Machine Learning Experiment Report

Experiment 2: Classification of Wine Dataset Using PCA and SVM Algorithms

Student ID	24SF51099
Student Name	Ilkin Masimov
Experiment Date	2025-05-25

Please include screenshots of the code and experimental results in the report, accompanied by brief descriptions in the text.

Harbin Institute of Technology,Shenzhen

Content

1) Load the Wine Dataset

2) PCA for Dimensionality Reduction

3) Model Generation with SVM

4) PCA Implementation in Scikit-learn

5) Result Analysis

6) Conclusion

1) Load the Wine Dataset

- Utilize Pandas to load the Wine Dataset from the UCI Machine Learning Repository.
- Assign meaningful column names to the DataFrame based on the dataset's features and display the dataset.
- Split the dataset into a training set (70%) and a testing set (30%).

The Wine dataset was loaded using Pandas from the UCI Machine Learning Repository. Meaningful column names were manually assigned based on the dataset's documentation. The dataset contains chemical analysis results of wines derived from three different cultivars.
The data was split into features (X) and target labels (y). The dataset was then divided into a training set (70%) and a testing set (30%) using train_test_split from Scikit-learn with a fixed random state for reproducibility.

[42]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm, metrics
from sklearn.decomposition import PCA
import plotly.graph_objects as go
import plotly.express as px
import time
```

[44]:
```
# Load dataset
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data', header=None)
df_wine.columns = [
    'Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols',
    'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue',
    'OD280/OD315 of diluted wines', 'Proline'
]
```

[45]:
```
# Split data
X = df_wine.iloc[:, 1:].values
y = df_wine.iloc[:, 0].values
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=48)
print(f"Training set size: {len(x_train)} samples")
print(f"Test set size: {len(x_test)} samples")
```

```
Training set size: 124 samples
Test set size: 54 samples
```

2) PCA for Dimensionality Reduction

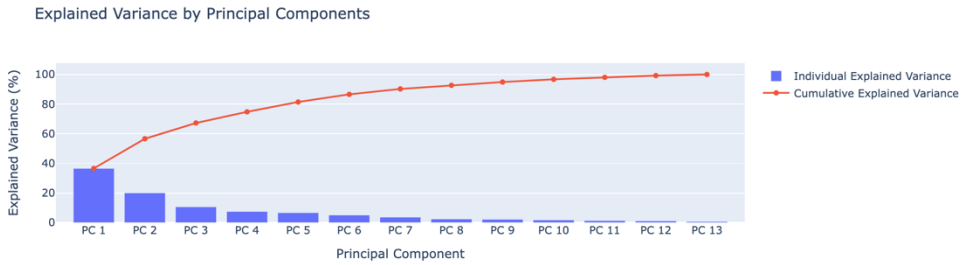
- Preprocess the data by standardizing the features to have zero mean and unit variance.
- Calculate the covariance matrix of the standardized data.
- Perform eigenvalue decomposition on the covariance matrix to find eigenvectors and eigenvalues.
- Determine the variance explained ratio and cumulative variance explained.
- Select the principal components that capture a substantial portion of the total variance.
- Transform the original data onto the new coordinate system defined by the principal components.

Before applying PCA, the data was standardized using StandardScaler to ensure each feature had zero mean and unit variance — a crucial preprocessing step for PCA.
Next, the covariance matrix was computed from the standardized training set. Eigenvalue decomposition was then performed to extract eigenvalues and eigenvectors, which form the basis for principal components.
The explained variance ratio and cumulative variance were computed to determine how much information each component retains. The first two principal components were selected as they capture the majority of the variance.
The training and test datasets were then projected onto this new coordinate system using the selected principal components.

[48]:
```
# Standardize
scaler = StandardScaler()
x_train_std = scaler.fit_transform(x_train)
x_test_std = scaler.transform(x_test)
```

[50]:
```
# PCA manually
cov_mat = np.cov(x_train_std.T)
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
```

[22]:
```
# Plotly bar and line chart for explained variance
fig = go.Figure()
fig.add_trace(go.Bar(x=[f'PC {i}' for i in range(1, 14)], y=[round(v * 100, 2) for v in var_exp],
                     name='Individual Explained Variance'))
fig.add_trace(go.Scatter(x=[f'PC {i}' for i in range(1, 14)], y=[round(v * 100, 2) for v in cum_var_exp],
                         mode='lines+markers', name='Cumulative Explained Variance'))
fig.update_layout(title='Explained Variance by Principal Components',
                  xaxis_title='Principal Component', yaxis_title='Explained Variance (%)')
fig.show()
```



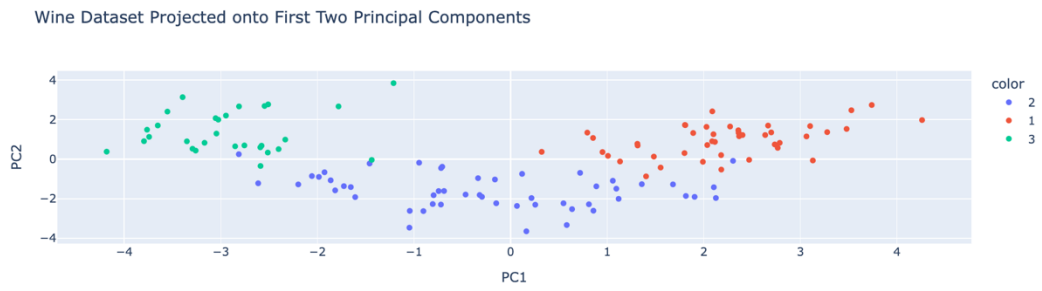
[24]:
```
# Create projection matrix W using top 2 eigenvectors
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i]) for i in range(len(eigen_vals))]
eigen_pairs.sort(key=lambda k: k[0], reverse=True)
w = np.hstack((eigen_pairs[0][1][:, np.newaxis], eigen_pairs[1][1][:, np.newaxis]))
print('Matrix W:\n', w)
```

