Assignment No. 8

Problem Statement :
Implement ensemble learning techniques for classification tasks using approaches such as Voting, Bagging, Boosting, and Stacking

Objective :
To explore and apply ensemble learning methods by combining multiple base classifiers to enhance prediction accuracy, minimize overfitting, and improve model robustness in classification problems.

Prerequisite :

1. Python environment with libraries such as `numpy`, `pandas`, `matplotlib`, `seaborn`, and `scikit-learn`.
2. Fundamental understanding of classification algorithms (e.g., Logistic Regression, Decision Trees, SVM).
3. Familiarity with ensemble methods: voting, bagging, boosting, and stacking.

Theory :

Ensemble learning is a machine learning approach where multiple models—often referred to as "weak learners"—are combined to form a stronger, more accurate predictive model.
By aggregating outputs from diverse models, ensemble methods improve generalization and typically outperform individual models.They are especially effective in reducing bias, variance, and overfitting.

Types of Ensemble Techniques -
1. Voting Classifier
Description: Combines the predictions from multiple models using majority rule (hard voting) or average probabilities (soft voting).
Use Case: Ideal when combining models that have similar performance but make different errors.
Example: Logistic Regression, SVM, and Decision Tree combined using soft voting for multi-class classification.

2. Bagging (Bootstrap Aggregating)
Description: Trains multiple models (commonly Decision Trees) on different random subsets of the training data, and aggregates their predictions through majority voting.
Use Case: Helps reduce variance and prevent overfitting, particularly effective for high-variance models.
Example: BaggingClassifier with 100 Decision Trees trained on bootstrapped samples.

3. Boosting
Description: Builds models sequentially, with each new model correcting the errors of its predecessor.
Use Case: Excellent for reducing bias and constructing strong models from weak learners.

4. Stacking
Description: Trains several base models and combines their outputs using a meta-model for final predictions.
Use Case: Leverages the strengths of multiple models and improves overall performance through second-level learning.
Example: Base models include Logistic Regression, SVM, and Decision Tree; the meta-learner is a Gaussian Naive Bayes classifier.

Although ensemble techniques inherently mitigate overfitting, specific strategies help even more:

– Bagging reduces variance by averaging predictions across varied data subsets.

– Boosting sequentially focuses on hard-to-classify samples, reducing bias.

– Stacking depends on a well-tuned meta-model to generalize effectively.

Code & Output :

```
[21]: import numpy as np
      import pandas as pd

      from sklearn.linear_model import LogisticRegression
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.svm import SVC
      from xgboost import XGBClassifier
      from sklearn.ensemble import VotingClassifier, StackingClassifier
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import accuracy_score
      from sklearn.preprocessing import LabelEncoder, StandardScaler

[19]: !pip install xgboost
```

```
[11]: # ----------------------------
      # Load and Preprocess Titanic Data
      # ----------------------------
      data = pd.read_csv('Titanic-Dataset.csv')
      data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]
      data.dropna(inplace=True)

[13]: data
```

[13]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...
885	0	3	female	39.0	0	5	29.1250	Q
886	0	2	male	27.0	0	0	13.0000	S
887	1	1	female	19.0	0	0	30.0000	S
889	1	1	male	26.0	0	0	30.0000	C
890	0	3	male	32.0	0	0	7.7500	Q

712 rows × 8 columns

```
[27]: # Encode categorical features
label_encoders = {}
for col in ['Sex', 'Embarked']:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# Features and Target
X = data.drop('Survived', axis=1)
y = data['Survived']

# Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
[29]: # ----------------------------
# Train/Test Split
# ----------------------------
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
# 1. Voting Classifier
voting_model = VotingClassifier(estimators=[
    ('SVC', SVC(probability=True)),
    ('LR', LogisticRegression()),
    ('KNN', KNeighborsClassifier())
], voting='soft')
```

```
# 2. Stacking Classifier
stacking_model = StackingClassifier(
    estimators=[('svc', SVC()), ('knn', KNeighborsClassifier())],
    final_estimator=LogisticRegression()
)
```

```
# 3. XGBoost Classifier
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
```

```
model = stacking_model
```

```
# ----------------------------
# Train & Evaluate
# ----------------------------
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy: %.2f%%" % (accuracy_score(y_test, y_pred) * 100))
```

```
Accuracy: 81.82%
```

```
model = voting_model   # Change to:voting_model
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy: %.2f%%" % (accuracy_score(y_test, y_pred) * 100))
```

```
Accuracy: 81.12%
```

```
model = xgb_model   # Change to:voting_model
```

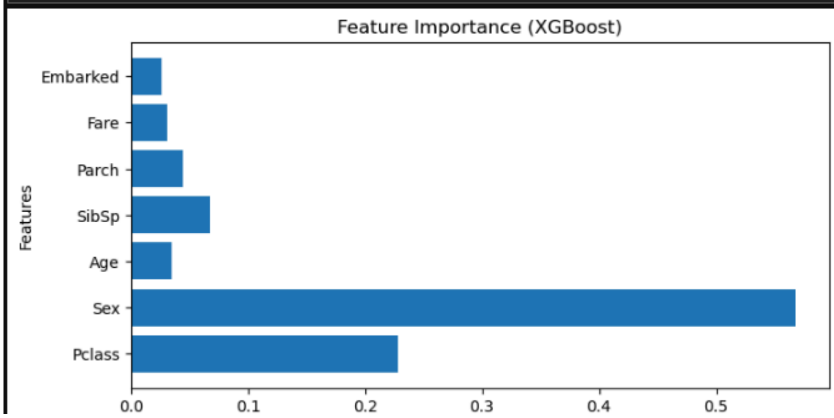
```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy: %.2f%%" % (accuracy_score(y_test, y_pred) * 100))
```

```
Accuracy: 76.22%
```

```
import matplotlib.pyplot as plt  # Make sure this line is included

if hasattr(model, 'feature_importances_'):
    plt.figure(figsize=(8, 4))
    plt.barh(X.columns, model.feature_importances_)
    plt.title("Feature Importance (XGBoost)")
    plt.xlabel("Importance")
    plt.ylabel("Features")
    plt.show()
```



Link to Github : https://github.com/Pranav-Divekar/Machine-learning-

Conclusion

In this assignment, we implemented and evaluated multiple ensemble methods, including Voting, Bagging, Boosting, and Stacking. Each approach offered distinct advantages in improving accuracy, stability, and generalization.

– Ensemble learning enhances model performance by integrating multiple base models.
– Voting is simple and useful when combining strong individual classifiers.
– Bagging effectively reduces variance and combats overfitting.
– Boosting improves accuracy by iteratively learning from mistakes.
– Stacking uses a layered learning approach to combine model predictions.
– Overall, ensemble learning is a powerful strategy for tackling classification problems with greater reliability and accuracy.