Name : Ashuwin P

Reg.No : 3122 22 5002 013

Course : UIT2622 ~ Advanced Artificial Intelligence Techniques

Topic : Language Models ~ Implementation of Bigram model for word prediction

Last Update: 05 November 2024

Aim

The aim of this project is to develop a bigram model for predicting the next word in a sequence of text using natural language processing (NLP) techniques. The model will be built using a corpus of text, which will be preprocessed to generate bigrams and their corresponding probabilities for accurate next-word prediction.

Introduction

Natural Language Processing (NLP) involves the interaction between computers and humans through natural language. One of the essential tasks in NLP is language modeling, which helps in understanding and predicting the next words in a sentence. A bigram model is a type of language model that uses pairs of consecutive words (bigrams) to make predictions. This project focuses on implementing a bigram-based next-word prediction system by processing a given corpus of text.

Overview

The implemented model consists of several key components:

- ***Corpus Reading:*** Reading and loading the text data from a specified file.
- ***Preprocessing:*** Converting the text to lowercase, removing punctuation, and marking the end of sentences.
- ***Tokenization:*** Splitting the text into individual tokens (words).
- ***Frequency Calculation:*** Counting the occurrences of each token and bigram in the text.
- ***Probability Calculation:*** Computing the conditional probabilities of each word following another word using bigrams.

- ***Prediction:*** Using the bigram probabilities to suggest the next word based on the last word entered by the user.

Methodology

The methodology for implementing the bigram model includes the following steps:

1. ***Read the Corpus:*** The text data is read from a file and stored in a string format.
2. ***Preprocess the Text:*** The text is preprocessed to remove punctuation and convert it to lowercase. An end-of-sentence marker is added for clarity.
3. ***Tokenization:*** The preprocessed text is split into tokens for further analysis.
4. ***Frequency Calculation:*** A frequency count of individual tokens and bigrams is performed to facilitate probability calculations.
5. ***Probability Calculation:*** Conditional probabilities are calculated using the frequency of bigrams and individual tokens.
6. ***Next Word Prediction:*** Given an input phrase, the model predicts the next word based on the calculated probabilities.

Formulae

The following formula is used to calculate the conditional probability of a word given the previous word in the bigram model:

$$P(w_2|w_1) = \frac{C(w_1, w_2)}{C(w_1)}$$

where:

- (P(w_2 | w_1)) is the probability of word (w_2) given word (w_1).
- (C(w_1, w_2)) is the count of the bigram ((w_1, w_2)) in the corpus.
- (C(w_1)) is the count of the word (w_1) in the corpus.

The implementation uses the Python programming language and the `collections` module to manage frequencies and probabilities. Key functions include:

- **read_corpus(file_path):** Reads the text data from a specified file path.
- **preprocess(text):** Cleans and prepares the text for tokenization.