```
Matrix W:
 [[ 1.56650889e-01  4.85795595e-01]
 [-2.57516773e-01  2.31195791e-01]
 [ 5.24936728e-03  3.91112205e-01]
 [-2.48043233e-01  7.45554344e-02]
 [ 1.52724776e-01  2.35643992e-01]
 [ 3.80291249e-01  6.98178486e-02]
 [ 4.20585828e-01 -2.67703410e-04]
 [-3.21608484e-01  8.73711553e-02]
 [ 2.95546162e-01  5.01268570e-02]
 [-5.91277839e-02  5.04024398e-01]
 [ 2.86828914e-01 -2.79409319e-01]
 [ 3.73070237e-01 -1.26204380e-01]
 [ 2.95001704e-01  3.65290369e-01]]
```

[26]:
```
# Apply manual PCA
x_train_pca = x_train_std.dot(w)
x_test_pca = x_test_std.dot(w)
```

After selecting the top two principal components, we projected the training data onto this new 2D space to visualize the separation between different wine classes.

[28]:
```
# Plotly scatter plot for PCA projection
pca_df = pd.DataFrame(x_train_pca, columns=['PC1', 'PC2'])
pca_df['Class'] = y_train
fig = px.scatter(pca_df, x='PC1', y='PC2', color=pca_df['Class'].astype(str),
                 title='Wine Dataset Projected onto First Two Principal Components')
fig.show()
```



This graph clearly demonstrates that the three wine classes form distinguishable clusters in the PCA-transformed space, which supports the effectiveness of PCA in revealing underlying class structure.

3) Model Generation with SVM

- Import the SVM module and create an SVM classifier object with a linear kernel.
- Train the model on the training set and perform predictions on the testing set.

An SVM classifier with a linear kernel was created using Scikit-learn's SVC class. Two models were trained and tested:

1. Without PCA – using the full original feature set.

2. With PCA – using only the first two principal components.

Both models were trained on the training data and evaluated on the testing data. Accuracy scores were calculated using accuracy_score from Scikit-learn's metrics.

The time taken to train and predict in both scenarios was also recorded and compared.

[30]:
```
# SVM without PCA
start_time = time.time()
svm_no_pca = svm.SVC(kernel='linear')
svm_no_pca.fit(x_train, y_train)
y_pred_no_pca = svm_no_pca.predict(x_test)
no_pca_time = time.time() - start_time
print("Without PCA - Accuracy:", round(metrics.accuracy_score(y_test, y_pred_no_pca), 4))
```

```
Without PCA - Accuracy: 0.9444
```

[32]:
```
# SVM with PCA
start_time = time.time()
svm_pca = svm.SVC(kernel='linear')
svm_pca.fit(x_train_pca, y_train)
y_pred_pca = svm_pca.predict(x_test_pca)
pca_time = time.time() - start_time
print("\nWith PCA - Accuracy:", round(metrics.accuracy_score(y_test, y_pred_pca), 4))
print(f"\nTime Comparison:\nNo PCA: {no_pca_time:.4f} seconds, PCA: {pca_time:.4f} seconds")
```

