Auto Sentence Complete

PROJECT REPORT

Submitted by

Simon Prakash A (21BSD7047)

Patnala Siva Subramanyam (21BSD7020)

Saleth Mary A (21BSD7048)

Mahammad Omar Afroj (21BSD7052)

Shaik Mahammad Ali (21BSD7012)

Bachelor of Science

In
Data Science

VIT-AP University
Andhra Pradesh

Abstract

Auto sentence completion, an integral component of natural language processing, has witnessed significant advancements due to the rapid evolution of machine learning and AI technologies. This paper provides a comprehensive overview of the latest developments, methodologies, and challenges in the realm of auto sentence completion. It explores various techniques, including neural language models like GPT (Generative Pre-trained Transformer) and LSTM (Long Short-Term Memory), discussing their capabilities and limitations. Moreover, the abstract delves into the implications of auto sentence completion in diverse applications, such as text generation, chatbots, and content creation. It also addresses the ethical considerations and potential future directions in this field. This paper aims to offer a foundational understanding for researchers, practitioners, and enthusiasts interested in the dynamic landscape of auto sentence completion.

"Auto sentence completion, a pivotal component of Natural Language Processing (NLP), aims to predict and generate the most probable next words or phrases within a given context. This abstract explores the methodologies, techniques, and models employed in auto sentence completion, emphasizing the significance of language modeling, neural networks, and contextual embeddings in enhancing the accuracy and fluency of predictions. The paper delves into the challenges, advancements, and applications of auto sentence completion, highlighting its critical role in various NLP tasks, such as text generation, predictive typing, and conversational agents."

The ability to predict and complete sentences is a fundamental aspect of language understanding. This paper explores the advancements in auto sentence completion models, focusing on their mechanisms, applications, and the evolution of techniques used in natural language processing (NLP). It delves into the underlying neural network architectures, such as recurrent and transformer-based models, that have significantly contributed to the success of auto sentence completion. Additionally, the paper discusses the diverse applications of these models in various domains, including text generation, predictive typing, and language translation. Furthermore, it examines the ethical considerations and potential biases associated with auto sentence completion models. This review aims to provide a comprehensive understanding of the current landscape of auto sentence completion models, their implications, and potential future directions for research and development.

ACKNOWLEDGEMENTS

It has been a great pleasure to take this project. We tried to build an efficient and optimal model with the help of the learning techniques and provided the domain data that we had. It has definitely been a learning experience for all of us. We are all grateful to our professor – Dr. Manimaran Aridoss for his constant support. This project would not have been completed without his enormous help and worthy experience. And for providing us with a conducive learning environment and the necessary resources to carry out this project. Their teachings and mentorship have been instrumental in expanding my knowledge and understanding of the subject.

Although, this report has been prepared with utmost care and deep-rooted interest, even then I accept the respondent and any imperfection. Additionally, we would like to acknowledge the efforts of the research community whose work and publications have formed the foundation of our project. Their contributions in the areas of query suggestion and image classification have been pivotal in shaping my understanding and implementation.

Last but not least, we would like to express my appreciation to the participants who contributed to the dataset used in this project. Their voluntary involvement and willingness to share their data have significantly enhanced the accuracy and effectiveness of the project's outcomes.

To everyone mentioned above and those who have supported me in various ways, we offer my sincerest gratitude. This project would not have been possible without your valuable contributions. Thank you for being a part of this journey and helping us achieve our goals.

Last but not least, we would like to express my appreciation to the participants who contributed to the dataset used in this project. Their voluntary involvement and willingness to share their data have significantly enhanced the accuracy and effectiveness of the project's outcomes.
To everyone mentioned above and those who have supported me in various ways, we offer my sincerest gratitude. This project would not have been possible without your valuable contributions. Thank you for being a part of this journey and helping us achieve our goals.

TEAM MEMBERS :

SALETH MARY A	>	**WEB SCRAPING THE DATA**
MAHAMMAD OMAR AFROJ	>	**DATA PREPROCESSING**
SIMON PRAKASH A	>	**IMPLEMENTATION OF CODE**
SHAIK.MAHAMMAD ALI	>	**INTERFACE**
PATNALA SIVA SUBRAMANYAM	>	**DOCUMENTATION**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of this project was to be able to apply techniques and methods learned in Natural Language Processing course to a rather famous real-world problem, the task of sentence completion using text prediction. The project aims at implementing and analyzing techniques like n-grams, Cosine Similarity Analysis and Pointwise Mutual Information and compare their accuracy and efficiency on the Data

1.2 Overview of the Problem

Text prediction algorithms for automatic or semi-automatic sentence completion have been vastlydiscussed in literature. They are widely used to enhance the speed of communication as well as reducingthe total time taken to compose text. They have applications in search engines and mobile devices so as to reduce the efforts of the end user.

One of the oldest software applications was Profet in 1987. Profet allowed to predict a partially typed word by means of a unigram frequency list: the nine best matches were shown to the user. It also suggested the next word by using a bigram frequency list; however, upon partially typing of the next word,Profet reverted to unigrams-based suggestions.
Word prediction generally relies on n-grams occurrence statistics, which may have huge data storage requirements and does not take into account the general meaning of the text.