- **generate_tokens(text):** Splits the text into individual tokens (words).
- **generate_token_frequencies(tokens):** Counts the frequency of each token in the list of tokens.
- **generate_bigrams(tokens):** Creates a list of bigrams from the tokens.
- **generate_bigram_freq(bigrams):** Calculates the frequency of each bigram.
- **build_bigram_probabilities(bigram_freq, token_freq):** Computes conditional probabilities for bigrams based on token frequencies.
- **predict_next_word(input_text, bigram_probs):** Provides suggestions for the next word based on user input.

```python
from collections import defaultdict
import string

# Function to read text from file
def read_corpus(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()
    return text

# Preprocessing and Tokenizing functions
def preprocess(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Replace period with end-of-sentence marker
    text = text.replace(".", " eos")
    return text

def generate_tokens(text):
    # Split text on whitespace and remove any remaining punctuation
    tokens = text.split()
    return tokens

def generate_token_frequencies(tokens):
    token_freq = {}
    for token in tokens:
        if token in token_freq:
            token_freq[token] += 1
        else:
            token_freq[token] = 1
    return token_freq
```

```python
def generate_bigrams(tokens):
    return [(tokens[i], tokens[i + 1]) for i in range(len(tokens) - 1)]

def generate_bigram_freq(bigrams):
    freq_dict = defaultdict(int)
    for bigram in bigrams:
        freq_dict[" ".join(bigram)] += 1
    return freq_dict



def build_bigram_probabilities(bigram_freq, token_freq):
    bigram_probs = defaultdict(dict)
    for bigram, count in bigram_freq.items():
        word1, word2 = bigram.split()
        if token_freq[word1] > 0:  # Ensure we do not divide by zero
            bigram_probs[word1][word2] = count / token_freq[word1]
    return bigram_probs

def predict_next_word(input_text, bigram_probs):
    tokens = generate_tokens(preprocess(input_text))
    last_word = tokens[-1] if tokens else ""
    if last_word not in bigram_probs:
        return "No suggestions available."

    next_word_candidates = sorted(bigram_probs[last_word].items(),
key=lambda x: x[1], reverse=True)
    suggestions = [word for word, prob in next_word_candidates[:5]]  # Top
5 suggestions
    return suggestions if suggestions else "No suggestions available."

def print_some(dictionary, n):
    for i, (key, value) in enumerate(dictionary.items()):
        if i < n:
            print(f"{key}: {value}")
        else:
            break

file_path = r"D:\SEM5\AAIT\Practical\Bigram_NLP\NLP_Corpus.txt"
corpus_text = read_corpus(file_path)

preprocessed_text = preprocess(corpus_text)

tokens = generate_tokens(preprocessed_text)

print(tokens[:20])
```

```
['natural', 'language', 'processing', 'nlp', 'defined', 'natural',
'language', 'processing', 'nlp', 'is', 'a', 'branch', 'of', 'artificial',
'intelligence', 'ai', 'that', 'enables', 'computers', 'to']
```

```
token_freq = generate_token_frequencies(tokens)

print_some(token_freq, 10)
```

```
natural: 14
language: 20
processing: 15
nlp: 55
defined: 1
is: 56
a: 74
branch: 2
of: 84
artificial: 2
```

```
bigrams = generate_bigrams(tokens)

print(bigrams[:10])
```

```
[('natural', 'language'), ('language', 'processing'), ('processing',
'nlp'), ('nlp', 'defined'), ('defined', 'natural'), ('natural',
'language'), ('language', 'processing'), ('processing', 'nlp'), ('nlp',
'is'), ('is', 'a')]
```

```
bigram_freq = generate_bigram_freq(bigrams)

print_some(bigram_freq, 10)
```

```
natural language: 13
language processing: 8
processing nlp: 3
nlp defined: 1
defined natural: 1
nlp is: 6
is a: 16
a branch: 2
branch of: 2
of artificial: 2
```

```
# Generate probability table for bigram model
bigram_probs = build_bigram_probabilities(bigram_freq, token_freq)

print_some(bigram_probs, 3)
```

```
natural: {'language': 0.9285714285714286, 'sentences':
0.07142857142857142}
language: {'processing': 0.4, 'natural': 0.05, 'text': 0.05, 'nlp': 0.05,
'understanding': 0.05, 'generation': 0.05, 'respectively': 0.05, 'while':
0.05, 'research': 0.05, 'from': 0.05, 'of': 0.05, 'typically': 0.05, 'is':
0.05}
```

```
processing: {'nlp': 0.2, 'has': 0.06666666666666667, 'automate':
0.06666666666666667, 'it': 0.06666666666666667, 'pipeline':
0.06666666666666667, 'they': 0.13333333333333333, 'time':
0.06666666666666667, 'large': 0.06666666666666667, 'library':
0.06666666666666667, 'pipelines': 0.06666666666666667, 'is':
0.06666666666666667, 'will': 0.06666666666666667}
```

```
# Example usage
input_text = input("Type a phrase: ")
suggestions = predict_next_word(input_text, bigram_probs)
print("Suggestions for the next word:", suggestions)
```

```
Type a phrase:  NLP

Suggestions for the next word: ['is', 'can', 'models', 'to', 'technology']
```

Conclusion

The bigram model implemented in this project demonstrates the basic principles of next-word prediction using natural language processing techniques. By utilizing bigrams, the model can effectively suggest the next word based on the previous word, thus enabling a simple yet functional language prediction system. This project serves as a foundational step towards understanding more complex language models and their applications in various NLP tasks.