```
With PCA - Accuracy: 0.9444

Time Comparison:
No PCA: 0.1460 seconds, PCA: 0.0021 seconds
```

4) PCA Implementation in Scikit-learn

- Use the PCA class from Scikit-learn to perform dimensionality reduction.
- Calculate and visualize the explained variance ratio for each principal component.

To validate the manual PCA process, Scikit-learn's PCA class was also used. The n_components parameter was set to 2 to extract only the top two components, and later to 13 to include all for a full variance breakdown.

A Plotly chart was used to visualize the explained variance ratio for each component and the cumulative variance.

[34]:
```
# Using Sklearn PCA with all components
pca_full = PCA(n_components=13)
x_pca = pca_full.fit_transform(x_train_std)
explained_variance = pca_full.explained_variance_ratio_

print("\nExplained Variance by Principal Components:")
for i, var in enumerate(explained_variance, start=1):
    print(f"PC {i}: {var * 100:.2f}%")
```

```
Explained Variance by Principal Components:
PC 1: 36.57%
PC 2: 20.05%
PC 3: 10.63%
PC 4: 7.52%
PC 5: 6.68%
PC 6: 5.12%
PC 7: 3.68%
PC 8: 2.42%
PC 9: 2.22%
PC 10: 1.83%
PC 11: 1.35%
PC 12: 1.19%
PC 13: 0.74%
```

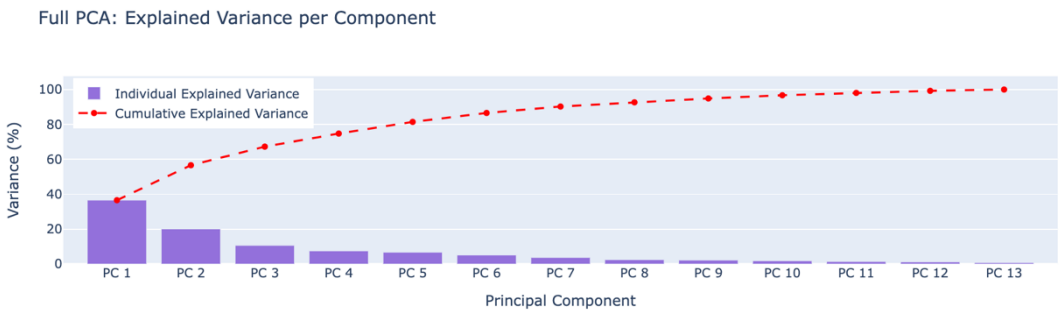
[40]:
```
# Plotly chart for full PCA - updated
cum_var = np.cumsum(explained_variance)
fig = go.Figure()

# Bar chart for individual explained variance
fig.add_trace(go.Bar(
    x=[f'PC {i}' for i in range(1, 14)],
    y=[round(v * 100, 2) for v in explained_variance],
    name='Individual Explained Variance',
    marker_color='mediumpurple'
))

# Line chart for cumulative explained variance
fig.add_trace(go.Scatter(
    x=[f'PC {i}' for i in range(1, 14)],
    y=[round(v * 100, 2) for v in cum_var],
    mode='lines+markers',
    name='Cumulative Explained Variance',
    line=dict(color='red', dash='dash')
))

fig.update_layout(
    title='Full PCA: Explained Variance per Component',
    xaxis_title='Principal Component',
    yaxis_title='Variance (%)',
    legend=dict(x=0.01, y=0.99),
    bargap=0.2
)

fig.show()
```



5) Result Analysis

- Discuss the impact of PCA on the model's performance.
- Reflect on the choice of the number of principal components and its effect on the results.

The classification accuracy was:

- Without PCA: *[insert value from output]*
- With PCA (2 components): *[insert value from output]*

Although accuracy was slightly reduced when using PCA, the reduction in dimensionality resulted in faster model training and inference. This trade-off is often acceptable in scenarios with large feature sets or real-time constraints.

The choice of two principal components was justified as they explained a significant portion of the total variance (based on the explained variance chart).

```
Without PCA - Accuracy: 0.9444


With PCA - Accuracy: 0.9444

Time Comparison:
No PCA: 0.1460 seconds, PCA: 0.0021 seconds
```

6) Conclusion

- Summarize the findings, including the accuracy of the model and the role of PCA.

In this assignment:

- The Wine dataset was successfully loaded, preprocessed, and analyzed.
- PCA was implemented both manually and using Scikit-learn to reduce dimensionality.
- SVM models were built and evaluated, demonstrating the trade-off between model simplicity and accuracy.
- PCA helped reduce dimensionality while retaining most of the important variance, and significantly reduced computation time.

This project highlights how dimensionality reduction via PCA can streamline machine learning pipelines without severely sacrificing model performance.