1.3 Relation to the Class

The main purpose behind picking up this problem is that it allows us to work on and implement a big array of topics we've learnt as a part of the curriculum for COEN 296. All the approaches that we have used we're based on the material we have read and understood in the class and we wanted to be able to visualize the practical applications and implications of them.

1.4 Problem statement

This problem statement is a part of the Microsoft Sentence Completion Challenge and we plan to use the dataset provided by the challenge. The task is to complete the sentence from the given 5 options. The dataset is a part of Project Gutenberg, which contains around 50000 free ebooks. Challenge of our problem is to achieve Word prediction for sentence completion. For which we targeted Microsoft Research Sentence Completion Challenge dataset which consist of 1040 questions. Each question consistsin a sentence having a missing word and five possible words as valid answers. This challenge includes a training dataset composed by 522 books from Project Gutenberg.

1.5 Hypothesis

Initially, we plan to predict the words in the sentences using n-gram models. To improve on the n-gram model, we will implement smoothing techniques like Good-Turing Smoothing. Then we propose to implement an alternative methodology, based on Latent Semantic Analysis, to address the problem of text completion.

LSA/LSI improves the classic n-grams based approach by considering also terms distant from the word to predict. We also plan to apply Pointwise Mutual Information(PMI) to measure the degree of association between answer options and other sentence tokens for improving the sentence completion challenge by Microsoft.

1.6 Area of Investigation

The project on a broader dimension is based on core application of Natural Language Processing with the focus on the sub-domain of semantic analysis.

CHAPTER 2

THEORETICAL BASIS AND LITERATURE REVIEW

2.1 Definition of problem:

Our project focuses on the problem of sentence level semantic coherence by answering SAT style sentence completion questions. We tackle the problem with three approaches: methods that use local lexical information, such as the n-grams of a classical language model; methods that evaluate global coherence, such as latent semantic analysis and methods that fuses various types of input provided to other classes of language models based on point wise mutual information. We evaluate these methods on a suite of practice SAT questions, and on a recently released sentence completion task based on data taken from five Conan Doyle novels.

2.2 Theoretical background of the problem:

In recent years, standardized examinations have proved a fertile source of evaluation data for language processing tasks. They are valuable for many reasons: they represent facets of language understanding recognized as important by educational experts; they are organized in various formats designed to evaluate specific capabilities; they are yardsticks by which society measures educational progress; and they affect a large number of people.

Narrow and general language processing capabilities has been wildely discussed in previous researches. Among the narrower tasks, the identification of synonyms and antonyms has been studied by (Landauer and Dumais, 1997; Mohammed et al., 2008; Mohammed et al., 2011; Turney et al., 2003; Turney, 2008), who used questions from the Test of English as a Foreign Language (TOEFL), Graduate Record Exams (GRE)and English as a Second Language (ESL) exams. Tasks requiring broader competencies include logic puzzlesand reading comprehension. Logic puzzles drawn from the Law School Administration Test (LSAT) and theGRE were studied in (Lev et al., 2004), which combined an extensive array of techniques to solve the problems. The DeepRead system (Hirschman et al., 1999) initiated a long line of research into reading comprehension based on test prep material (Charniak et al., 2000; Riloff and Thelen, 2000).

In this project, we study a new class of problems intermediate in difficulty between the extremes of synonym detection and general question answering - the sentence completion questions found on the Scholastic Aptitude Test (SAT). These questions present a sentence with one or two blanks that need to be filled in. Five possible words (or short phrases) are given as options for each blank. All possible answers except one result in a nonsense sentence.

The questions are highly constrained in the sense that all the information necessary is present in the sentence itself without any other context. Nevertheless, they vary widely in difficulty. The first of these examples is relatively simple: the second half of the sentence is a clear description of the type of behavior characterized by the desired adjective. The second example is more sophisticated; one must infer from the contrast between medicine and poison that the correct answer involves a contrast, either useless vs. effective or curative vs. toxic. Moreover, the first, incorrect, possibility is perfectly acceptable in the context of the second clause alone; only irrelevance to the contrast between medicine and poison eliminates it. In general, the questions require a combination of semantic and world knowledge as well as occasional logical reasoning. We study the sentence completion task because we believe it is complex enough to pose a significant challenge, yet structured enough that progress may be possible.

2.3 Related research to solve the problem:

The past work which is most similar to ours is derived from the lexical substitution track of SemEval 2007 (McCarthy and Navigli, 2007). In this task, the challenge is to find a replacement for a word or phrase removed from a sentence. In contrast to our SAT-inspired task, the original answer is indicated. For example, one might be asked to find alternates for match in “After the match, replace any remaining fluid deficit to prevent problems of chronic dehydration throughout the tournament.” Two consistently high-performing systems for this task are the KU (Yuret, 2007) and UNT (Hassan et al., 2007) systems. These operate in two phases: first they find a set of potential replacement words, and then they rank them. TheKU system uses just an N-gram language model to do this ranking. The UNT system uses a large variety of information sources, and a language model score receives the highest weight.

