Aryan Solanki (21BCON392)

Assignment – 02

Section – A

Ans1 – Web Scrapping , Text Corpora or corpus, Crowdsourcing, APIs and Databases.

Ans2 - Lowercasing or uppercasing in NLP ensures text consistency, reduces vocabulary size, simplifies tokenization, and aids in handling named entities, improving text processing accuracy and efficiency.

Ans3 - import re

def remove_url(text):

pattern=re.compile('https?://S+|www\.\S+')

return pattern.sub(r'',text)

Ans4 - Data augmentation diversifies NLP training data by applying transformations like synonym replacement or shuffling, improving model generalization, addressing data imbalance, and aiding domain adaptation and regularization, enhancing overall performance.

Ans5 - "Bigram flipping" is not a standard term in NLP, but if you're referring to "bigram flipping" as a data augmentation technique, it involves randomly swapping adjacent word pairs (bigrams) in a sentence. This technique aims to introduce variability in the text data while preserving overall context and syntactic structure, helping NLP models learn to handle different word arrangements and improving generalization.

SECTION – B

Ans1 - Tokenization in NLP:

1. Definition: Tokenization is the process of breaking down a piece of text into smaller units, known as tokens. These tokens can be words, subwords, or characters, depending on the granularity of tokenization.

2. Word Tokenization: The most common form of tokenization involves splitting text into individual words. For example, the sentence "I love natural language processing" would be tokenized into ["I", "love", "natural", "language", "processing"].

3. Subword Tokenization: In languages with complex morphology or in scenarios with limited vocabulary, subword tokenization breaks words into smaller units, capturing morphological variations and reducing vocabulary size. For instance, "unhappiness" might be tokenized into ["un", "happiness"].

4. Character Tokenization: Tokenizing text at the character level involves breaking down the text into individual characters. For example, the word "hello" would be tokenized into ["h", "e", "l", "l", "o"].

5. Example: Consider the sentence "The quick brown fox jumps over the lazy dog." Word tokenization would yield ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog", "."].

6. Importance: Tokenization is a fundamental preprocessing step in NLP, enabling subsequent analysis such as part-of-speech tagging, named entity recognition, and sentiment analysis. It helps convert raw text data into a format suitable for computational processing.

7. Challenges: Tokenization can be challenging in languages with complex word boundaries, punctuation, or non-standard writing systems. Careful consideration is needed to ensure that tokenization preserves the intended meaning of the text.

Ans2 - The key stages in the pipeline are as follows:

• Data acquisition

• Text cleaning

• Pre-processing

• Feature engineering

• Modelling

• Evaluation

• Deployment

• Monitoring and model updating

Ans3 - In NLP pipelines for tasks like text classification and sequence labeling, machine learning models such as Support Vector Machines (SVM) and Recurrent Neural Networks (RNN) are integrated in the following ways:

1. Feature Extraction: Raw text data is preprocessed and transformed into numerical feature vectors that can be fed into machine learning models. This preprocessing may involve techniques such as tokenization, stemming or lemmatization, and vectorization (e.g., TF-IDF or word embeddings).

2. Model Training: The preprocessed feature vectors are used to train the machine learning models. For tasks like text classification, SVMs are popular due to their ability to handle high-dimensional feature spaces and binary classification problems effectively. RNNs, on the other hand, are well-suited for sequence labeling tasks where input sequences have variable lengths, such as part-of-speech tagging or named entity recognition.

3. Hyperparameter Tuning: Both SVMs and RNNs have hyperparameters that need to be tuned for optimal performance. Techniques like grid search or random search are commonly used to find the best combination of hyperparameters.

4. Model Evaluation: Once trained, the models are evaluated on a held-out validation set or through cross-validation to assess their performance metrics such as accuracy, precision, recall, or F1-score.

5. Deployment: After selecting the best-performing model, it is deployed into the production environment where it can be used to make predictions on new, unseen data. This may involve setting up APIs or other interfaces for integrating the model into applications or systems.

6. Monitoring and Maintenance: Continuous monitoring of the deployed model's performance is crucial to ensure its effectiveness over time. Periodic retraining and updating of the model may be necessary to adapt to changing data distributions or requirements.

SECTION – C

Ans1- import nltk

nltk.download("stopwords")

pip install nltk

from nltk.corpus import stopwords

stopwords.words('english')

def remove_stopwords(text):

new_text = []

for word in text.split():

if word in stopwords.words('english'):

new_text.append('')

else:

new_text.append(word)

x = new_text[:]

new_text.clear()

return " ".join(x)

Ans2 -

import nltk

from nltk.stem import PorterStemmer

Download the Porter Stemmer algorithm if not already downloaded

nltk.download('punkt')

Create a Porter stemmer object

porter_stemmer = PorterStemmer()

List of words to be stemmed

words = ["running", "ran", "runs", "runner", "easily", "fairly", "fairness"]

Perform stemming on each word

stemmed_words = [porter_stemmer.stem(word) for word in words]

Print the original words and their stemmed forms

for word, stemmed_word in zip(words, stemmed_words):

print(f"{word} -> {stemmed_word}")

Ans3 –

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
# Sample text
text = """Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type specimen book."""
# Perform sentence tokenization
sentences = sent_tokenize(text)
# Perform word tokenization for each sentence
word_tokens = [word_tokenize(sentence) for sentence in sentences]
# Print the tokenized sentences
print("Sentence Tokenization:")
for sentence in sentences:
print(sentence)
# Print the word tokens for each sentence
print("\nWord Tokenization:")
for tokens in word_tokens:
print(tokens)
```