REPORT ON PROJECT

DATA MINING

JANUARY 24, 2024

Hamidy Adeebullah (539745)

Content

Importing the necessary libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import sys
```

pandas (pd):

Role: Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrame, which is useful for handling structured data, such as CSV files.

Usage in my project : Used for reading and manipulating the dataset (pd.read_csv, df.head(), df.info(), df.describe()).

matplotlib.pyplot (plt):

Role: Matplotlib is a widely used library for creating visualizations in Python. The pyplot module provides a convenient interface for creating various types of plots and charts.

Usage in my project: Used for plotting histograms and scatter plots to visualize data distributions and clusters.

sklearn.model_selection:

Role: The model_selection module in scikit-learn provides tools for model selection and evaluation, including functions for splitting datasets into train and test sets, cross-validation, and stratified k-fold splitting.

Usage in my project: Used for splitting the dataset into training and testing sets (train_test_split) and performing cross-validation (cross_val_score, StratifiedKFold).

sklearn.preprocessing:

Role: The preprocessing module in scikit-learn provides various functions for preprocessing data, including scaling, normalization, and feature selection.

Usage in my project: Used for standardizing features (StandardScaler).

sklearn.feature_selection:

Role: The feature_selection module in scikit-learn provides methods for selecting relevant features in a dataset. It helps in choosing the most informative features for model training.

Usage in my project: Used for feature selection (SelectKBest, f_classif).

sklearn.decomposition:

Role: The decomposition module in scikit-learn provides methods for dimensionality reduction. PCA (Principal Component Analysis) is a common technique for reducing the number of features while retaining most of the information.

Usage in my project: Used for dimensionality reduction (PCA).

sklearn.cluster:

Role: The cluster module in scikit-learn provides clustering algorithms. In my code, KMeans clustering is used to cluster data points based on their features.

Usage in my project: Used for clustering (KMeans).

sklearn.ensemble:

Role: The ensemble module in scikit-learn provides ensemble methods, including the Random Forest classifier. Ensemble methods combine multiple models to improve predictive performance and control overfitting.

Usage in my project: Used for classification using Random Forest (RandomForestClassifier).

sklearn.svm:

Role: The svm module in scikit-learn provides support vector machine algorithms. SVM is a powerful algorithm used for classification and regression tasks.

Usage in my project: Used for classification using Support Vector Classifier (SVC).

sys:

Role: The sys module provides access to some variables used or maintained by the Python interpreter and functions that interact with the interpreter. In my code, it is used to read the file path from the command line arguments.

Usage in my project: Used for reading the file path from command line arguments (sys.argv).

These libraries collectively provide a comprehensive set of tools for data manipulation, exploration, visualization, and machine learning tasks in your project. They help streamline the workflow and make it easier to perform various operations on your dataset.

Cleaning the Data:

```
print("Missing Values:\n", df.isnull().sum())
```

This code prints the number of missing values for each column in the dataset. If the output indicates that there are missing values, additional steps would be needed to handle them appropriately. For example, we might consider using the fillna method to replace missing values or the dropna method to remove rows or columns with missing values.

Data Exploration:

```
print(df.head())  # Display a few rows of the dataset
print(df.info())
print(df.describe())
```

In This Code:

Displaying the First Few Rows of the Dataset:

The first line of the code prints the first few rows of the dataset, allowing you to see the structure and content of the data.

Displaying Information About the Dataset:

The second line of the code prints information about the dataset, including the data types of each column and the number of non-null values.

Displaying Descriptive Statistics:

This code prints descriptive statistics (e.g., mean, min, max) for each numerical column in the dataset.

Data Visualization:

```
df.hist(figsize=(15, 15))
plt.suptitle("Histograms of Features")
plt.show()
```

Explanation:

This code uses the matplotlib.pyplot library to create histograms for each feature in my dataset. The histograms provide a visual representation of the distribution of values for each feature. The figsize parameter adjusts the size of the overall figure, and plt.suptitle adds a title to the entire set of histograms.

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df["Cluster"], cmap='viridis')
plt.colorbar(label='Cluster')
plt.title('Clustering Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

Explanation:

This code creates a 2D scatter plot where each data point is represented by a dot, and the color of the dot corresponds to the cluster assigned by **KMeans**. This visualization helps in understanding the grouping or clustering of data points in a reduced-dimensional space.

These visualizations aid in the exploration and interpretation of my data, providing insights into the distribution of individual features and the grouping of data points into clusters.

Data Normalization:

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

In this code:

- I create a **StandardScaler** object named scaler.
- I fit the scaler on the training data (X_train) using **scaler.fit_transform(X_train)**, which computes the mean and standard deviation of the features in the training set and scales the features.
- I transform both the training and testing sets using the computed mean and standard deviation, ensuring that both sets are scaled consistently.

Normalization is essential in machine learning because it helps to bring different features to a similar scale, preventing one feature from dominating others during the training of models. The StandardScaler scales features to have a mean of 0 and a standard deviation of 1, making the features more comparable and contributing to better model performance.

Feature Selection:

