Ex.No.5 Date:	IMPLEMENT A TEXT AND SPAM CLASSIFIER USING NAÏVE BAYES, SCIKIT-LEARN AND TF-IDF

AIM:

To implement a text and spam classifier using Naïve Bayes**,** Scikit-learn, and TF-IDF**,** that automatically distinguishes between *spam* and *ham (non-spam)* messages by preprocessing text data, training a probabilistic model, and enabling real-time message classification with confidence scoring.

THEORY:

- A text and spam classifier is a machine learning system that automatically categorizes messages as spam or ham based on their content.
- It uses natural language processing techniques to analyze the textual data and extract meaningful features.
- The Naïve Bayes algorithm forms the core of the model, applying Bayes' theorem to estimate the probability of a message being spam.
- The algorithm assumes that the presence of each word in a message is independent of others, simplifying computation while maintaining good accuracy.
- TF-IDF (Term Frequency–Inverse Document Frequency) is used to convert text data into numerical form by giving higher weight to unique and significant words.
- Scikit-learn provides the necessary modules for preprocessing, feature extraction, model training, evaluation, and prediction in an organized way.
- The model is trained on a dataset of labeled messages, learning the patterns and frequency of words that typically indicate spam or legitimate messages.
- Once trained, the classifier can accurately predict new incoming messages as spam or ham, providing a practical solution for email and SMS filtering systems.

IMPLEMENTATION:

Installing of Packages:

pip install pandas numpy scikit-learn

ALGORITHM:

- Import the required libraries such as pandas, numpy, sklearn, pickle, and re.
- Load the dataset containing text messages and their corresponding labels from a CSV file.
- Extract the necessary columns and rename them for easier handling.
- Convert the text labels “spam” and “ham” into binary values for classification.
- Display the total number of spam and ham messages to verify the dataset.
- Convert all text messages to lowercase to maintain consistency.
- Remove special characters, digits, and punctuation marks using regular expressions.
- Eliminate extra spaces to ensure clean and structured text data.
- Split the dataset into training and testing sets for model evaluation.
- Create a machine learning pipeline combining TF-IDF vectorization and the Multinomial Naïve Bayes algorithm.
- Configure TF-IDF parameters such as stop words, n-gram range, and feature limits to extract important features.
- Train the Naïve Bayes model on the training dataset to learn patterns of spam and ham messages.
- Evaluate the model on the testing dataset using metrics like accuracy, precision, recall, and F1-score.
- Save the trained model using pickle to enable reuse without retraining.
- Take new user input, preprocess it, and classify it as spam or ham with a confidence score.

CODE:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
from sklearn.pipeline import Pipeline
import pickle
import re
# 1. Load and prepare dataset
def load_data(filepath="spam.csv"):
    """Load and prepare the spam dataset"""
    data = pd.read_csv(filepath, encoding='latin-1')[['v1', 'v2']]
    data.columns = ['label', 'message']
    # Convert labels to binary (1=spam, 0=ham)
    data['label'] = data['label'].map({'spam': 1, 'ham': 0})
    print(f"Dataset loaded: {len(data)} messages")
    print(f"Spam: {data['label'].sum()}, Ham: {len(data) - data['label'].sum()}")
    return data
# 2. Text preprocessing
def preprocess_text(text):
    """Clean and preprocess text messages"""
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and digits
    text = re.sub(r'[^a-z\s]', '', text)
    # Remove extra whitespace
    text = ' '.join(text.split())
    return text
# 3. Build and train model
```