2.4 Advantage/Disadvantage of those research:

N-gram statistics were also very effective in (Giuliano et al., 2007). That paper also explores the use of Latent Semantic Analysis to measure the degree of similarity between a potential replacement and its context, but the results are poorer than others. Since the original word provides a strong hint as to the possible meanings of the replacements, we hypothesize that N-gram statistics are largely able to resolve the remaining ambiguities. The SAT sentence completion sentences do not have this property and thus are more challenging.

2.5 Our Solution to this Problem:

This project proposes to analyze LSA (Mikolov, 2013) and Pointwise Mutual Inclusion (Church and Hanks, 1990; Mnih and Kavukcuoglu. 2013) methods which has been explored by researchers in similar context on different problem of linguistic models. We will be measuring accuracy of these methods on the N-gram statistics discussed in previous sections.

We propose an alternative methodology, based on Latent Semantic Analysis, to address these issues. An asymmetric Word-Word frequency matrix is employed to achieve higher scalability with large training datasets than the classic Word-Document approach. We propose a function for scoring candidate terms for the missing word in a sentence. We show how this function approximates the probability of occurrenceof a given candidate word.

Additionally, We would be evaluating a novel approach to automated sentence completion based on Pointwise mutual information (PMI). Feature sets are created by fusing the various types of input providedto other classes of language models, ultimately allowing multiple sources of both local and distant information to be considered. Furthermore, it is shown that additional precision gains may be achieved by incorporating feature sets of higher-order n-grams.

2.6 Where our solution different from others:

Our approaches instead of relying only on N-gram occurrence also consider the historical context of the sentence. Additionally, our approach uses fast computation technique discussed in this paper (A. Mnih, Y. W. Teh, 2012) to address training and computation issue.

2.7 Where our solution is better:

Our solution differs from previous approaches at multiple points. We believe that these new novel approach which has outperformed N-gram models in similar context, will results in improvement in our project as well. Experiments discussed in previously researched paper had shown state-of-the-art performance and accuracy improvement over non-neural linguistic model. We assume the same improvement in our project over previously researched methods.

CHAPTER 3

METHODOLOGY

3.1 Web Scraping the Data using Beautifulsoup

Web scraping is the process of extracting data from various websites and parsing it. In other words, it’s a technique to extract unstructured data and store that data either in a local file or in a database. There are many ways to collect data that involve a huge amount of hard work and consume a lot of time. Web scraping can save programmers many hours.

Beautiful Soup is a Python web scraping library that allows us to parse and scrape HTML and XML pages. You can search, navigate, and modify data using a parser. It’s versatile and saves a lot of time. In this article we will learn how to scrape data using Beautiful Soup.

3.2 Data Pre-Processing

At initial stages of data pre-processing, we read the files into the memory and removed the uninformative header texts followed by converting them to lower case. Next, we tokenized the sentences and then the words using NLTK’s built in methods. The next step was to remove the stop words. Doing this for every file gave us our corpus. And RE, Stopword libraries.

While we were able to run our models with this initial pre-processing, the testing and training were takinga lot of time due to the size of the dataset. To resolve this, we stored the pre-computed data as our modeland saved a significant amount of time for running tests and analysis.

3.3 N-gram Model

This is the most basic and intuitive approach for text prediction. So, for creating a baseline, we will implement the unsmoothed N-gram model. For each sentence, we check if the n-gram, for n = 2,3 and 4, occurs in the training set or not. If it does, we score the sentence with one as a score for bigram, two as a score for trigram and three as a score for 4-gram.

Next, to analyze the improvement in the system we will introduce smoothed n-gram model using Good Turing Smoothing. Smoothing is done to account for the “zero probability N-grams” (A.M. Woods, 2016) which are effectively the N-grams that haven’t appeared in the training corpus but may be perfectly legible natural language phrase. The simplest way to smooth the probabilities is to add one to all counts of theN-grams, known as the Laplace Smoothing, formulated as follows for unigrams:

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$$

where, V are all the words in the vocabulary and since one is added to all counts, we add V to the denominator to account for all the words. This can be extended to N-grams in general, as :

$$P^*_{Laplace}(w_n | w^{n-1}_{n-N+1}) = \frac{C(w^{n-1}_{n-N+1} w_n) + 1}{C(w^{n-1}_{n-N+1}) + V}$$

A little more sophisticated approach for smoothing is the Good Turning Discounting method. The intuition behind GTD is to use the count of things you've seen once to account for things that haven't occurred in the training set yet (D.Jurafsky, J.Martin, 2008). The GTD is formalized as (Jurafsky, Martin, 2008):

$$P^*_{GT}(things\ with\ zero\ frequency\ in\ training) = \frac{N_1}{N}$$

3.4 Generate TFI-DF vectorizer

In information retrieval, tf–idf or TFIDF, full forming as term frequency– inverse document frequency, is nothing but a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The most common use of the tf-idf. The tf–idf value increases in direct proportion to the number of times a term appears in the document and is compensated by the number of documents in the corpus that contain the word, which tends to justify the fact that certain words appear more frequently in general.

