3/11/20

EXPERIMENT NO. 6

1KS17CS030

AIM: Assuming a set of documents that need to be classified, use the naive Bayesian classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision and recall for your dataset.

Dataset:

message	label
I love this sandwich	pos
This is an amazing place	pos
I feel very good about these beers	pos
This is my best work	pos
What an awesome view	pos
I donot like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg
He is my sworn enemy	neg
My boss is horrible	neg
This is an awesome place	pos
I donot like the taste of this juice	neg
I love to dance	pos
I am sick & tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemy's house today	neg

Algorithm:

LEARN_NAIVE_BAYES_TEXT (Examples, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k | v_j)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. collect all words, punctuation and other tokens that occur in Examples.

   - Vocabulary ← the set of all distinct words and other tokens occurring in any text document from Examples.

2. calculate the required $P(v_j)$ and $P(w_k | v_j)$ probability terms

   - For each target value v_j in V do
     - $docs_j$ ← the subset of documents from Examples for which the target value is v_j.
     - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
     - $Text_j$ ← a single document created by concatenating all members of $docs_j$.
     - n ← total number of distinct word positions in $Text_j$
     - for each work w_k in vocabulary
       - n_k ← number of work w_k occurs in $Text_j$
       - $P(w_k | v_j) \leftarrow \frac{n_k + 1}{n + |Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT (Doc)

Return the estimated target value for the document Doc, a_i denotes the word found in the ith position within Doc.

- positions ← all word positions in Doc that contains tokens found in vocabulary

- Return v_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} \; P(v_j) \prod_{i \in positions} P(a_i \mid v_j)$$

Program:

```
import pandas as pd
msg=pd.read_csv('6pg.csv', names=['message', 'label'])
print('The dimensions of the dataset', msg.shape)
msg['labelnum'] = msg.label.map({'pos':1, 'neg':0})
x = msg.message
y= msg.labelnum
print(x)
print(y)

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest= train_test_split(x, y)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm= count_vect.transform(xtest)
print(count_vect.get_feature_names())
df = pd.DataFrame(xtrain_dtm.toarray(), columns= count_vect.get_feature_
    names())
print(df)
print(xtrain_dtm)
```

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm, ytrain)
predicted= clf.predict(xtest_dtm)
from sklearn import metrics.
print('Accuracy metrics')
print('Accuracy of the classifier is', metrics.accuracy_score(ytest, predicted))
print('confusion matrix')
print(metrics.confusion_matrix(ytest, predicted))
print('Recall & Precision')
print(metrics.recall_score(ytest, predicted))
print(metrics.precision_score(ytest, predicted))
```

OUTPUT:

```
Accuracy metrics
Accuracy of the classifer is 1.0
Confusion matrix
[[3 0]
 [0 2]]
Recall and Precison
1.0
1.0
PS C:\Users\kindr>
```