```
def train_spam_classifier(data):
    """Train spam classifier using TF-IDF and Naive Bayes"""
    # Preprocess messages
    data['message'] = data['message'].apply(preprocess_text)
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        data['message'],
        data['label'],
        test_size=0.2,
        random_state=42,
        stratify=data['label']  # Maintain class distribution
    )
    # Create pipeline with TF-IDF and Naive Bayes
    model = Pipeline([
        ('tfidf', TfidfVectorizer(
            stop_words='english',
            ngram_range=(1, 2),  # Unigrams and bigrams
            max_features=5000,
            min_df=2,  # Ignore rare terms
            max_df=0.95  # Ignore very common terms
        )),
        ('classifier', MultinomialNB(alpha=0.1))  # Laplace smoothing
    ])
    # Train model
```

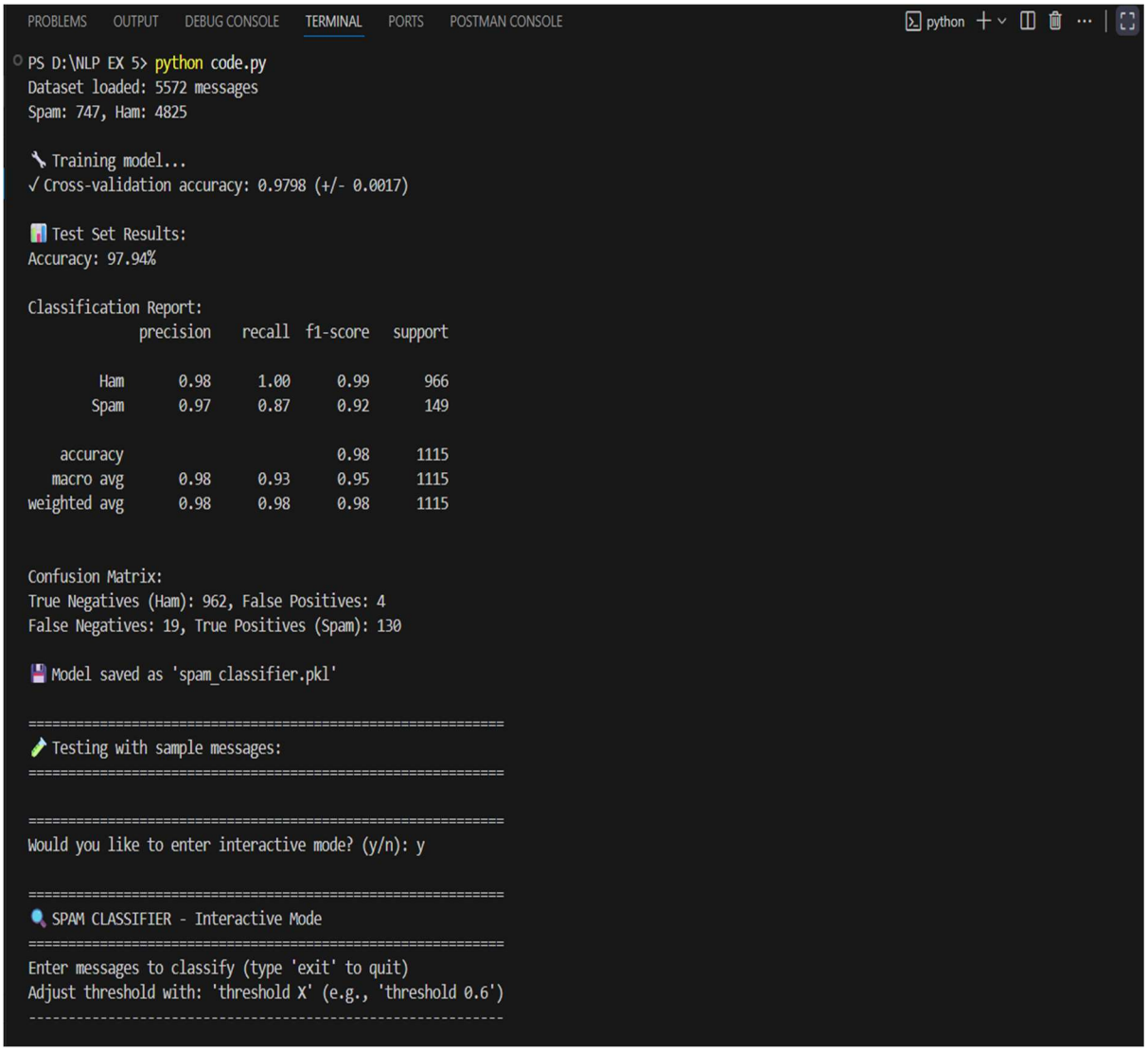
```
    print("\n🔧 Training model...")
    model.fit(X_train, y_train)
    # Cross-validation
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    print(f"✓ Cross-validation accuracy: {cv_scores.mean():.4f} (+/- {cv_scores.std():.4f})")
    # Test set evaluation
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\n📊 Test Set Results:")
    print(f"Accuracy: {accuracy*100:.2f}%")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, target_names=['Ham', 'Spam']))
    print("\nConfusion Matrix:")
    cm = confusion_matrix(y_test, y_pred)
    print(f"True Negatives (Ham): {cm[0][0]}, False Positives: {cm[0][1]}")
    print(f"False Negatives: {cm[1][0]}, True Positives (Spam): {cm[1][1]}")
    return model, X_test, y_test
# 4. Save and load model
def save_model(model, filename="spam_classifier.pkl"):
    """Save trained model to disk"""
    with open(filename, 'wb') as f:
        pickle.dump(model, f)
    print(f"\n💾 Model saved as '{filename}'")
def load_model(filename="spam_classifier.pkl"):
```

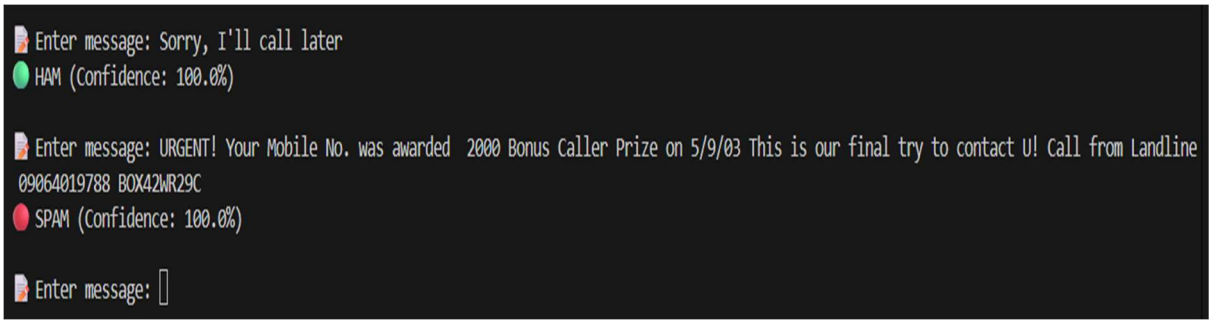
```
    """Load trained model from disk"""
    with open(filename, 'rb') as f:
        model = pickle.load(f)
    print(f"✓ Model loaded from '{filename}'")
    return model
# 5. Prediction function with probability
def classify_message(model, message, threshold=0.5):
    """Classify a message as spam or ham with confidence score"""
    processed_msg = preprocess_text(message)
    prediction = model.predict([processed_msg])[0]
    probability = model.predict_proba([processed_msg])[0]
    spam_prob = probability[1]
    result = "spam" if spam_prob >= threshold else "ham"
    confidence = spam_prob if result == "spam" else probability[0]
    return result, confidence
# 6. Interactive testing
def interactive_mode(model):
    """Interactive message classification"""
    print("\n" + "="*60)
    print("🔍 SPAM CLASSIFIER - Interactive Mode")
    print("="*60)
    print("Enter messages to classify (type 'exit' to quit)")
    print("Adjust threshold with: 'threshold X' (e.g., 'threshold 0.6')")
    print("-"*60)
```