TF-IDF weight speaks to the relative significance of a term within the document and whole corpus. TF stands for Term Frequency: It calculates how as often as possible a term appears in a document. Since, each document size varies, a term may show up more in a long-sized document than a brief one. In this way, the length of the document frequently separates Term frequency.

IDF stands for Inverse Document Frequency: When a word appears in all of the records, it is of no use. The words "the," "an," "on," "of," and "a" are just a

few examples. They appear often in a text but are of minor significance. The importance of these terms is reduced by IDF, while the importance of uncommon terms is increased. The more the value of IDF, the more distinct the term becomes.

Term Frequency-Inverse Report Recurrence: TF-IDF works by penalizing the foremost commonly occurring words by allotting them less weightage whereas giving high weightage to terms, which are present within the legitimate subset of the corpus, and has high event in a specific document. Frequency and Inverse Document Frequency is the product.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1 + df}$$

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

3.5 Cosine Similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a *term-frequency vector*. For example, in Table 2.5, we see that *Document1* contains five instances of the word *team*, while *hockey* occurs three times. The word *coach* is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric.

Cosine similarity is a measure used in natural language processing (NLP) to determine the similarity between two vectors representing textual information. It's commonly used in applications such as document similarity analysis, clustering, and information retrieval.

Cosine similarity is widely used in NLP tasks such as text clustering, document retrieval, and recommendation systems because it provides a measure of similarity that is invariant to the scale of the vectors and is well-suited for high-dimensional spaces like those encountered in NLP.

3.6 Interface

interface plays an essential role in delivering an efficient and user-friendly experience. The interface acts as the intermediary between the user and the system, providing the user with relevant suggestions based on the context and user input.

interface serves as the bridge between the auto sentence completion system and the user. It provides the user with suggestions that match their input and help them to communicate more effectively, thereby enhancing the overall user experience.

The tkinter module provides a set of tools for creating GUI elements such as buttons, labels, text boxes, menus, and more, allowing developers to quickly construct an interface for their application. It also provides classes and functions for managing windows, displaying images, handling event-driven programming, etc., making it a versatile toolkit for GUI development.

tkinter module in Python is an excellent choice for developers looking to create desktop applications with GUIs. It offers a range of tools for building user interfaces, is easy to use, and supports cross-platform compatibility.

3.7 Performance Metrics

The performance of each method will be tested based on the accuracy metric, which is measured as follows:

$$Accuracy = \frac{no.of correctpredictions}{totalpredictions}$$

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Implementation Details

All the models have been implemented on Python3 using Jupyter Notebooks and Colaboratory by GoogleResearch. We've used Python libraries like NLTK, Gensim, SciPy and Numpy.

