SMS Spam Classifier Project Explanation (A to Z Notes)

📚 **Project Title:**

SMS Spam Classifier using Logistic Regression in Python (with user input)

🔍 **Objective:**

To create a machine learning model that can classify SMS messages as **"Spam"** or **"Not Spam"** using logistic regression, and allow real-time user input for predictions.

✅ **Step-by-Step Breakdown:**

1. Import Required Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

- These libraries are used for:
    - Data handling (numpy, pandas)
    - Text processing (CountVectorizer)
    - Model building (LogisticRegression)
    - Evaluation (accuracy_score, classification_report)
    - Visualization (matplotlib, seaborn)

---

2. Load Dataset

```
data = pd.read_csv('SMSSpamCollection', sep='\t', names=['label', 'text'], encoding='utf-8')
data['label'] = data['label'].map({'spam': 1, 'ham': 0})
data.head()
```

- The dataset is loaded from a text file.
- It's tab-separated, so sep='\t' is used.
- Columns are renamed to label and text.
- Labels are mapped: spam = 1, ham = 0.

---

3. Feature Extraction

vectorizer = CountVectorizer()

X = vectorizer.fit_transform(data['text'])

y = data['label']

- Text data (text) is converted to a numerical form using CountVectorizer.
- X contains the vectorized messages.
- y contains the labels.

---

4. Train-Test Split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

- Data is split into training (80%) and testing (20%) parts.
- random_state=42 ensures reproducibility.

---

5. Model Training

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

- Logistic Regression is used for binary classification.
- max_iter=1000 ensures convergence during training.

---

6. Model Evaluation

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f'Model Accuracy: {accuracy:.2f}')

print(classification_report(y_test, y_pred))

- Model predictions are made on the test set.
- Accuracy score and classification report are printed.

---

7. Confusion Matrix

```
plt.figure(figsize=(8, 6))
sns.heatmap(pd.crosstab(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

- A confusion matrix is visualized to show true/false positives and negatives.

---

8. Prediction Function

```
def predict_spam(message):
    message_vectorized = vectorizer.transform([message])
    prediction = model.predict(message_vectorized)[0]
    return 'Spam' if prediction == 1 else 'Not Spam'
```

- Takes a message string, vectorizes it, and predicts using the trained model.
- Returns 'Spam' or 'Not Spam'.

---

9. User Input Loop

```
while True:
    user_input = input("\nEnter a message to check (or type 'exit' to stop): ")
    if user_input.lower() == 'exit':
        print("Exiting spam predictor.")
        break
    result = predict_spam(user_input)
    print(f"Prediction: {result}")
```

- Allows user to enter messages one-by-one.
- Type exit to stop.
- Classifies each entered message in real time.

🔧 **Tools Used:**

- Dataset: SMSSpamCollection
- Algorithm: Logistic Regression
- Vectorizer: CountVectorizer (Bag-of-Words model)

🎯 **Final Output:**

- Accuracy score
- Classification report
- Confusion matrix
- Real-time user prediction capability

✉️ **Example Test Inputs:**

Try inputs like:

- "Win a free ticket to Bahamas"
- "Hey, let’s catch up tomorrow"
- "Get rich fast now!"
- "Meeting rescheduled to 4pm"

This covers everything A to Z of how the SMS spam classifier works.