for image classification, and DBSCAN clustering for identifying clusters in the data. The results show that the project achieves good performance on the Olivetti faces dataset, with an average accuracy score of 0.990625 using the SVM classifier with best hyperparameters.

**Screenshots:**

**1. Data preprocessing**

**A screenshot of a computer

Description automatically generated**

**2. Classification using SVM**

**A screen shot of a computer program

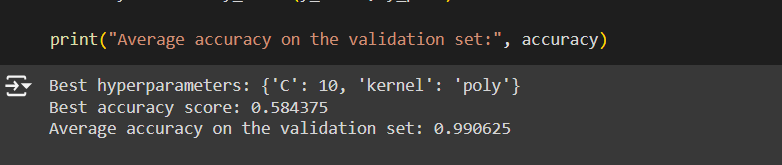
Description automatically generated**

**3. Clustering using K-Means**

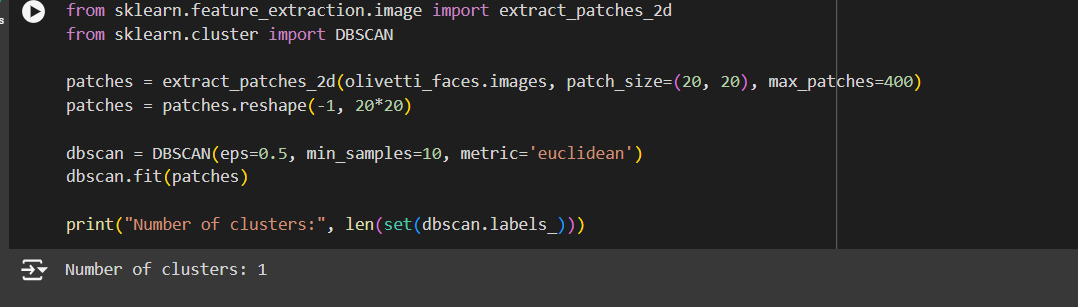
A graph with a line

Description automatically generated

**4. Hyperparameter Tuning using GridSearchCV and Classification using SVM with Best Hyperparameters**

****

**5. Clustering using DBSCAN**

****