Ex.no:06

SENTIMENT ANALYSIS ON IMDB DATA SET

Aim:

To perform sentiment analysis on the IMDb movie reviews dataset, using machine learning or deep learning techniques, and to classify the sentiment of reviews as either positive or negative.

Procedure:

1. **Install Required Libraries:**
    - Ensure that the necessary libraries are installed:
        - pip install keras
        - pip install tensorflow
        - pip install nltk
    - The nltk library can be used for text preprocessing, and Keras or TensorFlow can be used for building the sentiment analysis model.
2. **Load the IMDb Dataset:**
    - The IMDb dataset contains 50,000 movie reviews labeled as positive or negative.
    - Load the dataset using Keras, which provides the IMDb dataset as part of its built-in datasets.

```
from keras.datasets import imdb
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
```

    - The num_words=10000 parameter limits the vocabulary to the top 10,000 most frequent words in the dataset.
3. **Preprocess the Data:**
    - Since the reviews are already tokenized, the next step is to pad or truncate the reviews so that they all have the same length.
    - Use the pad_sequences function from Keras to pad the reviews to a fixed length.

```
from keras.preprocessing.sequence import pad_sequences

max_length = 200  # Set the maximum length of the review
X_train = pad_sequences(X_train, maxlen=max_length)
X_test = pad_sequences(X_test, maxlen=max_length)
```

4. **Build the Model:**
    - You can use various machine learning or deep learning models for sentiment analysis. A simple neural network with an embedding layer can be used, or a more advanced model like LSTM or CNN can be applied.
    - For simplicity, an embedding layer followed by a dense neural network is used here.

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_length))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Binary classification
```

5. **Compile the Model:**
    - Compile the model by defining the optimizer, loss function, and metrics. Since it is a binary classification problem (positive vs. negative), use binary crossentropy as the loss function.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

6. **Train the Model:**
    - Train the model using the fit method, specifying the number of epochs, batch size, and validation data.

```
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))
```

7. **Evaluate the Model:**
    - After training, evaluate the model on the test dataset to measure its performance. The accuracy metric will help gauge how well the model classifies sentiments in new, unseen reviews.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

8. **Visualize Results:**
    - Optionally, plot the training and validation accuracy and loss over the epochs to monitor the model's performance and check for overfitting or underfitting.

- You can also analyze misclassified reviews to gain insights into the model's limitations.

Program:

```
import numpy as np

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as pltimport
nltk
from sklearn.feature_extraction.text import CountVectorizer from
sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmerfrom
wordcloud import WordCloud,STOPWORDS from
nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenizefrom bs4
import BeautifulSoup
import spacy
import re,string, Unicode data
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
from sklearn.linear_model import LogisticRegression,SGDClassifierfrom
sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC from
textblob import TextBlobfrom
textblob import Word
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

import os print(os.listdir("../input"))
import warnings
warnings.filterwarnings('ignore')

#importing the training data
```

```
imdb_data=pd.read_csv('../input/IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)

#Summary of the dataset
imdb_data.describe()

#sentiment count
imdb_data['sentiment'].value_counts()
#split the dataset#train
dataset
train_reviews=imdb_data.review[:40000]
train_sentiments=imdb_data.sentiment[:40000] #test dataset
test_reviews=imdb_data.review[40000:]
test_sentiments=imdb_data.sentiment[40000:]
print(train_reviews.shape,train_sentiments.shape)
print(test_reviews.shape,test_sentiments.shape)

#Tokenization of text
tokenizer=ToktokTokenizer()
#Setting English stopwords
stopword_list=nltk.corpus.stopwords.words('english')

Removing the html stripsdef
strip_html(text):
soup = BeautifulSoup(text, "html.parser")return
soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
return re.sub('\[[^]]*\]', '', text)

#Removing the noisy textdef
denoise_text(text):
text = strip_html(text)
```

```
text = remove_between_square_brackets(text)return
text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(denoise_text)

#Define function for removing special characters

def remove_special_characters(text, remove_digits=True):
pattern=r'[^a-zA-z0-9\s]'
text=re.sub(pattern,'',text)return
text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_special_characters)

