Name : Bodhisatya Ghosh
Class : CSE DS
UID : 2021700026
Subject : NLP
Experiment number : 4

Aim: Perform document classification using NB(no use of library of NB classifier)
Copy abstract of papers and create labelled dataset.
Create word count vectors. Each row represents a document, and each column represents a word.
Read text from set of test documents and classify the unlabelled documents.
Generate confusion matrix, and calculate accuracy, precision, recall, F1 score.

Theory

The Naive Bayes algorithm is a supervised machine learning algorithm based on the Bayes' theorem. It is a probabilistic classifier that is often used in NLP tasks like sentiment analysis (identifying a text corpus' emotional or sentimental tone or opinion).

The Bayes' theorem is used to determine the probability of a hypothesis when prior knowledge is available. It depends on conditional probabilities. The formula is given below :

Naive Bayes Classifier:
P(A|B) is posterior probability i.e. the probability of a hypothesis A given the event B occurs.
P(B|A) is likelihood probability i.e. the probability of the evidence given that hypothesis A is true.
P(A) is prior probability i.e. the probability of the hypothesis before observing the evidence
P(B) is marginal probability i.e. the probability of the evidence.

In [ ]:
```
import pandas as pd
import numpy as np
import math
import nltk
import matplotlib.pyplot as plt
```

Load data (Fake news dataset)

In [ ]:
```
data = pd.read_csv("./data.csv", usecols=['text','label'])
df = data.copy().iloc[0:,:]
```

Preprocessing data

Dropping null values

In [ ]:
```
df.dropna(inplace=True)
```

Removing non-alphanumeric characters

In [ ]:
```
import re
def cleanText(string):
    result = re.sub(r'[^A-Za-z]',' ',string)    #remove non-alphanumeric characters
    result = re.sub(r'\s+',' ',result)
    result = result.lower()
    return result
```

Removing stop words

In [ ]:
```
df['text'] = df['text'].apply(lambda sent: cleanText(sent)).values
```

In [ ]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
```

Performing lemmatization

In [ ]:
```
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()

def lemmatize_text(text):
    st = ""
    for w in w_tokenizer.tokenize(text):
```

```
        st = st + lemmatizer.lemmatize(w) + " "
    return st
df['text'] = df.text.apply(lemmatize_text)
```

Split data 80:20 train:test

In [ ]:
```
text = df['text'].values
label = df['label'].values
```

In [ ]:
```
from sklearn.model_selection import train_test_split
train_text, test_text, train_label, test_label =  train_test_split(text, label, random_state=100, test_size=0.2)
```

Building the Naive Bayes classifier

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
from collections import defaultdict
```

Create word vectors

In [ ]:
```
vec = CountVectorizer(max_features = 3000)
X = vec.fit_transform(train_text)
vocab = vec.get_feature_names_out()
X = X.toarray()
word_counts = {}
for l in range(2):
    word_counts[l] = defaultdict(lambda: 0)
for i in range(X.shape[0]):
    l = train_label[i]
    for j in range(len(vocab)):
        word_counts[l][vocab[j]] += X[i][j]
```

Apply Laplace Smoothing

In [ ]:
```
def laplace_smoothing(n_label_items, vocab, word_counts, word, text_label):
    a = word_counts[text_label][word] + 1
    b = n_label_items[text_label] + len(vocab)
```

```
    return math.log(a/b)

def group_by_label(x, y, labels):
    data = {}
    for l in labels:
        data[l] = x[np.where(y == l)]
    return data

def fit(x, y, labels):
    n_label_items = {}
    log_label_priors = {}
    n = len(x)
    grouped_data = group_by_label(x, y, labels)
    for l, data in grouped_data.items():
        n_label_items[l] = len(data)
        log_label_priors[l] = math.log(n_label_items[l] / n)
    return n_label_items, log_label_priors

def predict(n_label_items, vocab, word_counts, log_label_priors, labels, x):
    result = []
    for text in x:
        label_scores = {l: log_label_priors[l] for l in labels}
        words = set(w_tokenizer.tokenize(text))
        for word in words:
            if word not in vocab: continue
            for l in labels:
                log_w_given_l = laplace_smoothing(n_label_items, vocab, word_counts, word, l)
                label_scores[l] += log_w_given_l
        result.append(max(label_scores, key=label_scores.get))
    return result
```

In [ ]:
```
from sklearn.metrics import accuracy_score,confusion_matrix, precision_score, recall_score, f1_score
```

In [ ]:
```
labels = [0,1]
n_label_items, log_label_priors = fit(train_text,train_label,labels)
pred = predict(n_label_items, vocab, word_counts, log_label_priors, labels, test_text)
print("Accuracy of prediction on test set : ", accuracy_score(test_label,pred))
print("Precision of prediction on test set : ", precision_score(test_label,pred))
print("Recall of prediction on test set : ", recall_score(test_label,pred))
print("F1 score of prediction on test set : ", f1_score(test_label,pred))
```

```
Accuracy of prediction on test set :  0.7278590748318191
Precision of prediction on test set :  0.9745921369350093
Recall of prediction on test set :  0.48762210624916363
F1 score of prediction on test set :  0.6500178380306814
```

In [ ]:
```
print("Confusion matrix of prediction on test set : ", confusion_matrix(test_label,pred))
```

```
Confusion matrix of prediction on test set :  [[6851   95]
 [3829 3644]]
```

Curiosity questions

1. What is the relation between accuracy and precision?

Precision:

Definition: Precision is the ratio of correctly predicted positive observations to the total predicted positives. It focuses on the accuracy of the positive predictions.
Interpretation: High precision means that the model has fewer false positives.

Accuracy:

Definition: Accuracy is the ratio of correctly predicted observations (both true positives and true negatives) to the total observations.
Interpretation: Accuracy provides an overall measure of the model's correctness, but it can be misleading if there is an imbalanced class distribution.

2. Give example where precision is significant compared to accuracy?

Precision focuses more on the ability to predict a single class, rather than the overall ability to classify. Precision is more important than accuracy in situations where wrong predictions have heavy consequences. For example: Medical predicition of a rare disease should have high precision as predicting the wrong class may have dire consequences

3. Give example where accuracy is significant compared to precision?

Accuracy is more important than precision in situations where wrong predictions do not have serious implications. For example: Spam email classification should have a high accuracy the consequences of both false positives and false negatives are important, but neither is inherently more critical than the other.