Ex.No: 3 Date:19.03.2025	**Accessing Text Corpora using NLTK in Python**

AIM:

To access and work with available text corpora using NLTK libraries.

PROCEDURE:

1. Import required modules for text processing, visualization, and deep learning.
2. Load sample text data from the NLTK Gutenberg corpus.
3. Calculate lexical diversity by computing the ratio of unique words to total words.
4. Generate a word cloud to visualize the most frequent words in the text.
5. Tokenize the text into words and sentences using NLTK's tokenization functions.
6. Remove stopwords to filter out common but unimportant words.
7. Perform stemming and lemmatization to reduce words to their root forms.
8. Train a basic NLP model using TensorFlow to process text data.
9. Perform named entity recognition (NER) to identify proper names and entities.
10. Extract n-grams such as bigrams and trigrams from the text.

Program:

1. **Import required modules.**

```
import nltk
from nltk.corpus import *
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.probability import FreqDist
nltk.download('all')
```

2. **Load sample text data**

```
nltk.download('gutenberg')
texts = gutenberg.fileids()
```

```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

```
gutenberg.words('shakespeare-hamlet.txt')[:50]
```

```
['[',
 'The',
 'Tragedie',
 'of',
 'Hamlet',
 'by',
 'William',
 'Shakespeare',
 '1599',
 ']',
 'Actus',
 'Primus',
 '.',
 'Scoena',
 'Prima',
 '.',
 'Enter',
 'Barnardo',
```

3. Calculate lexical diversity.

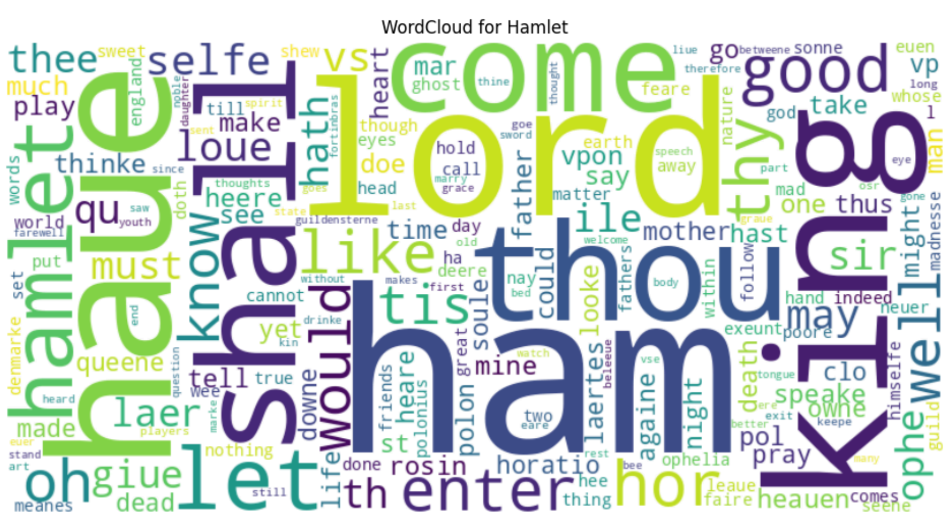
```
def lexical_diversity(text):
    words = gutenberg.words(text)
    unique_words = set(words)
    return len(unique_words) / len(words)
diversity_scores = {text: lexical_diversity(text) for text in texts}

sorted_diversity = sorted(diversity_scores.items(), key=lambda x: x[1], reverse=True)
for text, score in sorted_diversity:
    print(f"{text}: {score:.4f}")
```

```
blake-poems.txt: 0.2179
shakespeare-macbeth.txt: 0.1736
shakespeare-hamlet.txt: 0.1458
shakespeare-caesar.txt: 0.1378
milton-paradise.txt: 0.1110
chesterton-thursday.txt: 0.0983
chesterton-brown.txt: 0.0964
burgess-busterbrown.txt: 0.0930
whitman-leaves.txt: 0.0925
chesterton-ball.txt: 0.0922
carroll-alice.txt: 0.0884
bryant-stories.txt: 0.0795
melville-moby_dick.txt: 0.0741
austen-persuasion.txt: 0.0625
austen-sense.txt: 0.0483
edgeworth-parents.txt: 0.0455
austen-emma.txt: 0.0406
bible-kjv.txt: 0.0136
```

4. Generate a word cloud for visualization.