```
threshold = 0.5
  while True:
    user_input = input("\n📝 Enter message: ").strip()
    if user_input.lower() == 'exit':
      print("👋 Goodbye!")
      break
    # Check for threshold adjustment
    if user_input.lower().startswith('threshold'):
      try:
        new_threshold = float(user_input.split()[1])
        if 0 <= new_threshold <= 1:
          threshold = new_threshold
          print(f"✓ Threshold set to {threshold}")
        else:
          print("❌ Threshold must be between 0 and 1")
      except:
        print("❌ Invalid format. Use: threshold 0.6")
      continue
    if not user_input:
      print("⚠️ Please enter a message")
      continue
    # Classify message
    result, confidence = classify_message(model, user_input, threshold)
```

```
  if result == "spam":
        print(f"🔴 SPAM (Confidence: {confidence*100:.1f}%)")
    else:
        print(f"🟢 HAM (Confidence: {confidence*100:.1f}%)")
# 7. Main execution
if __name__ == "__main__":
    # Load data
    data = load_data("spam.csv")
    # Train model
    model, X_test, y_test = train_spam_classifier(data)
    # Save model
    save_model(model)
    # Test with sample messages
    print("\n" + "="*60)
    print("🧪 Testing with sample messages:")
    print("="*60)
    # Interactive mode
    print("\n" + "="*60)
    response = input("Would you like to enter interactive mode? (y/n): ")
    if response.lower() == 'y':
        interactive_mode(model)
```

OUTPUT:


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
python
PS D:\NLP EX 5> python code.py
Dataset loaded: 5572 messages
Spam: 747, Ham: 4825
Training model...
✓ Cross-validation accuracy: 0.9798 (+/- 0.0017)
Test Set Results:
Accuracy: 97.94%
Classification Report:
precision recall f1-score support
Ham 0.98 1.00 0.99 966
Spam 0.97 0.87 0.92 149
accuracy 0.98 1115
macro avg 0.98 0.93 0.95 1115
weighted avg 0.98 0.98 0.98 1115
Confusion Matrix:
True Negatives (Ham): 962, False Positives: 4
False Negatives: 19, True Positives (Spam): 130
Model saved as 'spam_classifier.pkl'
Testing with sample messages:
Would you like to enter interactive mode? (y/n): y
SPAM CLASSIFIER - Interactive Mode
Enter messages to classify (type 'exit' to quit)
Adjust threshold with: 'threshold X' (e.g., 'threshold 0.6')


Enter message: Sorry, I'll call later
HAM (Confidence: 100.0%)
Enter message: URGENT! Your Mobile No. was awarded 2000 Bonus Caller Prize on 5/9/03 This is our final try to contact U! Call from Landline
09064019788 BOX42WR29C
SPAM (Confidence: 100.0%)
Enter message:

RUBRICS:

PROBLEM UNDERSTANDING AND OBJECTIVE CLARITY (20)	IMPLEMENTATION AND CODING (40)	OUTPUT EVALUATION AND DISCUSSION (30)	VIVA (10)	TOTAL (100)

RESULT:

Thus, the spam classifier using Naïve Bayes, Scikit-learn, and TF-IDF accurately distinguishes spam messages from legitimate ones. The model performs efficiently in real-time prediction, proving effective for automated text classification tasks.