

Homework5 code

November 18, 2024

1 Spam ham classifier

Consider a text classification problem. In this case, you will try to classify text as either spam or ham. To do this, you will apply concepts of Likelihood, prior, and posterior given a dataset comprising pairs of text and labels. There are two types of labels: 1 (spam) and 0 (ham). Your goal is to create a simple classifier that, when given, determines if the text is spam or ham.

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import nltk
from nltk.corpus import stopwords
from sklearn.metrics import accuracy_score
import re
```

1.1 Data loading and cleaning

```
[2]: data = pd.read_csv('spam_ham_dataset.csv')
data = data.dropna()
data = data.drop_duplicates()
df = data
nltk.download('stopwords')
stop = stopwords.words('english')
```

[nltk_data] Downloading package stopwords to /home/kip/nltk_data...

[nltk_data] Package stopwords is already up-to-date!

```
[3]: def remove_stopwords(text):
    text = [word.lower() for word in text.split() if word.lower() not in stop]
    # remove special characters
    text = [re.sub('\W+', '', word) for word in text]

    return " ".join(text)

txt1 = df.text[0]
```

```

print(txt1)
# remove the filler words
txt1 = remove_stopwords(txt1)
print(txt1)
# apply the function to the entire dataset
df['text'] = df['text'].apply(remove_stopwords)
df.head()

```

```

<>:4: SyntaxWarning: invalid escape sequence '\W'
<>:4: SyntaxWarning: invalid escape sequence '\W'
/tmp/ipykernel_1420/2336932942.py:4: SyntaxWarning: invalid escape sequence '\W'
    text = [re.sub('\W+', '', word) for word in text]

```

Subject: enron methanol ; meter # : 988291
 this is a follow up to the note i gave you on monday , 4 / 3 / 00 { preliminary
 flow data provided by daren } .
 please override pop ' s daily volume { presently zero } to reflect daily
 activity you can obtain from gas control .
 this change is needed asap for economics purposes .
 subject enron methanol meter 988291 follow note gave monday 4 3 00
 preliminary flow data provided daren please override pop daily volume
 presently zero reflect daily activity obtain gas control change needed asap
 economics purposes

```

[3]: Unnamed: 0 label text \
0      605 ham subject enron methanol meter 988291 follow ...
1     2349 ham subject hpl nom january 9 2001 see attached ...
2     3624 ham subject neon retreat ho ho ho around wonderf...
3     4685 spam subject photoshop windows office cheap mai...
4     2030 ham subject indian springs deal book teco pvr rev...

label_num
0      0
1      0
2      0
3      1
4      0

```

Next we split the data into training and testing. We will derive the probabilities from the training data and then use them to predict the testing data.

```

[4]: # lets split the data into training and testing
X = df.text
y = df.label_num
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

```

1.1.1 (1)

Find the priors. What are the priors in this distribution? i.e find $P(\text{ham})$ and $P(\text{spam})$

```
[5]: # Calculate priors
P_ham = y_train.value_counts()[0] / len(y_train)
P_spam = y_train.value_counts()[1] / len(y_train)

print(f"P(ham): {P_ham}")
print(f"P(spam): {P_spam}")
```

P(ham): 0.7084139264990329

P(spam): 0.2915860735009671

1.1.2 (2)

Find the likelihoods for each word. For each word in the dataset, find the likelihood that the word is in spam and ham. This will represent the conditional probability $P(w|\text{spam})$ and $P(w|\text{ham})$ for w where $w \in V$. V is the vocabulary of the dataset.

```
[6]: from collections import defaultdict, Counter

# Initialize counters for spam and ham words
spam_words = Counter()
ham_words = Counter()

# Separate spam and ham texts
spam_texts = X_train[y_train == 1]
ham_texts = X_train[y_train == 0]

# Count words in spam and ham texts
for text in spam_texts:
    for word in text.split():
        spam_words[word] += 1

for text in ham_texts:
    for word in text.split():
        ham_words[word] += 1

# Calculate total number of words in spam and ham texts
total_spam_words = sum(spam_words.values())
total_ham_words = sum(ham_words.values())

# Calculate likelihoods
likelihoods_spam = {word: (count / total_spam_words) for word, count in spam_words.items()}
likelihoods_ham = {word: (count / total_ham_words) for word, count in ham_words.items()}
```

```
print("Likelihoods for spam words:", list(likelihoods_spam.items())[:10])
print("Likelihoods for ham words:", list(likelihoods_ham.items())[:10])
```

```
Likelihoods for spam words: [('subject', 0.008392657843131073), ('message',
0.001513326733547301), ('hey', 0.00020808242586275387), ('julie',
1.2611056112894174e-05), ('_', 0.004785895794843339), ('turned',
8.827739279025922e-05), ('18', 0.00030266534670946016), ('high',
0.0008134131192816742), ('school', 8.827739279025922e-05), ('senior',
8.197186473381214e-05)]
```

```
Likelihoods for ham words: [('subject', 0.01633299063026014), ('april',
0.0013167581802791138), ('activity', 0.0006810818173857485), ('surveys',
4.216220774292729e-05), ('starting', 0.00016216233747279728), ('collect',
3.8918960993471346e-05), ('data', 0.000535135713660231), ('attached',
0.002737300256540818), ('survey', 0.00037945986968634563), ('drives',
1.6216233747279726e-05)]
```

1.1.3 (3)

Define a function that, when given a text sequence, returns the probability of the text being in spam. I.e., it returns $P(\text{spam}|\text{text})$. Note that this function calculates the likelihood using the Bayes rule. Do the same for ham.

```
[7]: def calculate_posterior(text, priors, likelihoods, total_words):
    words = text.split()
    posterior = np.log(priors)

    for word in words:
        if word in likelihoods:
            posterior += np.log(likelihoods[word])
        else:
            # Apply Laplace smoothing for unseen words
            posterior += np.log(1 / (total_words + len(likelihoods)))

    return posterior

def predict_spam(text):
    P_spam_given_text = calculate_posterior(text, P_spam, likelihoods_spam,
    ↪total_spam_words)
    return P_spam_given_text

def predict_ham(text):
    P_ham_given_text = calculate_posterior(text, P_ham, likelihoods_ham,
    ↪total_ham_words)
    return P_ham_given_text

# Example usage
text_example = "free money now"
print(f"P(spam|text): {predict_spam(text_example)}")
```

```
print(f"P(ham|text): {predict_ham(text_example)}")
```

P(spam|text): -26.23636612458862

P(ham|text): -29.381719006113546

1.1.4 (4)

Perform inference, i.e., given a string of text, determine if it is ham or spam based on the posterior probabilities calculated from the previous steps. Your function will determine the posterior probability of your text being in ham and spam and classify it as being the larger of the two.

```
[9]: def classify_text(text):
    P_spam_given_text = predict_spam(text)
    P_ham_given_text = predict_ham(text)

    if P_spam_given_text > P_ham_given_text:
        return 'spam'
    else:
        return 'ham'

# Example usage
text_example = "free money now"
classification = classify_text(text_example)
print(f"The text '{text_example}' is classified as: {classification}")
```

The text 'free money now' is classified as: spam

1.1.5 (5)

Evaluate the data based on your test set and report the accuracy of your classifier. Your accuracy must be greater than 85%.

```
[11]: # Predict the labels for the test set
y_pred = X_test.apply(classify_text)

# Convert predictions to numerical labels
y_pred_num = y_pred.apply(lambda x: 1 if x == 'spam' else 0)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_num)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Accuracy: 97.00%