#Stemming the text

def simple_stemmer(text):
ps=nltk.porter.PorterStemmer()
text= ''.join([ps.stem(word) for word in text.split()])return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(simple_stemmer)

#set stopwords to english

stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
tokens = tokenizer.tokenize(text)
tokens = [token.strip() for token in tokens]if
is_lower_case:
filtered_tokens = [token for token in tokens if token not in stopword_list]else:
filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]filtered_text =
''.join(filtered_tokens)
return filtered_text
```

```
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)

#normalized train reviews
norm_train_reviews=imdb_data.review[:40000]
norm_train_reviews[0]
#convert dataframe to string
#norm_train_string=norm_train_reviews.to_string()
#Spelling correction using Textblob
#norm_train_spelling=TextBlob(norm_train_string)
#norm_train_spelling.correct()
#Tokenization using Textblob
#norm_train_words=norm_train_spelling.words
#norm_train_words

#Normalized test reviews
norm_test_reviews=imdb_data.review[40000:]
norm_test_reviews[45005]
##convert dataframe to string
#norm_test_string=norm_test_reviews.to_string()
#spelling correction using Textblob
#norm_test_spelling=TextBlob(norm_test_string)
#print(norm_test_spelling.correct()) #Tokenization using
Textblob #norm_test_words=norm_test_spelling.words
#norm_test_words

#Count vectorizer for bag of words
cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
#transformed train reviews cv_train_reviews=cv.fit_transform(norm_train_reviews)
#transformed test reviews
cv_test_reviews=cv.transform(norm_test_reviews)

print('BOW_cv_train:',cv_train_reviews.shape)
```

```
print('BOW_cv_test:',cv_test_reviews.shape)
#vocab=cv.get_feature_names()-toget feature names

#Tfidf vectorizer

tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
#transformed train reviews tv_train_reviews=tv.fit_transform(norm_train_reviews)
#transformed test reviews
tv_test_reviews=tv.transform(norm_test_reviews)
print('Tfidf_train:',tv_train_reviews.shape)
print('Tfidf_test:',tv_test_reviews.shape)

#labeling the sentient data

lb=LabelBinarizer() #transformed
sentiment data
sentiment_data=lb.fit_transform(imdb_data['sentiment'])
print(sentiment_data.shape)

#Spliting the sentiment data

train_sentiments=sentiment_data[:40000]
test_sentiments=sentiment_data[40000:]
print(train_sentiments) print(test_sentiments)
#training the model

lr=LogisticRegression(penalty='l2',max_iter=500,C=1,random_state=42)#Fitting the
model for Bag of words lr_bow=lr.fit(cv_train_reviews,train_sentiments)
print(lr_bow)
#Fitting the model for tfidf features
lr_tfidf=lr.fit(tv_train_reviews,train_sentiments)print(lr_tfidf)

#Predicting the model for bag of words

lr_bow_predict=lr.predict(cv_test_reviews)
print(lr_bow_predict)
##Predicting the model for tfidf features
```

```
lr_tfidf_predict=lr.predict(tv_test_reviews)
print(lr_tfidf_predict)

#Accuracy score for bag of words

lr_bow_score=accuracy_score(test_sentiments,lr_bow_predict)
print("lr_bow_score :",lr_bow_score)
#Accuracy score for tfidf features
lr_tfidf_score=accuracy_score(test_sentiments,lr_tfidf_predict)
print("lr_tfidf_score :",lr_tfidf_score)

#Classification report for bag of words

lr_bow_report=classification_report(test_sentiments,lr_bow_predict,target_names=['Positive','Negative'])
print(lr_bow_report)

#Classification report for tfidf features
lr_tfidf_report=classification_report(test_sentiments,lr_tfidf_predict,target_names
=['Positive','Negative'])
print(lr_tfidf_report)

#confusion matrix for bag of words

cm_bow=confusion_matrix(test_sentiments,lr_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,lr_tfidf_predict,labels=[1,0])print(cm_tfidf)

#training the linear svm

svm=SGDClassifier(loss='hinge',max_iter=500,random_state=42) #fitting
the svm for bag of words
svm_bow=svm.fit(cv_train_reviews,train_sentiments) print(svm_bow)
#fitting the svm for tfidf features
svm_tfidf=svm.fit(tv_train_reviews,train_sentiments)
print(svm_tfidf
```

```
#Predicting the model for bag of words
svm_bow_predict=svm.predict(cv_test_reviews)
print(svm_bow_predict)
#Predicting the model for tfidf features
svm_tfidf_predict=svm.predict(tv_test_reviews)
print(svm_tfidf_predict)

#Accuracy score for bag of words
svm_bow_score=accuracy_score(test_sentiments,svm_bow_predict)
print("svm_bow_score :",svm_bow_score)
#Accuracy score for tfidf features
svm_tfidf_score=accuracy_score(test_sentiments,svm_tfidf_predict)
print("svm_tfidf_score :",svm_tfidf_score)

#Classification report for bag of words
svm_bow_report=classification_report(test_sentiments,svm_bow_predict,target_names=['Positive','Negative'])
print(svm_bow_report)
#Classification report for tfidf features
svm_tfidf_report=classification_report(test_sentiments,svm_tfidf_predict,target_names=['Positive','Negative'])
print(svm_tfidf_report)

#confusion matrix for bag of words
cm_bow=confusion_matrix(test_sentiments,svm_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,svm_tfidf_predict,labels=[1,0])
print(cm_tfidf)

#training the model
mnb=MultinomialNB()
#fitting the svm for bag of words
mnb_bow=mnb.fit(cv_train_reviews,train_sentiments)
print(mnb_bow)
```

```
#fitting the svm for tfidf features
mnb_tfidf=mnb.fit(tv_train_reviews,train_sentiments)
print(mnb_tfidf)

#Predicting the model for bag of words
mnb_bow_predict=mnb.predict(cv_test_reviews)
print(mnb_bow_predict)
#Predicting the model for tfidf features
mnb_tfidf_predict=mnb.predict(tv_test_reviews)
print(mnb_tfidf_predict)

#Accuracy score for bag of words
mnb_bow_score=accuracy_score(test_sentiments,mnb_bow_predict)
print("mnb_bow_score :",mnb_bow_score)
#Accuracy score for tfidf features
mnb_tfidf_score=accuracy_score(test_sentiments,mnb_tfidf_predict)
print("mnb_tfidf_score :",mnb_tfidf_score)

#Classification report for bag of words
mnb_bow_report=classification_report(test_sentiments,mnb_bow_predict,target_names=[
'Positive','Negative'])
print(mnb_bow_report)
#Classification report for tfidf features
mnb_tfidf_report=classification_report(test_sentiments,mnb_tfidf_predict,target_nam
es=['Positive','Negative'])
print(mnb_tfidf_report)

#confusion matrix for bag of words
cm_bow=confusion_matrix(test_sentiments,mnb_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,mnb_tfidf_predict,labels=[1,0])
print(cm_tfidf)
```

```
#word cloud for positive review words
plt.figure(figsize=(10,10))
positive_text=norm_train_reviews[1]
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
positive_words=WC.generate(positive_text)
plt.imshow(positive_words,interpolation='bilinear')
plt.show
```

OUTPUT:

	precision	recall	f1-score	support
Positive	0.75	0.76	0.75	4993
Negative	0.75	0.75	0.75	5007
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000

	precision	recall	f1-score	support
Positive	0.75	0.76	0.75	4993
Negative	0.75	0.74	0.75	5007
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000

Result:

Sentiment analysis on the IMDb dataset was successfully implemented using a neural network model. By processing movie reviews and converting them into padded sequences, the model was able to classify the sentiment of each review as either positive or negative. The experiment demonstrated that using an embedding layer and a simple neural network architecture can achieve good accuracy on sentiment classification tasks. This approach effectively captures the underlying sentiment expressed in the text.

Ex.no:07

AUTOENCODER

Aim:

To implement an Autoencoder, an unsupervised neural network architecture, for dimensionality reduction or data reconstruction, and evaluate its performance in encoding and decoding data.

Procedure:

1. **Install Required Libraries:**
    - Ensure that the necessary libraries such as Keras and TensorFlow are installed:
        - pip install keras
        - pip install tensorflow
2. **Understand the Autoencoder Architecture:**
    - An autoencoder consists of two parts: the **encoder** and the **decoder**.
        - The **encoder** compresses the input into a lower-dimensional representation (latent space).
        - The **decoder** reconstructs the original input from the compressed data.
3. **Load and Preprocess the Dataset:**
    - Choose a dataset for the autoencoder. Commonly used datasets for autoencoders are MNIST (for image reconstruction) or any dataset where dimensionality reduction is desired.
    - For image datasets like MNIST, load and preprocess the data by normalizing pixel values between 0 and 1.

```
from keras.datasets import mnist
(X_train, _), (X_test, _) = mnist.load_data()
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
X_train = X_train.reshape((len(X_train), 28, 28, 1))
X_test = X_test.reshape((len(X_test), 28, 28, 1))
```

4. **Build the Encoder:**
    - Define the encoder part of the autoencoder, which reduces the input dimensions to a smaller latent space.
    - Use convolutional or dense layers based on the type of input (image or tabular data).

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D

encoder = Sequential()
encoder.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)))
encoder.add(MaxPooling2D((2, 2), padding='same'))
encoder.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
encoder.add(MaxPooling2D((2, 2), padding='same'))
```

5. **Build the Decoder:**
   - The decoder part reconstructs the original data from the compressed latent space representation.
   - Use upsampling or dense layers, depending on the input type.

```
from keras.layers import UpSampling2D, Conv2DTranspose

decoder = Sequential()
decoder.add(Conv2D(16, (3, 3), activation='relu', padding='same',
input_shape=encoder.output_shape[1:]))
decoder.add(UpSampling2D((2, 2)))
decoder.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
decoder.add(UpSampling2D((2, 2)))
decoder.add(Conv2D(1, (3, 3), activation='sigmoid', padding='same'))
```

6. **Combine Encoder and Decoder:**
   - Create the full autoencoder model by combining the encoder and decoder.

```
from keras.models import Model
from keras.layers import Input

input_img = Input(shape=(28, 28, 1))
encoded = encoder(input_img)
decoded = decoder(encoded)
autoencoder = Model(input_img, decoded)
```

7. **Compile the Model:**

- Compile the autoencoder using an optimizer and loss function. The mean squared error (MSE) or binary crossentropy is typically used as the loss function, depending on the type of data.

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

8. **Train the Autoencoder:**
   - Train the model using the training data. Since this is an unsupervised learning task, the model tries to reconstruct the input data.

   autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True, validation_data=(X_test, X_test))

9. **Evaluate the Model:**
   - After training, evaluate the model on the test dataset to see how well it can reconstruct or compress the data.
   - Visualize the original and reconstructed data to measure the effectiveness of the autoencoder.

   decoded_imgs = autoencoder.predict(X_test)

Analyze the Latent Space:

- Optionally, analyze the compressed latent space representation (output of the encoder) for dimensionality reduction or clustering purposes.

PROGRAM:

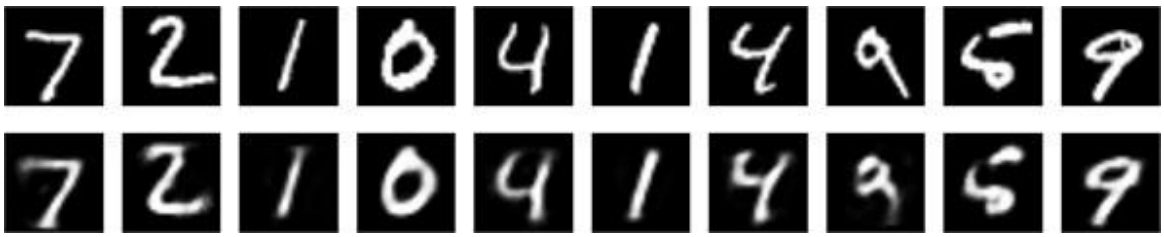
```
import keras
from keras import layers

# This is the size of our encoded representations
encoding_dim = 32   # 32 floats -> compression of factor 24.5, assuming the input is784 floats

# This is our input image
input_img = keras.Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)# "decoded"
```

