Practical No: 9

Backpropagation algorithm and Text pre-processing, Text clustering, classification

AIM: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

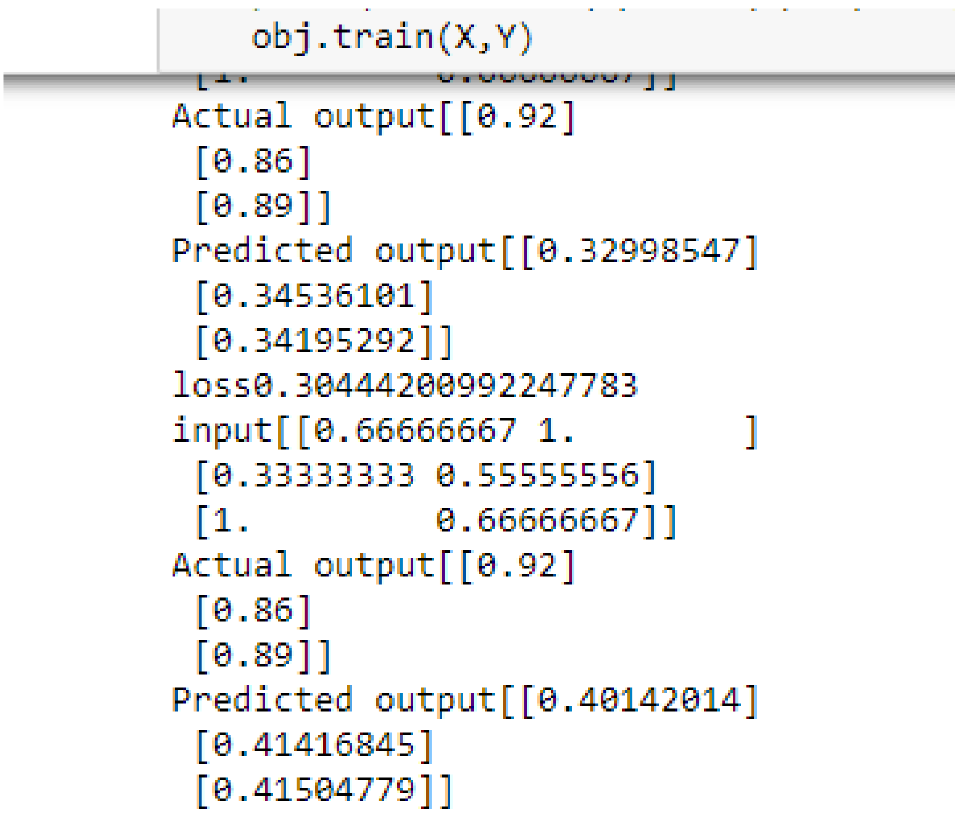
Description:

Artificial Neural Network (ANN) with Backpropagation is a machine learning model designed to mimic the human brain's learning process. By iteratively adjusting connection weights during training, it learns intricate patterns in data, making it capable of making predictions and generalizing from the provided datasets.

Code and output:

```
import numpy as np
X=np.array(([2,9],[1,5],[3,6]),dtype=float)
Y=np.array(([92],[86],[89]),dtype=float)
X=X/np.amax(X,axis=0)
Y=Y/100;

class NN(object):
   def __init__(self):
      self.inputsize=2
      self.outputsize=1
      self.hiddensize=3
      self.W1=np.random.randn(self.inputsize,self.hiddensize)
      self.W2=np.random.randn(self.hiddensize,self.outputsize)
   def forward(self,X):
      self.z=np.dot(X,self.W1)
      self.z2=self.sigmoidal(self.z)
      self.z3=np.dot(self.z2,self.W2)
      op=self.sigmoidal(self.z3)
      return op;
   def sigmoidal(self,s):
      return 1/(1+np.exp(-s))
   def sigmoidalprime(self,s):
      return s* (1-s)
   def backward(self,X,Y,o):
      self.o_error=Y-o
      self.o_delta=self.o_error * self.sigmoidalprime(o)
      self.z2_error=self.o_delta.dot(self.W2.T)
      self.z2_delta=self.z2_error * self.sigmoidalprime(self.z2)
      self.W1 = self.W1 + X.T.dot(self.z2_delta)
      self.W2= self.W2+ self.z2.T.dot(self.o_delta)
   def train(self,X,Y):
      o=self.forward(X)
      self.backward(X,Y,o)
```

```
obj=NN()
for i in range(2000):
    print("input"+str(X))
    print("Actual output"+str(Y))
    print("Predicted output"+str(obj.forward(X)))
    print("loss"+str(np.mean(np.square(Y-obj.forward(X)))))
    obj.train(X,Y)
```



```
    obj.train(X,Y)
Actual output[[0.92]
 [0.86]
 [0.89]]
Predicted output[[0.32998547]
 [0.34536101]
 [0.34195292]]
loss0.30444200992247783
input[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output[[0.92]
 [0.86]
 [0.89]]
Predicted output[[0.40142014]
 [0.41416845]
 [0.41504779]]
```

