Ayan_Sur_002410504003_Assignment_2

Question 2. Write Python code to implement Logistic Regression Model for Spam detection.

THEORY

Goals of Logistic Regression for Spam Detection:

1.Probability Estimation: To estimate the probability of a given email being spam or not spam(ham).

2.Feature Interpretation: To understand the impact of different features (like the presence of certain keywords) on the likelihood of an email being spam.

Equation: The Logistic Regression model uses the logistic function (also called the sigmoid function) to model a binary dependent variable. The equation is as follows:

$p(y = 1 \mid x) = 1/(1 + e - (\beta 0 + \beta 1\, x 1 + \beta 2\, x 2 + \ldots + \beta n\, x n))$

Where:

$p(y = 1 \mid x)$ is the probability that the email is spam (y=1) given the feature vector x .

$\beta 0$ is the intercept.

$\beta 1, \beta 2, \ldots, \beta n$ are the coefficients for the features $x 1, x 2, \ldots, x n$.

e is the base of the natural logarithm.

Pros of Using Logistic Regression:

1.Simplicity and Efficiency : Easy to implement and interpret,Computationally efficient and quick to train, even on large datasets.

2.Probabilistic Output: Provides probabilities for predictions, which is useful for thresholding and making decisions based on confidence levels.

3.Feature Importance: Coefficients provide insights into the importance of features.

4.Less Prone to Overfitting: Regularization techniques (like L1 and L2) can be applied to prevent overfitting.

Cons of Using Logistic Regression:

1.Linearity Assumption: Assumes a linear relationship between the features and the log-odds of the outcome, which may not always be true.

2.Feature Scaling Required: Requires features to be scaled for better performance.

3.Binary Output: Primarily used for binary classification; may not perform as well on multi-class problems without modifications.

CODE

```
# Importing Required Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.utils.class_weight import compute_class_weight

# Load the data
raw_mail_data = pd.read_csv('/content/mail_data.csv', encoding='latin-1')

# Replace null values with an empty string
mail_data = raw_mail_data.where(pd.notnull(raw_mail_data), '')

# Label encoding: spam -> 0, ham -> 1
mail_data['Category'] = mail_data['Category'].map({'spam': 0, 'ham': 1})

# Drop rows where 'Category' is NaN
mail_data = mail_data.dropna(subset=['Category'])

# Separate features and labels
X = mail_data['Message']
Y = mail_data['Category']
print(X)
print(Y)
# Split 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=3, stratify=Y)
```

```
# Feature extraction using TF-IDF with enhanced preprocessing
vectorizer = TfidfVectorizer(
    stop_words='english',
    max_features=10000,  # Increase feature set
    lowercase=True,
    ngram_range=(1, 3),  # Use unigrams, bigrams, and trigrams
    sublinear_tf=True  # Apply sublinear term frequency scaling
)

X_train_features = vectorizer.fit_transform(X_train)
X_test_features = vectorizer.transform(X_test)

# Convert Y_train and Y_test to integers
Y_train = Y_train.astype(int)
Y_test = Y_test.astype(int)

# Handle class imbalance
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(Y_train), y=Y_train)
class_weight_dict = dict(enumerate(class_weights))

# Train logistic regression with regularization and class weights
model = LogisticRegression(
    solver='liblinear',
    penalty='l2',  # L2 regularization
    class_weight=class_weight_dict,
    C=1.0  # Adjust regularization strength
)
model.fit(X_train_features, Y_train)

# Evaluate the model
# Accuracy on training data
train_predictions = model.predict(X_train_features)
training_accuracy = accuracy_score(Y_train, train_predictions)
print('Accuracy on training data:', training_accuracy)

# Accuracy on test data
test_predictions = model.predict(X_test_features)
testing_accuracy = accuracy_score(Y_test, test_predictions)
print('Accuracy on test data:', testing_accuracy)

# Predictive system
input_mail = ["Congratulations!! You have won Iphone 16,Click here and Claim your prize!!!!"]
input_data_features = vectorizer.transform(input_mail)
prediction = model.predict(input_data_features)

if prediction[0] == 1:
    print('Ham mail')
else:
    print('Spam mail')
```

```
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                              ...
5583    This is the 2nd time we have tried 2 contact u...
5584                Will Ã¼ b going to esplanade fr home?
5585    Pity, * was in mood for that. So...any other s...
5586    The guy did some bitching but I acted like i'd...
5587                            Rofl. Its true to its name
Name: Message, Length: 5587, dtype: object
0       1.0
1       1.0
2       0.0
3       1.0
4       1.0
       ...
5583    0.0
5584    1.0
5585    1.0
5586    1.0
5587    1.0
Name: Category, Length: 5587, dtype: float64
Accuracy on training data: 0.9923920340120832
Accuracy on test data: 0.9874776386404294
Spam mail
```

Explanation :

Training Accuracy:

Training accuracy is calculated by using the trained model to predict the labels of the data it was trained on (X_train_features in the code) and comparing these predictions to the true labels (Y_train).

Training accuracy represents how well the model has learned the patterns in the training data. It's a measure of how well the model fits the data it was trained on.

Testing Accuracy:

Testing accuracy is calculated by using the trained model to predict the labels of a separate, unseen dataset (X_test_features) and comparing these predictions to the true labels (Y_test).

Testing accuracy gives an estimate of how well the model is likely to perform on new, unseen data. This is a crucial metric because the goal of machine learning models is to generalize well to new data.

The code performs the following steps:

1.***Data Loading and Preprocessing***:

Loads email data from a CSV file. Replaces missing values and converts labels to numerical form (spam: 0, ham: 1).

2.***Data Splitting***:

Divides the data into training and testing sets using train_test_split. This ensures a portion of the data is reserved for evaluating the model's performance on unseen examples.

3.***Feature Extraction (TF-IDF)***:

Uses TF-IDF (Term Frequency-Inverse Document Frequency) to convert text emails into numerical vectors that the model can understand. It considers word frequencies and their importance across the entire dataset to create these representations.

4.***Model Training***:

Trains a Logistic Regression model using the training data. It learns patterns and relationships between email features and their labels (spam or ham).

5.***Model Evaluation***:

Predicts labels for the training and testing sets using the trained model. Calculates the accuracy scores by comparing predictions to the true labels. Prints these accuracy scores, which represent the model's performance on training and unseen data.

6.***Prediction on New Email***:

Takes a sample input email, converts it into numerical features using TF-IDF, and then predicts its label using the trained model. Prints "Spam mail" or "Ham mail" based on the prediction.

7.***The Output***:

Accuracy on training data: A value close to 1 (e.g., 0.99) indicates the model has learned the training data well. Accuracy on test data: A high value (similar to training accuracy) suggests the model generalizes well to unseen emails. "Spam mail" or "Ham mail": This is the prediction for the provided input email.

In summary, the output shows the model's performance on training and testing data and provides a prediction for a sample email. High accuracy scores indicate good performance.

The accuracy scores show that the model is performing quite well, with high accuracy on both training and test data. This suggests it has learned to effectively distinguish between spam and ham emails.

The final output "**Spam mail**" in this example means the model classified the provided input email as spam. This is based on the features extracted from the text by the TF-IDF vectorizer.