Ensemble Learning with Stacking using Scikit-Learn

Project Overview

This project demonstrates how to implement a **Stacking Classifier** using Python's Scikit-Learn library. Stacking, or stacked generalization, is an ensemble learning technique that combines multiple models to improve predictive performance. It involves training several base models and then using a meta-model to learn how to best combine their predictions.

Why Use Stacking?

- **Improved Accuracy**: By combining diverse models, stacking often achieves better performance than individual models.
- **Flexibility**: Allows the use of various base models, including classifiers and regressors.
- **Robustness**: Helps in reducing overfitting by leveraging the strengths of multiple models.

Prerequisites

Required Libraries

- `pandas` : For data manipulation and analysis.
- `numpy` : For numerical computations.
- `scikit-learn` : For machine learning algorithms and evaluation metrics.
- `matplotlib` & `seaborn` : For data visualization.

Installation

Install the necessary libraries using pip:

```
pip install pandas numpy scikit-learn matplotlib seaborn
```

Files Included

- `your_dataset.csv` : The dataset file containing the features and target variable.
- `stacking_classification.py` : The Python script implementing the Stacking Classifier.

Code Description

The implementation is divided into several key steps:

1. Importing Libraries

```
 import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

2. Loading and Exploring the Dataset

```
# Load the dataset
data = pd.read_csv('your_dataset.csv')

# Display the first few rows
print(data.head())
```

3. Preprocessing the Data

```
# Assuming the last column is the target variable
X = data.iloc[:, :-1]  # Features
y = data.iloc[:, -1]   # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

4. Defining Base Learners and Meta-Learner

```
# Define base learners
base_learners = [
    ('svc', SVC(probability=True, kernel='linear', random_state=42)),
    ('dt', DecisionTreeClassifier(max_depth=5, random_state=42))
]

# Define meta-learner
meta_learner = LogisticRegression()
```

5. Initializing and Training the Stacking Classifier

```
# Initialize the Stacking classifier
stacking_model = StackingClassifier(estimators=base_learners, final_estimator=meta_lear

# Train the model
stacking_model.fit(X_train, y_train)
```

6. Making Predictions

```
# Make predictions on the test set
y_pred = stacking_model.predict(X_test)
```

7. Evaluating the Model

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
# Classification report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", class_report)

# Accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy Score:", accuracy)
```

8. Visualizing the Confusion Matrix

```
 # Plot confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Expected Outputs

- **Confusion Matrix**: A table showing the performance of the classification model.
- **Classification Report**: Includes precision, recall, f1-score, and support for each class.
- **Accuracy Score**: The overall accuracy of the model.
- **Confusion Matrix Heatmap**: A visual representation of the confusion matrix.

Use Cases

- **Finance**: Predicting loan defaults or fraud detection.
- **Healthcare**: Disease classification based on patient data.
- **Marketing**: Customer segmentation and targeted advertising.
- **Manufacturing**: Predictive maintenance and quality control.

Future Enhancements

- **Hyperparameter Tuning**: Use techniques like Grid Search or Random Search for optimal model parameters.
- **Feature Engineering**: Analyze and select the most significant features to improve performance.
- **Model Comparison**: Compare with other classifiers to evaluate accuracy and efficiency.
- **Cross-Validation**: Implement cross-validation to ensure robustness and generalizability.

References

- Stacking Ensemble Machine Learning With Python
- Stacking in Machine Learning - GeeksforGeeks
- Stacking to Improve Model Performance: A Comprehensive Guide