Learnings:

This code creates a basic computer program that tries to learn patterns from a small set of information. It uses a special math function to make predictions and adjusts itself to get better over 2000 tries. However, it needs some extra details, like how fast it should learn and a small fix to improve its performance. In a nutshell, it's like a beginner's attempt at building a smart program that needs a bit of fine-tuning.

b) AIM: Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix

DESCRIPTION:

1. Text Pre-processing:

Clean and prepare the restaurant review text data by removing non-alphabetic characters, converting to lowercase, stemming, and eliminating common English stopwords, ensuring the dataset is ready for analysis.

2. Text Clustering:

Utilize a Bag of Words model with CountVectorizer to transform the pre-processed text data into numerical features, enabling the application of clustering algorithms to group similar reviews together and identify patterns within the dataset.

3. Classification with Prediction:

Train a Gaussian Naive Bayes classifier on the pre-processed and transformed data to predict sentiment labels (positive or negative) for restaurant reviews, allowing the model to learn from the training set and make predictions on unseen data.

4. Test Score:

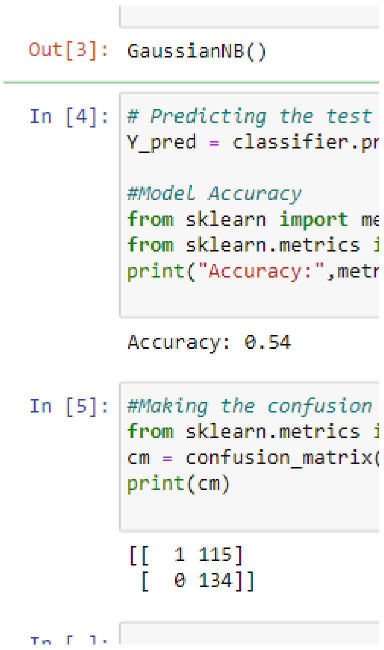
Evaluate the performance of the Naive Bayes classifier by calculating accuracy scores, using metrics such as accuracy_score to measure the model's effectiveness in correctly predicting sentiments on the test set.

5. Confusion Matrix:

Generate a confusion matrix to provide a detailed breakdown of the model's predictions, showcasing true positive, true negative, false positive, and false negative results. This matrix offers insights into the classifier's strengths and weaknesses in sentiment classification.

Code and output:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)

import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = []
for i in range(0,1000):
  review = re.sub('[^a-zA-Z]','',dataset['Review'][i])
  review = review.lower()
  review = review.split()
  ps = PorterStemmer()
  review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
  review = ''.join(review)
  corpus.append(review)
#Creating the bag of words model
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()
Y = dataset.iloc[:,1].values

#Splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.25, random_state=100)


#Fitting naive bayes to the training set.
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
# Predicting the test set results.
Y_pred = classifier.predict(X_test)

#Model Accuracy
from sklearn import metrics
from sklearn.metrics import confusion_matrix
print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))
#Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```



```
Out[3]: GaussianNB()

In [4]: # Predicting the test
        Y_pred = classifier.pr

        #Model Accuracy
        from sklearn import me
        from sklearn.metrics i
        print("Accuracy:",metr

        Accuracy: 0.54

In [5]: #Making the confusion
        from sklearn.metrics i
        cm = confusion_matrix(
        print(cm)

        [[  1 115]
         [  0 134]]

In [ ]:
```

Learnings:

This code performs sentiment analysis on restaurant reviews using a Naive Bayes classifier. Initially, the dataset is loaded and pre-processed, including removing non-alphabetic characters, converting text to lowercase, and stemming words. The Bag of Words model is then implemented using the Count Vectorizer to transform the text data into numerical features. The dataset is split into training and testing sets, with 75% for training and 25% for testing. A Gaussian Naive Bayes classifier is trained on the training set and used to predict sentiments on the test set. The model's accuracy is evaluated using metrics like accuracy score and a confusion matrix, providing insights into the classifier's performance. This approach leverages natural language processing techniques, including text cleaning and machine learning, to classify reviews as positive or negative based on the words used. The Naive Bayes model proves useful for its simplicity and effectiveness in handling textual data.