```
X = df.drop("Label", axis=1)
y = df["Label"]

# SelectKBest feature selection
selector = SelectKBest(f_classif, k=10)
X_new = selector.fit_transform(X, y)

# Print selected features and their scores
selected_features = X.columns[selector.get_support()]
feature_scores = selector.scores_
feature_ranking = selector.pvalues_
print("Selected Features and Scores:")
for feature, score, rank in zip(selected_features, feature_scores, feature_ranking):
    print(f"{feature}: {score} (p-value: {rank})")
```

Explanation:

- X is defined as the features (independent variables) of my dataset, excluding the target variable "Label."
- y is defined as the target variable "Label."
- The SelectKBest method is instantiated with the chosen scoring function (f_classif for ANOVA F-statistic in this case) and the desired number of features (k=10).
- The fit_transform method is applied to select the top k features based on their scores.
- The selected features, along with their scores and p-values, are printed to provide insights into the feature selection process.

This section of your code helps identify and retain the most informative features for further analysis and model building. The selected features (X_new) are then used in subsequent steps, such as dimensionality reduction with PCA and clustering.

Principle Component Analysis:

```
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_new)
```

Explanation:

- The PCA class from scikit-learn's sklearn.decomposition module is instantiated with the desired number of principal components (n_components=5 in this case).

- The fit_transform method is used to perform PCA on the selected features (X_new), reducing the dimensionality of the data to the specified number of principal components.

- The transformed data, represented by the principal components, is stored in the variable X_pca for further analysis.

This section of the code utilizes PCA for dimensionality reduction, aiming to capture the most important information in the data while reducing the number of features. The reduced dataset (X_pca) can be used in subsequent steps, such as clustering or classification, where a lower-dimensional representation of the data may be beneficial.

Clustering:

```
kmeans = KMeans(n_clusters=3, random_state=42)
df["Cluster"] = kmeans.fit_predict(X_pca)
```

Explanation:

- The KMeans class from scikit-learn's sklearn.cluster module is instantiated with the desired number of clusters (n_clusters=3 in this case).

- The fit_predict method is used to apply KMeans clustering to the reduced dataset (X_pca). The cluster assignments are then added as a new column named "Cluster" to the DataFrame df.

This section of the code applies KMeans clustering to group data points based on their principal components, forming clusters that share similar characteristics. The resulting "Cluster" column in the DataFrame can be used for further analysis or visualization to understand the natural groupings within the data.

Classification:

In my code two types of classifiers are being used:

1. Random Forest Classifier:

```
clf = RandomForestClassifier(n_estimators=100, random_state=42)


cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(clf, X_train_scaled, y_train, cv=cv, scoring='accuracy')
print("Cross-Validation Mean Accuracy (Random Forest):", cv_scores.mean(), "±", cv_scores.std())


clf.fit(X_train_scaled, y_train)


y_pred = clf.predict(X_test_scaled)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred, zero_division=1))
```

Explanation:

Random Forest Classifier (clf):

- The RandomForestClassifier is instantiated with 100 decision trees (n_estimators=100) and a fixed random state for reproducibility (random_state=42).
- Cross-validation is performed using StratifiedKFold to assess the model's performance on the training data.
- The model is trained using fit on the scaled training data (X_train_scaled and y_train).
- Predictions are made on the scaled test data (X_test_scaled), and accuracy, confusion matrix, and classification report metrics are printed to evaluate the model's performance.

2. Support Vector Classifier (SVC) Classification

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


cv_svm = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores_svm = cross_val_score(clf_svm, X_train_scaled, y_train, cv=cv_svm, scoring='accuracy')
print("SVM Cross-Validation Mean Accuracy:", cv_scores_svm.mean(), "±", cv_scores_svm.std())


clf_svm.fit(X_train_scaled, y_train)


y_pred_svm = clf_svm.predict(X_test_scaled)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm, zero_division=1))
```

Explanation:

Support Vector Classifier (SVC) (clf_svm):

- The SVC is instantiated with a linear kernel (kernel='linear') and a regularization parameter C set to 1.
- Cross-validation is performed using StratifiedKFold to assess the model's performance on the training data.
- The model is trained using fit on the scaled training data (X_train_scaled and y_train).
- Predictions are made on the scaled test data (X_test_scaled), and accuracy, confusion matrix, and classification report metrics are printed to evaluate the model's performance.