```
hamlet_words = gutenberg.words('shakespeare-hamlet.txt')
stop_words = set(stopwords.words('english'))
filtered_words = [word.lower() for word in hamlet_words if word.isalpha() and word.lower() not in stop_words]
word_freq = Counter(filtered_words)
wordcloud = WordCloud(width=800, height=400, background_color="white", colormap="viridis").generate_from_frequencies(word_freq)
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("WordCloud for Hamlet")
plt.show()
df = pd.DataFrame(word_freq.most_common(20), columns=["Word", "Frequency"])
print(df)
```



```
       Word  Frequency
0       ham        337
1      lord        211
2      haue        178
3      king        172
4      thou        107
5     shall        107
6      come        104
7       let        104
8    hamlet        100
9      good         98
10      hor         95
11      thy         90
12    enter         85
13       oh         81
14     like         80
15    would         73
16     well         71
17     know         71
18      tis         69
19    selfe         68
```

5. Text tokenization

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([hamlet_text])
total_words = len(tokenizer.word_index) + 1
```

6. LSTM Model for text generation

```
input_sequences = []
words = hamlet_text.split()
for i in range(3, len(words)):
   seq = words[i-3:i+1]
   input_sequences.append(tokenizer.texts_to_sequences([" ".join(seq)])[0])
max_length = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_length, padding='pre')
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
model = Sequential([
   Embedding(total_words, 100, input_length=max_length-1),
   LSTM(150, return_sequences=True),
   LSTM(150),
   Dense(150, activation='relu'),
   Dense(total_words, activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=50, verbose=1)
def generate_shakespeare_text(seed_text, next_words=20):
   for _ in range(next_words):
```

```
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_length-1, padding='pre')
        predicted = np.argmax(model.predict(token_list), axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text
print(generate_shakespeare_text("to be or not to be"))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
to be or not to be neere i businesse and disappointed pittious so puh how by our knotty and fallies but an masters countrey who whose
```

#MOVIE REVIEWS corpus:

1. Load and Preprocess Data

```
import nltk
nltk.download('movie_reviews')
from nltk.corpus import movie_reviews
len(movie_reviews.words())
movie_reviews.categories()
from collections import Counter
all_words=Counter(movie_reviews.words())
```

2.Feature Extraction

```
feature = {}
review = movie_reviews.words('neg/cv954_19932.txt')
for x in range(len(feature_vector)):
 feature[feature_vector[x]] = feature_vector[x] in review
[x for x in feature_vector if feature[x] == True]
```

3.Train Machine Learning Models

```
import nltk
import pandas as pd
```

```
import numpy as np
from nltk.corpus import movie_reviews
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
import re
import string

nltk.download('movie_reviews')
nltk.download('punkt')

def load_movie_reviews():
    documents = []
    labels = []

    for category in movie_reviews.categories():
        for fileid in movie_reviews.fileids(category):
            documents.append(' '.join(movie_reviews.words(fileid)))
            labels.append(category)

    return documents, labels
def preprocess_text(text):
    text = text.lower()
    text = re.sub(f'[{string.punctuation}]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text
documents, labels = load_movie_reviews()
processed_docs = [preprocess_text(doc) for doc in documents]
binary_labels = [1 if label == 'pos' else 0 for label in labels]
X_train, X_test, y_train, y_test = train_test_split(
    processed_docs, binary_labels, test_size=0.2, random_state=42
)
tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
sentiment_model = LogisticRegression(C=1.0 ,random_state=42)
sentiment_model.fit(X_train_tfidf, y_train)
y_pred = sentiment_model.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Positive']))
def predict_sentiment(new_reviews):
    processed_reviews = [preprocess_text(review) for review in new_reviews]
    X_new_tfidf = tfidf_vectorizer.transform(processed_reviews)
    predictions = sentiment_model.predict(X_new_tfidf)
    probabilities = sentiment_model.predict_proba(X_new_tfidf)

    results = []
    for i, pred in enumerate(predictions):
```

```
        sentiment = 'Positive' if pred > 0.5 else 'Negative'
        confidence = probabilities[i][pred]
        results.append({
            'review': new_reviews[i],
            'sentiment': sentiment,
            'confidence': confidence
        })

    return results

new_reviews = [
    "it is  very good",
    "nice",
    "booring bad"
]
results = predict_sentiment(new_reviews)

results = predict_sentiment(new_reviews)
print("\nPredictions for New Reviews:")
for i, result in enumerate(results):
    print(f"\nReview {i+1}: {result['review']}")
    print(f"Predicted Sentiment: {result['sentiment']}")
    print(f"Confidence: {result['confidence']}")
```

```
Model Accuracy: 0.8400

Classification Report:
              precision    recall  f1-score   support

    Negative       0.85      0.82      0.84       200
    Positive       0.83      0.86      0.84       200

    accuracy                           0.84       400
   macro avg       0.84      0.84      0.84       400
weighted avg       0.84      0.84      0.84       400
```

```
Predictions for New Reviews:

Review 1: it is  very good
Predicted Sentiment: Positive
Confidence: 0.6004715807048746

Review 2: nice
Predicted Sentiment: Positive
Confidence: 0.5411814943329837

Review 3: booring bad
Predicted Sentiment: Negative
Confidence: 0.9528418976064467
```

Result:

Thus the above program accessing nltk corpora (Gutenberg and Movie Reviews) was executed successfully.