Exercise-9
02-05-2024

M. Rohith
3122 21 5001 085

UCS2612 MACHINE LEARNING LABORATORY

ASSIGNMENT-9

Program Code:

```
# -*- coding: utf-8 -*-
"""MRohith_Ex-9.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1xRVCX5fPYzamlOs-
Inip4Ef7ik3r9LOC

## 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"""
```

M. Rohith
3122 21 5001 085

```
# Step 2: Pre-processing

# Import necessary libraries
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

## Outlier Detection and removal
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))

# Find rows where any z-score is greater than the threshold
outlier_indices = np.any(z_scores > threshold, axis=1)

# Remove outliers from X_final_df and wine_data_combined
```

M. Rohith
3122 21 5001 085

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

# Print the shapes of the cleaned data
print("Shape of X_final_df:", X_final_df.shape)
print("Shape of wine_data_combined:", wine_data_combined.shape)

"""## Exploratory Data Analysis"""

# Step 3: Exploratory Data Analysis

# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Define colors for red and white wines
colors = ['lightblue', 'red']

# Visualization 1: Pie chart for the proportion of red and white
wines
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()

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

# Visualization 3: Quality distribution for red and white wines
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()

"""## Feature Engineering"""

# Step 4: Feature Engineering

# Import necessary libraries
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
as LDA

# Determine the number of unique classes minus one
num_classes_minus_one = len(wine_data_combined['type'].unique()) - 1

# Apply PCA with the determined number of components
pca = PCA(n_components=num_classes_minus_one)
X_pca = pca.fit_transform(X_final_df)

# Apply LDA with the determined number of components
lda = LDA(n_components=num_classes_minus_one)
X_lda = lda.fit_transform(X_final_df, wine_data_combined['type'])

# Display the shape of the transformed data
print("PCA Transformed Data Shape:", X_pca.shape)
print("LDA Transformed Data Shape:", X_lda.shape)

"""## Train-Test Split"""

# Step 5: Split the data into training and testing sets

# Import necessary library
from sklearn.model_selection import train_test_split

# Split the PCA transformed data into training and testing sets
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)

# Split the LDA transformed data into training and testing sets
```

M. Rohith
3122 21 5001 085

```
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)

# Display the shapes of the training and testing sets
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)

"""## Training"""

# Step 6: Train the Logistic Regression model

# Import necessary library
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Train Logistic Regression model using PCA transformed features
logreg_pca = LogisticRegression()
logreg_pca.fit(X_pca_train, y_train)

# Train Logistic Regression model using LDA transformed features
logreg_lda = LogisticRegression()
logreg_lda.fit(X_lda_train, y_train)

"""## Testing"""

# Step 7: Test the model
# Predict on the testing set
y_pred_pca = logreg_pca.predict(X_pca_test)
y_pred_lda = logreg_lda.predict(X_lda_test)

"""## Performance Evaluation"""

# Step 8: Measure the performance of the trained model
# Calculate accuracy
accuracy_pca = accuracy_score(y_test, y_pred_pca)
accuracy_lda = accuracy_score(y_test, y_pred_lda)

# Display accuracy
```

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

# Import necessary library
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Function to plot confusion matrix
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 for PCA model
plot_confusion_matrix(y_test, y_pred_pca, title='Confusion Matrix -
PCA Transformed Features')

# Plot confusion matrix for LDA model
plot_confusion_matrix(y_test, y_pred_lda, title='Confusion Matrix -
LDA Transformed Features')

from sklearn.metrics import classification_report

# Calculate classification report for PCA model
print("Classification Report - PCA Transformed Features:")
print(classification_report(y_test, y_pred_pca))

# Calculate classification report for LDA model
print("Classification Report - LDA Transformed Features:")
print(classification_report(y_test, y_pred_lda))

"""## ROC-AUC Curve"""

from sklearn.metrics import roc_curve, auc

# Compute probabilities for PCA and LDA models
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]

# Compute ROC curves and AUC scores for training and test sets
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)

# Plot ROC curves for training and test sets
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()

"""From the analysis of the classification reports and ROC AUC
scores, it is clear that the management of outliers significantly
enhanced the models' effectiveness:

PCA Transformed Features:
```

```
The precision for class 0 is acceptable at 82%, but it is notably
low for class 1 at 0%, indicating frequent incorrect predictions for
class 1.
The recall for class 1 is also very low at 0%, suggesting the model
fails to identify most instances of class 1.
The F1-score for class 1 stands at 0, reflecting inadequate
performance in predicting class 1.
The ROC AUC scores for both the training and testing sets are
relatively modest (approximately 0.59), pointing to the model's
limited ability to distinguish between classes.
LDA Transformed Features:



The precision, recall, and F1-scores for both classes are
outstanding, demonstrating high predictive accuracy.
The ROC AUC scores are exceptionally high (close to 1.0), indicating
superior ability to discriminate between classes.
Outlier detection appears to be a key factor in the varying
performances between the PCA and LDA models.
By eliminating outliers, the LDA model significantly outperformed
the PCA model, particularly in terms of precision, recall, and
overall accuracy.
Outliers tend to disproportionately impact PCA by influencing the
principal components, leading to less effective classification
outcomes. Conversely, LDA is less affected by outliers and focuses
on maximizing the separation between classes, thus enhancing
classification results.

Key Takeaways:

Gained insights into dimensionality reduction techniques such as PCA
and LDA.
Developed skills in data preprocessing and exploratory data
analysis.
Learned to apply PCA and LDA for feature engineering and reducing
dimensionality.
Assessed model performance using reduced feature sets.
Enhanced practical knowledge and critical analysis capabilities in
data science.
"""
```

Exercise-9 M. Rohith
02-05-2024 3122 21 5001 085

Github Link:

Machine-Learning-Lab/Exercise-9 at main · rohith18111407/Machine-Learning-Lab (github.com)

Colab Link:

https://colab.research.google.com/drive/1xRVCX5fPYzamlOs-Inip4Ef7ik3r9LOC?usp=sharing