Both classifiers are trained and evaluated separately, and their performances are compared based on metrics such as accuracy, confusion matrix, and classification report.

Final Evaluation and Accuracy:

```
final_accuracy_rf = accuracy_score(y_test, y_pred)
final_accuracy_svm = accuracy_score(y_test, y_pred_svm)

print(f"Final Random Forest Accuracy: {final_accuracy_rf}")
print(f"Final SVM Accuracy: {final_accuracy_svm}")


final_model = clf if final_accuracy_rf > final_accuracy_svm else clf_svm


final_y_pred = final_model.predict(X_test_scaled)
print("Final Accuracy:", accuracy_score(y_test, final_y_pred))
print("Final Confusion Matrix:\n", confusion_matrix(y_test, final_y_pred))
print("Final Classification Report:\n", classification_report(y_test, final_y_pred, zero_division=1))
```

Explanation:

1. Calculate Final Accuracies:
   - final_accuracy_rf: Compute the final accuracy of the Random Forest model on the test set (y_test and y_pred).
   - final_accuracy_svm: Compute the final accuracy of the Support Vector Classifier (SVC) model on the test set (y_test and y_pred_svm).

2. Print Final Accuracies:
   - Print the final accuracies of both models for comparison.

3. Choose the Best Model:
   - Determine the best model (final_model) based on the higher final accuracy. If the Random Forest accuracy is greater than the SVM accuracy, final_model is set to clf (Random Forest model); otherwise, it is set to clf_svm (SVC model).

4. Use the Best Model for Predictions:
   - Make final predictions (final_y_pred) on the test set using the selected best model (final_model).

5. Print Final Overall Accuracy and Evaluation Metrics:
   - Print the final overall accuracy and additional evaluation metrics (confusion matrix and classification report) for the best model on the test set.

This part of the code helps you compare the final accuracies of both models and select the best-performing model for making final predictions and obtaining comprehensive evaluation metrics. The decision on the best model is based on their final accuracies on the test set.

Conclusion

In this project, we undertook a comprehensive exploration and analysis of a dataset aimed at predicting different emotional states. The project encompassed various stages, including data import, cleaning, exploration, visualization, and normalization. Leveraging advanced machine learning techniques, we employed feature selection, principal component analysis (PCA), clustering, and classification to build robust models for predicting emotional states.

Key Highlights:

1. **Data Exploration and Cleaning:**

    - The dataset was thoroughly explored, revealing valuable insights into its structure and characteristics.
    - Missing values were addressed through appropriate data cleaning techniques to ensure the integrity of the analysis.

Feature Selection and Dimensionality Reduction:

- Feature selection techniques, such as SelectKBest, were employed to identify the most informative features for modeling.
- Principal Component Analysis (PCA) was used for dimensionality reduction, capturing essential information in a lower-dimensional space.

Clustering Analysis:

- KMeans clustering was applied to identify natural groupings within the data, aiding in understanding inherent patterns.

Classification Models:

- Two powerful classification models, Random Forest and Support Vector Classifier (SVC), were employed to predict emotional states.
- Cross-validation was used to assess model performance, and final accuracies were compared to select the best-performing model.

Conclusion and Model Selection:

- The project concludes with the selection of the best model based on final accuracies.
- The chosen model, whether Random Forest or SVC, represents a robust solution for predicting emotional states in the given dataset.

Final Thoughts:

This project demonstrates the application of advanced machine learning techniques to understand and predict emotional states. The combination of feature selection, dimensionality reduction, clustering, and classification provides a comprehensive approach to uncover patterns in the data and build accurate predictive models. The insights gained from this project contribute to the broader field of emotion prediction and can have practical applications in areas such as healthcare, human-computer interaction, and emotion-aware systems.