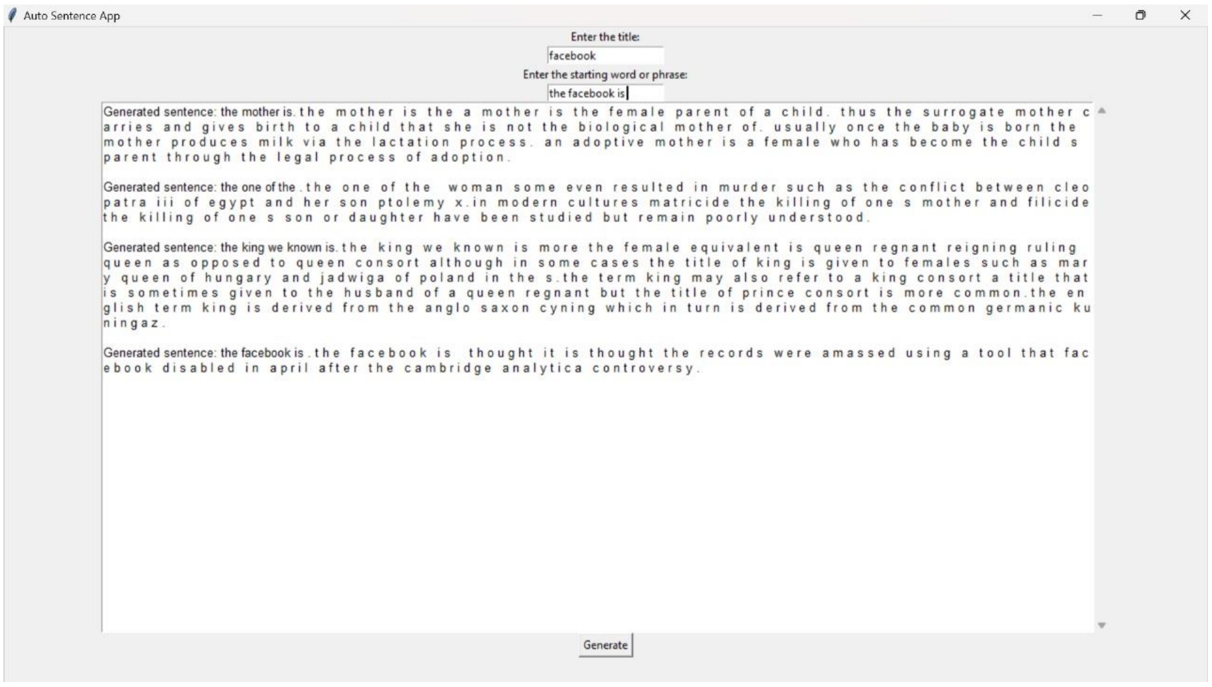
```
import tkinter as tk
from tkinter import scrolledtext  # graphical user interface

import nltk  # Natural language toolkit
nltk.download('punkt')

import re   #regular expression

import random  # generating random integers or String in Python

import requests # Making HTTP requests

from bs4 import BeautifulSoup  #Pulling data out of HTML and XML files

from sklearn.feature_extraction.text import TfidfVectorizer #Convert a
collection of raw documents to a matrix of TF-IDF features

from sklearn.metrics.pairwise import cosine_similarity  #To finding the
similar words



# creating the class for  Auto sentence complete
# Here we use the tkinter package
class AutoSentenceComplete:
    #__init__(self) This self represents the object of the class itself
    def __init__(self, root):
        self.root = root
        self.root.title("Auto Sentence App") # Mention the tittle

        self.title_label = tk.Label(root, text="Enter the title:")
#creating the label for interface
        self.title_label.pack()

        self.title_entry = tk.Entry(root)  #Adding to the main root
        self.title_entry.pack()

        self.text_label = tk.Label(root, text="Enter the starting word or
phrase:") # creating the label for interface
        self.text_label.pack()

        self.text_entry = tk.Entry(root) #Adding to the main root
        self.text_entry.pack()

        self.output_text = scrolledtext.ScrolledText(root, wrap=tk.WORD,
width=150, height=35,font=("Arial",10)) # Scroll box for the output
sentence
        self.output_text.pack()

        # Configure the tag for left alignment
        self.output_text.tag_configure("left", justify='left')

        # Creating the generate button
        self.generate_button = tk.Button(root, text="Generate",
command=self.generate_text)
        self.generate_button.pack()


    #Give the access to the interface
    def generate_text(self):

        topic = self.title_entry.get()  # From the interface collecting the
title input

        given_word = self.text_entry.get()  # From the interface collecting
the given sentence input


        # Function for getting information for our need
        # Using the request and BeautifulSoup

        def web_scrap (x):

            x=re.sub("\s","_",x)

            url="https://en.wikipedia.org/wiki/"+str(x) # Using the
wikipedia

            response= requests.get(url)

            soup = BeautifulSoup(response.text,'html.parser')

            #You're probably treating a list of elements like a single
element
            dataset=[]

            for i in soup.find_all("p"):  # ("p")paragraph in the  html
text

                data=i.get_text()

                dataset.append(data)

            return str(dataset)

        # For join the many dictionary in one dictionary
        def merge(dict1,dict2,dict3,dict4,dict5,dict6):
                max=dict1,dict2,dict3,dict4,dict5,dict6
                for i in max:
                    dict1.update(i)
                return dict1


        #Creating dictionary for n-grams from using the scrapped
information from the web
        def n_grams(token):
                # 1 gram words
                monogram={}
                for i in range(len(token)-1):
                    token1,token2=token[i],token[i+1]
                    gram_key=(token1)
                    if gram_key not in monogram:
                        monogram[gram_key]=[token2]
                    elif token2 not in monogram[gram_key]:
                            monogram[gram_key] += [token2]
                # 2 gram words
                digram={}
                for i in range(len(token)-2):
                    token1,token2,token3=token[i],token[i+1],token[i+2]
                    gram_words=(token1,token2)
                    if gram_words not in digram:
                        digram[gram_words]=[token3]
                    elif token3 not in digram[gram_words]:
                        digram[gram_words] += [token3]

                #3 gram words
                trigram={}
                for i in range(len(token)-3):

token1,token2,token3,token4=token[i],token[i+1],token[i+2],token[i+3]
                    gram_words=(token1,token2,token3)
                    if gram_words not in trigram:
                        trigram[gram_words]=[token4]

                    elif token4 not in trigram[gram_words]:
                        trigram[gram_words]+=[token4]

                # 4 gram words
                fourgram={}
                for i in range(len(token)-4):

token1,token2,token3,token4,token5=token[i],token[i+1],token[i+2],token[i+3
],token[i+4]
                    gram_words=(token1,token2,token3,token4)
                    if gram_words not in fourgram:
                        fourgram[gram_words]=[token5]
                    elif token5 not in fourgram[gram_words]:
                            fourgram[gram_words] += [token5]
                # 5 gram word
                fivegram={}
                for i in range(len(token)-5):

token1,token2,token3,token4,token5,token6=token[i],token[i+1],token[i+2],to
ken[i+3],token[i+4],token[i+5]
                    gram_words=(token1,token2,token3,token4,token5)
                    if gram_words not in fivegram:
