Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)

UCS2612 Machine Learning Laboratory

Assignment 9

Name: V.S. Pranav **Reg No:** 3122 21 5001 069

Applications of dimensionality reduction techniques

Github Link: https://github.com/vspr14/ml-lab-assn9

1. Develop a python program to perform dimensionality reduction using PCA and LDA. Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library.

Code:

```
## Importing the dataset
# Step 1: Importing and combining both datasets

# Import necessary libraries
import pandas as pd
import numpy as np
# Read the red wine dataset
red_wine_data = pd.read_csv("winequality-red.csv", sep=";")
red_wine_data['type'] = 1  # Add a column 'type' with value 1 for red wines

# Read the white wine dataset
white_wine_data = pd.read_csv("winequality-white.csv", sep=";")
white_wine_data['type'] = 0  # Add a column 'type' with value 0 for white wines

# Combine the datasets
wine_data_combined = pd.concat([red_wine_data, white_wine_data], ignore_index=True)

# Display the first few rows of the combined dataset
print("Combined Wine Dataset:")
print(wine_data_combined.head())

## Pre-processing
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

# Get all columns except 'type' as X
X = wine_data_combined.drop(columns=['type'])

# Encode non-numeric data into numeric
X_encoded = pd.get_dummies(X)

# Handle missing values by replacing them with the mean
imputer = SimpleImputer(strategy='mean')
```

```
X_imputed = imputer.fit_transform(X_encoded)

# Perform normalization
scaler_norm = MinMaxScaler()
X_normalized = scaler_norm.fit_transform(X_imputed)

# Perform standardization
scaler_std = StandardScaler()
X_final = scaler_std.fit_transform(X_normalized)

# Convert the pre-processed data back to a DataFrame
X_final_df = pd.DataFrame(X_final, columns=X_encoded.columns)

# Display the pre-processed data
print("Pre-processed Data:")
print(X_final_df.head())  # Display first 5 rows

from scipy import stats
# Calculate z-scores for each column in X_final_df
threshold=2.0
z_scores = np.abs(stats.zscore(X_final_df))

outlier_indices = np.any(z_scores > threshold, axis=1)

X_final_df = X_final_df[~outlier_indices]
wine_data_combined = wine_data_combined[~outlier_indices]

print("Shape of X_final_df:", X_final_df.shape)
print("Shape of wine_data_combined:", wine_data_combined.shape)

import matplotlib.pyplot as plt
import seaborn as sns

colors = ['lightblue', 'red']

plt.figure(figsize=(6, 6))
wine_data_combined['type'].value_counts().plot(kind='pie', autopct='%1.1f%%',
colors=colors)
plt.title('Proportion of White and Red Wines')
plt.xlabel('')
plt.ylabel('')
plt.legend(labels=['White wine', 'Red Wine'], loc='upper right')  # Add legend
plt.show()

plt.figure(figsize=(10, 8))
sns.heatmap(X_final_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()

plt.figure(figsize=(8, 6))
sns.histplot(data=wine_data_combined, x='quality', hue='type', kde=True, bins=20,
palette=colors)
plt.title('Quality Distribution for Red and White Wines')
```

```
plt.xlabel('Quality')
plt.ylabel('Count')
plt.legend(title='Wine Type', labels=['Red Wine', 'White Wine'])
plt.show()

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

num_classes_minus_one = len(wine_data_combined['type'].unique()) - 1

pca = PCA(n_components=num_classes_minus_one)
X_pca = pca.fit_transform(X_final_df)

lda = LDA(n_components=num_classes_minus_one)
X_lda = lda.fit_transform(X_final_df, wine_data_combined['type'])

print("PCA Transformed Data Shape:", X_pca.shape)
print("LDA Transformed Data Shape:", X_lda.shape)

from sklearn.model_selection import train_test_split

X_pca_train, X_pca_test, y_train, y_test = train_test_split(X_pca,
wine_data_combined['type'], test_size=0.3, random_state=42)

X_lda_train, X_lda_test, y_train, y_test = train_test_split(X_lda,
wine_data_combined['type'], test_size=0.3, random_state=42)

print("PCA Transformed Data - Training set shape:", X_pca_train.shape)
print("PCA Transformed Data - Testing set shape:", X_pca_test.shape)
print("LDA Transformed Data - Training set shape:", X_lda_train.shape)
print("LDA Transformed Data - Testing set shape:", X_lda_test.shape)

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logreg_pca = LogisticRegression()
logreg_pca.fit(X_pca_train, y_train)

logreg_lda = LogisticRegression()
logreg_lda.fit(X_lda_train, y_train)

y_pred_pca = logreg_pca.predict(X_pca_test)
y_pred_lda = logreg_lda.predict(X_lda_test)

accuracy_pca = accuracy_score(y_test, y_pred_pca)
accuracy_lda = accuracy_score(y_test, y_pred_lda)

print("Accuracy using PCA transformed features:", accuracy_pca)
print("Accuracy using LDA transformed features:", accuracy_lda)

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(4, 3))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title(title)
    plt.show()
plot_confusion_matrix(y_test, y_pred_pca, title='Confusion Matrix - PCA Transformed
Features')
plot_confusion_matrix(y_test, y_pred_lda, title='Confusion Matrix - LDA Transformed
Features')

from sklearn.metrics import classification_report
print("Classification Report - PCA Transformed Features:")
print(classification_report(y_test, y_pred_pca))
print("Classification Report - LDA Transformed Features:")
print(classification_report(y_test, y_pred_lda))

from sklearn.metrics import roc_curve, auc

y_proba_pca_train = logreg_pca.predict_proba(X_pca_train)[:, 1]
y_proba_lda_train = logreg_lda.predict_proba(X_lda_train)[:, 1]
y_proba_pca_test = logreg_pca.predict_proba(X_pca_test)[:, 1]
y_proba_lda_test = logreg_lda.predict_proba(X_lda_test)[:, 1]

fpr_pca_train, tpr_pca_train, _ = roc_curve(y_train, y_proba_pca_train)
fpr_lda_train, tpr_lda_train, _ = roc_curve(y_train, y_proba_lda_train)
fpr_pca_test, tpr_pca_test, _ = roc_curve(y_test, y_proba_pca_test)
fpr_lda_test, tpr_lda_test, _ = roc_curve(y_test, y_proba_lda_test)

roc_auc_pca_train = auc(fpr_pca_train, tpr_pca_train)
roc_auc_lda_train = auc(fpr_lda_train, tpr_lda_train)
roc_auc_pca_test = auc(fpr_pca_test, tpr_pca_test)
roc_auc_lda_test = auc(fpr_lda_test, tpr_lda_test)

plt.figure(figsize=(8, 6))
plt.plot(fpr_pca_train, tpr_pca_train, color='blue', lw=2, label='ROC Curve - PCA Train
(AUC = %0.5f)' % roc_auc_pca_train)
plt.plot(fpr_lda_train, tpr_lda_train, color='red', lw=2, label='ROC Curve - LDA Train
(AUC = %0.5f)' % roc_auc_lda_train)
plt.plot(fpr_pca_test, tpr_pca_test, color='green', lw=2, label='ROC Curve - PCA Test
(AUC = %0.5f)' % roc_auc_pca_test)
plt.plot(fpr_lda_test, tpr_lda_test, color='orange', lw=2, label='ROC Curve - LDA Test
(AUC = %0.5f)' % roc_auc_lda_test)
plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

Sample Screenshots: