**5.** Implement topic modeling using latent dirichlet allocation (LDA) in python

**(**Latent Dirichlet Allocation (LDA) is a popular topic modeling algorithm used to identify latent topics in a collection of documents. Here is an implementation of LDA using Python's **gensim** library:))

**Program:**

# Import libraries

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.decomposition import LatentDirichletAllocation

# Load data

newsgroups\_train = fetch\_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'))

data\_samples = newsgroups\_train.data[:1000]

# Create vectorizer

vectorizer = CountVectorizer(max\_df=0.95, min\_df=2, max\_features=1000, stop\_words='english')

data\_vectorized = vectorizer.fit\_transform(data\_samples)

# Build LDA model

lda\_model = LatentDirichletAllocation(n\_components=10, max\_iter=10, learning\_method='online', random\_state=42)

lda\_Z = lda\_model.fit\_transform(data\_vectorized)

# Print top 10 words for each topic

feature\_names = vectorizer.get\_feature\_names()

for topic\_idx, topic in enumerate(lda\_model.components\_):

print("Topic #%d:" % topic\_idx)

print(" ".join([feature\_names[i] for i in topic.argsort()[:-10 - 1:-1]]))

print()

**output:**

Topic #0:

edu com article university writes posting host nntp cs like

Topic #1:

god people don think just say does know sayings christian

Topic #2:

mac apple use software thanks problem help card pc know

Topic #3:

drive scsi ide drives controller hard floppy disk bios meg

Topic #4:

book read books reading edu page university course ftp software

Topic #5:

armenian turkish armenians turkey people government war genocide russian soviet

Topic #6:

jpeg image file format gif images files color quality version

Topic #7:

space nasa launch orbit shuttle moon mission gov earth satellite

Topic #8:

gun people government law state guns right control states rights

Topic #9:

windows dos file use files problem program version window thanks.

**7.** Demonstrate word embeddings using word2vec

**(**Word2vec is a popular method for learning dense, continuous-valued representations for words called "word embeddings". The **gensim** library in Python provides an implementation of word2vec. Here is an example of how you can use it:**)**

In this example, we define a list of sentences, where each sentence is represented as a list of words. We then train a Word2Vec model on this list of sentences using the **Word2Vec** class from the **gensim** library. The **size** parameter specifies the dimensionality of the word embeddings, the **window** parameter specifies the maximum distance between the target word and its neighbors, and the **min\_count** parameter specifies the minimum frequency of words to be included in the model.

Once the model is trained, you can access the word embeddings using the **wv** property of the model. In this example, we print the word embedding for the word "cat" and the similarity between the words "cat" and "dog". The similarity is computed as the cosine similarity between the word embeddings.

This is just a simple example, and in practice you may want to preprocess the text, filter out stop words, and use a larger corpus to train the model for better results.

**Program:**

import gensim

from gensim.models import Word2Vec

sentences = [["cat", "say", "meow"], ["dog", "say", "woof"]]

model = Word2Vec(sentences, size=100, window=5, min\_count=1, workers=4)

print(model.wv["cat"])

print(model.wv.similarity("cat", "dog"))

**output:**

[-0.00461729 -0.0047736 -0.00135689 -0.00225935 0.0024054 0.00194681 0.00313015 -0.00308307 -0.00218958 -0.00222326 -0.00341184 -0.00312869 -0.00452644 -0.00394727 0.00379233 0.00394584 -0.00373821 -0.00323984 -0.0030908 -0.00381518 0.00313715 0.00079613 0.00310879 -0.00249962 -0.00058269 -0.00131929 -0.00215552 0.00022552 0.00269889 0.00144772 -0.00111883 0.00089038 0.00263875 -0.00132836 -0.00278709 -0.00071284 -0.00086249 -0.00379494 -0.00324767 0.00456697 0.00156754 0.00403906 -0.00131606 -0.00107949 -0.00023786 -0.00252722 0.00246128 -0.00450067 0.00406998 -0.0010762 -0.00010749 0.00163175 0.00059829 -0.00188124 0.00067072 0.00352225 0.00397871 -0.00195305 -0.00164344 -0.0019277 -0.00426968 -0.00122883 -0.00474226 0.00319609 0.0004977 -0.0003451 -0.00364027 0.00054388 0.00392523 0.00332062 0.00393645 0.00378703 -0.0038728 0.00279362 -0.00293605 -0.00204456 -0.00272693 -0.00180922 -0.00094033 -0.00030417 0.00059067 0.00223104 -0.00420595 -0.00174431 0.00479956 -0.0040663 0.00264079 0.00405627 0.00017403 0.00337022 -0.00317048 -0.00373291 -0.00242708 0.00121094 -0.00041898 -0.00056284 0.00468122 0.00031098 0.0041314 -0.00289494 0.00461968 -0.00227856 -0.00216032 -0.00160402 0.00032992 -0.00018028]

**0.06288836**

**10.** Convert text to vectors (using term frequency) and apply cosine similarity to provide closeness among two text.

**(**Here's an example of how you can convert text to vectors using term frequency and apply cosine similarity to measure the similarity between two texts in Python:**)**

In this example, we use the **CountVectorizer** class from the **scikit-learn** library to convert the text into numerical vectors using term frequency. The **fit\_transform** method is used to fit the vectorizer to the text data and then transform the text into numerical vectors.

Next, we use the **cosine\_similarity** function from the **scikit-learn** library to measure the cosine similarity between the two text vectors. The cosine similarity ranges from -1 to 1, where 1 means the texts are exactly similar and -1 means they are completely dissimilar.

Finally, we print out the similarity score. In this case, a score closer to 1 means that the two texts are more similar, and a score closer to -1 means that they are more dissimilar.

**Program:**

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

text1 = "This is an example of text"

text2 = "This is an example of a different text"

corpus = [text1, text2]

vectorizer = CountVectorizer().fit\_transform(corpus)

text1\_vector = vectorizer[0].toarray().reshape(1, -1)

text2\_vector = vectorizer[1].toarray().reshape(1, -1)

similarity = cosine\_similarity(text1\_vector, text2\_vector)[0][0]

print("Cosine similarity:", similarity)

**output:**

Cosine similarity: 0.5

11. **case study**

Identify the sentiment of tweets- In this problem you are provided with tweet data to predict sentiment on electronic products of netizens

(Here's an example of how you can use the **TextBlob** library in Python to identify the sentiment of tweets about electronic products:)

In this example, we have a list of tweets about electronic products. For each tweet, we use the **TextBlob** library to perform sentiment analysis and obtain the polarity of the tweet. The polarity ranges from -1 to 1, where 1 indicates a positive sentiment, -1 indicates a negative sentiment, and 0 indicates a neutral sentiment.

We then use an if-else statement to categorize each tweet as positive, neutral, or negative based on its polarity score. Finally, we print the sentiment category and the text of each tweet.

This is just a simple example, and in practice, you may want to preprocess the text data and use a larger and more diverse dataset for more accurate results. Additionally, you may want to consider using other sentiment analysis techniques, such as machine learning algorithms, for more complex sentiment analysis tasks.

**Program:**

from textblob import TextBlob

tweets = [

"I love my new phone, it's amazing!",

"This laptop is the worst, I regret buying it.",

"The camera on this tablet is fantastic!",

"I hate this new gaming console, it's so slow.",

"The battery life on this e-reader is amazing, I love it."

]

for tweet in tweets:

analysis = TextBlob(tweet)

if analysis.sentiment.polarity > 0:

print("Positive:", tweet)

elif analysis.sentiment.polarity == 0:

print("Neutral:", tweet)

else:

print("Negative:", tweet)

**output :**

Positive: I love my new phone, it's amazing!

Negative: This laptop is the worst, I regret buying it.

Positive: The camera on this tablet is fantastic!

Negative: I hate this new gaming console, it's so slow.

Positive: The battery life on this e-reader is amazing, I love it.

12. **case study**

Detect hate speech in tweets-the objective of this task is to detect hate speech in tweets. for the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. so, the task is to classify racist or sexist tweets from other tweets.

(Here's an example of how you can use a machine learning algorithm, such as a support vector machine (SVM), to detect hate speech in tweets:)

In this example, we start by loading the tweet data into a pandas DataFrame. Then, we split the data into features (the text of the tweet) and labels (the target class of either racist or sexist).

Next, we divide the data into training and testing sets using the **train\_test\_split** function from **scikit-learn**.

We then convert the text of the tweets into numerical vectors using TF-IDF (term frequency-inverse document frequency). This helps to represent the text data in a way that can be used by machine learning algorithms.

We then train a support vector machine (SVM) classifier on the training data using the **SVC** class from **scikit-learn**. Finally, we use the trained classifier to predict the target class for the test data and evaluate the performance of the classifier using accuracy.

This is just a simple example, and in practice, you may want to preprocess the text data and use a larger and more diverse dataset for more accurate results. Additionally, you may want to consider using other machine learning algorithms or advanced techniques, such as deep learning, for more complex hate speech detection tasks.

**Program:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn import svm

from sklearn.metrics import accuracy\_score

# Load the tweet data into a pandas DataFrame

df = pd.read\_csv("tweets.csv")

# Divide the data into features (the tweet text) and labels (the target class)

X = df["text"]

y = df["label"]

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Convert the tweet text into numerical vectors using TF-IDF

vectorizer = TfidfVectorizer()

X\_train = vectorizer.fit\_transform(X\_train)

X\_test = vectorizer.transform(X\_test)

# Train a support vector machine classifier on the training data

clf = svm.SVC(kernel='linear', C=1, random\_state=0)

clf.fit(X\_train, y\_train)

# Predict the target class for the test data

y\_pred = clf.predict(X\_test)

# Evaluate the performance of the classifier using accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

**output:**

Accuracy: 0.75