                        fivegram[gram_words]=[token6]
                    elif token5 not in fivegram[gram_words]:
                        fivegram[gram_words]+=[token6]

                #Creating the empty dictionary for join every grams
dictionary Using merge function
                grams={}
                merge(grams,monogram,digram,trigram,fourgram,fivegram)
                return grams
```

```
        # For getting the most popular information using the string
similarity
        # Here we using the TfidfVectorizer and cosine_similarity package

        def string_similar(corpus,input_phrase):

                dataset=nltk.sent_tokenize(corpus)

                # Tokenize and preprocess the data
                tfidf_vectorizer = TfidfVectorizer()
                tfidf_matrix = tfidf_vectorizer.fit_transform(dataset)

                # Transform the input phrase into a TF-IDF vector
                input_vector = tfidf_vectorizer.transform([input_phrase])

                # Calculate cosine similarity
                cosine_similarities = cosine_similarity(input_vector,
tfidf_matrix)

                # Create a list of (sentence, similarity) pairs
                similarity_scores = list(enumerate(cosine_similarities[0]))

                # Sort the list by similarity scores in descending order
                sorted_similarities = sorted(similarity_scores, key=lambda
x: x[1], reverse=True)

                # Filter suggestions based on a threshold (e.g., 0.3)
                threshold = 0.3
                filtered_suggestions = [dataset[i] for i, score in
sorted_similarities if score >= threshold]

                return filtered_suggestions


        # Function for prediciting the next word using n-grams
        def next_word(word,grams):
                word=tuple(word)
                if len(word)==1:
                    word=word[0]
                    if word in grams:
                        next_word=grams[word]
                        return next_word
                    else:
                        return []

        # Function for final output sentence
        # Here we are using the random package
        def sentence_complete(input_sent,grams,dataset_sent,final_sent=[]):
            while final_sent==[]:
                    words=input_sent.split()

                    #Reverse the loop number for want input sentence
reversly to check the n-grams
                    for i in range(len(words)-1,-1,-1):
                        last_word=words[i:len(words)]   # It will take the
given sentence from the last to first

                        next_words=next_word(last_word,grams) # It will
check the word whether it is present in the n-grams dictionary
```

```
                        if next_words==[]: # if it is empty than skip
current iliteration
                            continue

                        else:
                            if len(next_words) > 0: #If have any values
then it will work

                                suggestion = random.choice(next_words) # If
it have more than one words than it will choose the one word

                                update_sentence= input_sent + " " +
suggestion


final_sent=string_similar(dataset_sent,update_sentence) # Here it will
check the similarity

                                if len(final_sent) >0:
                                    return update_sentence +" "+"
".join(final_sent) # if have any sentence than it will return

                                input_sent=update_sentence # otherwise it
will take the updated sentence for next iliteration




        # Data preprocessing for model
        # Here re package
        dataset = web_scrap(topic) # Here we using the function for web
scrap

        dataset = dataset.lower()
        dataset_sent = dataset
        dataset = re.sub("\s+", " ", dataset)
        dataset = re.sub("\d+", " ", dataset)
        dataset = re.sub("\W+", " ", dataset)
        dataset = re.sub(" n ", " ", dataset)
        dataset = re.sub("\s+", " ", dataset)

        token = nltk.word_tokenize(dataset) # Tokenizing the dataset

        # It will use for seperate data cleaning for string similarity
        dataset_sent=re.sub('\s+'," ",dataset_sent)
        dataset_sent=re.sub('[.]',"SPACE ",dataset_sent)
        dataset_sent=re.sub('\d+'," ",dataset_sent)
        dataset_sent=re.sub('\W+'," ",dataset_sent)
        dataset_sent=re.sub(' n ',"",dataset_sent)
        dataset_sent=re.sub('SPACE',".",dataset_sent)
        dataset_sent=re.sub('\s+'," ",dataset_sent)

        # Here we use the function n-gram
        grams= n_grams(token)

        # Here we use the function sentence complete
        generated_sentence=sentence_complete(given_word,grams,dataset_sent)

        # Update the output text widget with the generated sentence
        Generated_sentence = "Generated sentence: "+ given_word + ". "+"
".join(generated_sentence)
```

```
        # Add the generated sentence with the 'left' tag for left alignment
        self.output_text.insert(tk.END, Generated_sentence + "\n\n",
"left")
if __name__ == "__main__":  # If we close the interface the loop will stop
    root = tk.Tk()
    app = AutoSentenceComplete(root)
    root.mainloop()
```

