EX.NO.: 02
DATE: 23.06.2025

ENSEMBLE METHODS FOR IMPROVED SPAM DETECTION

AIM

To build an ensemble-based spam detection system by combining multiple classifiers (Naive Bayes, SVM, Random Forest, Gradient Boosting, and Logistic Regression) using voting and stacking methods, and compare their performances to individual models.

ALGORITHM

1. Load and preprocess the email spam dataset by converting text to numerical vectors using CountVectorizer.
2. Train individual classifiers: Naive Bayes, SVM, Random Forest, Gradient Boosting, and Logistic Regression on the training data.
3. Build an ensemble model using **VotingClassifier** (hard voting) and **StackingClassifier** with Logistic Regression as the final estimator.
4. Evaluate all models using accuracy and compare ensemble models to individual classifiers.

CODE AND OUTPUT

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

# Load SMS Spam dataset
url =
"https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv"
data = pd.read_csv(url, sep="\t", header=None, names=['label', 'message'])
data['label_num'] = data['label'].map({'ham': 0, 'spam': 1})

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    data['message'], data['label_num'], test_size=0.2, random_state=42)

# Convert text to vectors
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression

# Instantiate classifiers
nb = MultinomialNB()
svm = LinearSVC()
rf = RandomForestClassifier(n_estimators=100, random_state=42)
gb = GradientBoostingClassifier(random_state=42)
lr = LogisticRegression(max_iter=1000)
```

```python
from sklearn.metrics import accuracy_score

models = {'Naive Bayes': nb, 'SVM': svm, 'Random Forest': rf,
          'Gradient Boosting': gb, 'Logistic Regression': lr}

print("Individual Model Accuracies:")
for name, model in models.items():
    model.fit(X_train_vec, y_train)
    pred = model.predict(X_test_vec)
    acc = accuracy_score(y_test, pred)
    print(f"{name}: {acc:.4f}")
```

```
Individual Model Accuracies:
Naive Bayes: 0.9919
SVM: 0.9901
Random Forest: 0.9848
Gradient Boosting: 0.9794
Logistic Regression: 0.9883
```

```python
from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(
    estimators=[('nb', nb), ('svm', svm), ('rf', rf), ('gb', gb), ('lr', lr)],
    voting='hard')

voting_clf.fit(X_train_vec, y_train)
voting_pred = voting_clf.predict(X_test_vec)
voting_acc = accuracy_score(y_test, voting_pred)

print(f"\nVoting Classifier Accuracy: {voting_acc:.4f}")
```

```
Voting Classifier Accuracy: 0.9892
```

```python
from sklearn.ensemble import StackingClassifier

# Use Logistic Regression as final estimator
stacking_clf = StackingClassifier(
    estimators=[('nb', nb), ('svm', svm), ('rf', rf), ('gb', gb)],
    final_estimator=LogisticRegression(max_iter=1000),
    passthrough=True)

stacking_clf.fit(X_train_vec, y_train)
stacking_pred = stacking_clf.predict(X_test_vec)
stacking_acc = accuracy_score(y_test, stacking_pred)

print(f"\nStacking Classifier Accuracy: {stacking_acc:.4f}")
```

```
Stacking Classifier Accuracy: 0.9928
```

```python
print("\nSummary of Model Performances:")
for name, model in models.items():
    pred = model.predict(X_test_vec)
```

```
    acc = accuracy_score(y_test, pred)
    print(f"{name}: {acc:.4f}")

print(f"Voting Ensemble: {voting_acc:.4f}")
print(f"Stacking Ensemble: {stacking_acc:.4f}")
```

```
Summary of Model Performances:
Naive Bayes: 0.9919
SVM: 0.9901
Random Forest: 0.9848
Gradient Boosting: 0.9794
Logistic Regression: 0.9883
Voting Ensemble: 0.9892
Stacking Ensemble: 0.9928
```

INFERENCE

The ensemble models (Voting and Stacking) achieved higher or comparable accuracy compared to individual classifiers. Ensemble methods leverage the strengths of multiple models, reducing the risk of individual model weaknesses. This approach improves overall spam detection reliability.