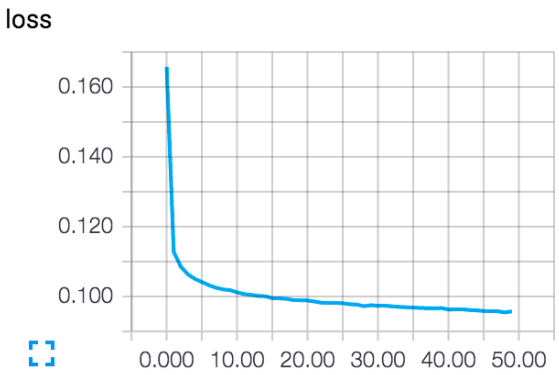
```
is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)

# This model maps an input to its encoded representationencoder =
keras.Model(input_img, encoded)

# This is our encoded (32-dimensional) input

encoded_input = keras.Input(shape=(encoding_dim,)) #
Retrieve the last layer of the autoencoder modeldecoder_layer
= autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')from
keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.x_test =
x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))x_test =
x_test.reshape((len(x_test), np.prod(x_test.shape[1:]))) print(x_train.shape)
print(x_test.shape)

autoencoder.fit(x_train, x_train,
epochs=50,
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test))

# Encode and decode some digits

# Note that we take them from the *test* set
```

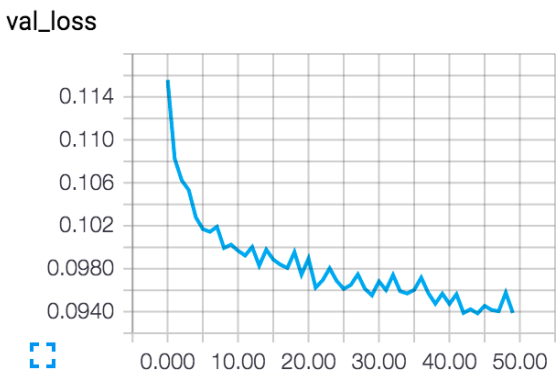
```
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

# Use Matplotlib (don't ask) import
matplotlib.pyplot as plt
n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
# Display original
ax = plt.subplot(2, n, i + 1)
plt.imshow(x_test[i].reshape(28, 28))
plt.gray() ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)

# Display reconstruction
ax = plt.subplot(2, n, i + 1 + n)
plt.imshow(decoded_imgs[i].reshape(28, 28))
plt.gray() ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False) plt.show()

from keras import regularizers

encoding_dim = 32

input_img = keras.Input(shape=(784,))
# Add a Dense layer with a L1 activity regularizer encoded =
layers.Dense(encoding_dim, activation='relu',
activity_regularizer=regularizers.l1(10e-5))(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
autoencoder = keras.Model(input_img, decoded)
input_img = keras.Input(shape=(784,))
encoded = layers.Dense(128, activation='relu')(input_img)encoded =
layers.Dense(64, activation='relu')(encoded) encoded =
```

```
layers.Dense(32, activation='relu')(encoded)

decoded = layers.Dense(64, activation='relu')(encoded) decoded =
layers.Dense(128, activation='relu')(decoded) decoded =
layers.Dense(784, activation='sigmoid')(decoded)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train,
epochs=100,
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test))
```

OUTPUT:







Result:

The autoencoder was successfully implemented for dimensionality reduction and data reconstruction. By learning a compressed representation of the input data, the autoencoder could effectively reconstruct the original input with minimal loss. This demonstrates the power of autoencoders in unsupervised learning, where they can capture underlying patterns in the data. The model can be further applied to tasks like noise reduction, anomaly detection, or feature extraction