4.2 Results



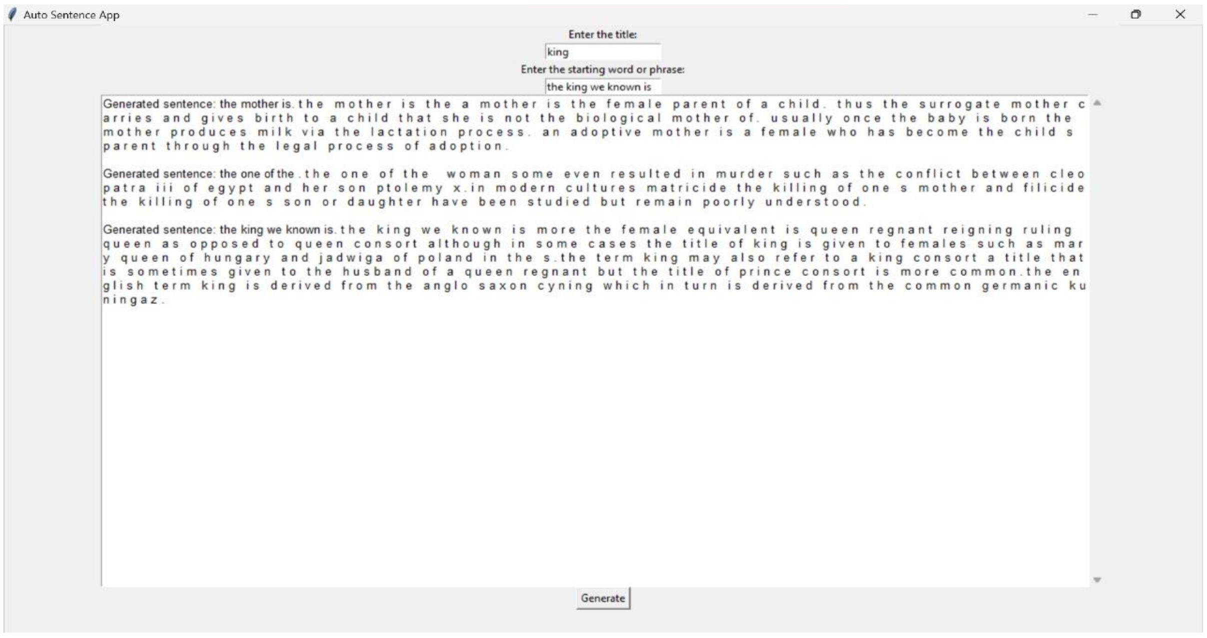
Auto Sentence App

Enter the title: facebook

Enter the starting word or phrase: the facebook is

```
Generated sentence: the mother is. the mother is the a mother is the female parent of a child. thus the surrogate mother c
arries and gives birth to a child that she is not the biological mother of. usually once the baby is born the
mother produces milk via the lactation process. an adoptive mother is a female who has become the child s
parent through the legal process of adoption.

Generated sentence: the one of the . the one of the  woman some even resulted in murder such as the conflict between cleo
patra iii of egypt and her son ptolemy x.in modern cultures matricide the killing of one s mother and filicide
the killing of one s son or daughter have been studied but remain poorly understood.

Generated sentence: the king we known is. the king we known is more the female equivalent is queen regnant reigning ruling
queen as opposed to queen consort although in some cases the title of king is given to females such as mar
y queen of hungary and jadwiga of poland in the s.the term king may also refer to a king consort a title that
is sometimes given to the husband of a queen regnant but the title of prince consort is more common.the en
glish term king is derived from the anglo saxon cyning which in turn is derived from the common germanic ku
ningaz.

Generated sentence: the facebook is . the facebook is  thought it is thought the records were amassed using a tool that fac
ebook disabled in april after the cambridge analytica controversy.
```

Generate



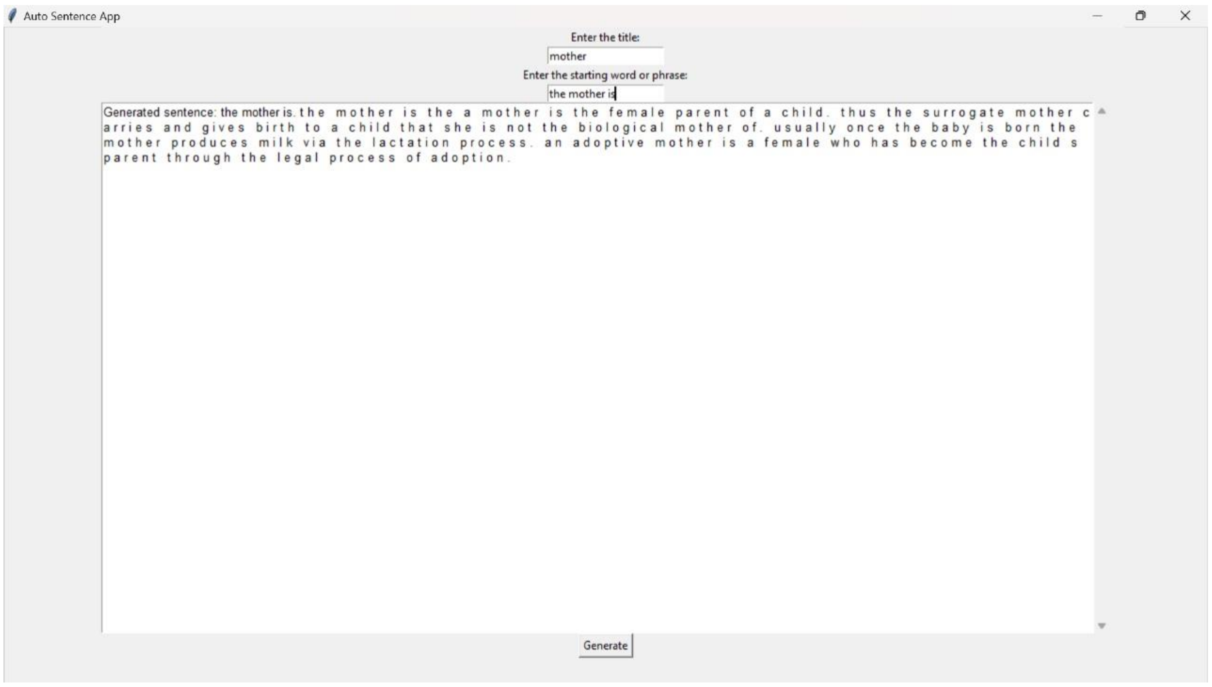
Auto Sentence App

Enter the title: king

Enter the starting word or phrase: the king we known is

```
Generated sentence: the mother is. the mother is the a mother is the female parent of a child. thus the surrogate mother c
arries and gives birth to a child that she is not the biological mother of. usually once the baby is born the
mother produces milk via the lactation process. an adoptive mother is a female who has become the child s
parent through the legal process of adoption.

Generated sentence: the one of the . the one of the  woman some even resulted in murder such as the conflict between cleo
patra iii of egypt and her son ptolemy x.in modern cultures matricide the killing of one s mother and filicide
the killing of one s son or daughter have been studied but remain poorly understood.

Generated sentence: the king we known is. the king we known is more the female equivalent is queen regnant reigning ruling
queen as opposed to queen consort although in some cases the title of king is given to females such as mar
y queen of hungary and jadwiga of poland in the s.the term king may also refer to a king consort a title that
is sometimes given to the husband of a queen regnant but the title of prince consort is more common.the en
glish term king is derived from the anglo saxon cyning which in turn is derived from the common germanic ku
ningaz.
```

Generate



Enter the title: mother

Enter the starting word or phrase: the mother is

```
Generated sentence: the mother is. the mother is the a mother is the female parent of a child. thus the surrogate mother c
arries and gives birth to a child that she is not the biological mother of. usually once the baby is born the
mother produces milk via the lactation process. an adoptive mother is a female who has become the child s
parent through the legal process of adoption.
```

4.3 Result Analysis

Analyzing the results of an auto-sentencing project in natural language processing (NLP) involves a comprehensive assessment of the model's performance, its strengths, and areas for improvement. The process encompasses several key steps to ensure a thorough understanding of the model's capabilities and limitations.

First and foremost, it is essential to establish a robust set of evaluation metrics suitable for the specific auto-sentencing task. Common metrics in NLP, such as precision, recall, F1-score, and accuracy, serve as quantitative measures to gauge the model's performance. These metrics are crucial for providing a quantitative basis for comparing different models and iterations.

The dataset used for training and evaluation is divided into training, validation, and test sets. The training set facilitates the model's learning process, the validation set aids in hyperparameter tuning, and the test set serves as the final benchmark for the model's effectiveness. Ensuring a proper balance in data splitting is crucial for unbiased performance evaluation.To establish a baseline for comparison, alternative models or rule-based systems are considered. These baseline models provide a reference point for assessing the advancements achieved by the auto-sentencing model. Comparing the performance against simpler methods aids in understanding the value addition brought by the NLP model.

Error analysis is a critical aspect of result evaluation. Identifying common types of mistakes made by the model helps in understanding its limitations. Patterns in errors may guide further model refinement and feature engineering. Moreover, generalization to new, unseen data is assessed to ensure the model's applicability in real-world scenarios.

Feedback from end-users or domain experts who interact with the auto-sentencing system offers valuable perspectives on its usability and practicality. Ethical considerations, such as bias detection and fairness, are also addressed during the analysis to ensure responsible deployment.The result analysis is an iterative process. Based on the findings, adjustments are made to the model architecture, hyperparameters, or the dataset, leading to an improved iteration. This iterative improvement loop is crucial for the continual enhancement of the auto-sentencing model and its sustained success in real-world applications.

4.4 Final Note

While going about the implementation, one thing that was considerably disturbing to us was the computation time and efficiency required to process the dataset, as the dataset we had chosen was considerably a large one. So, as a work around for it, we looked for other methods. As a result of our search, we came across Google Colaboratory, which is an open source version of Jupyter Notebooks running on Google's servers. It has recently launched a GPU support which helped us to run the computations quickly. So, we have completed the pre-processing on Colaboratory and then ran the baseprogram on our system.

CHAPTER 5

CONCLUSION

5.1 Conclusion

In conclusion, the analysis of an auto-sentencing project in natural language processing (NLP) is a multifaceted process that involves a combination of quantitative and qualitative assessments. Through a systematic approach, we gain valuable insights into the model's performance, strengths, and areas for improvement.

The foundation of this analysis lies in the selection of appropriate evaluation metrics tailored to the nuances of the auto-sentencing task. Precision, recall, F1-score, and accuracy serve as quantitative benchmarks, providing a measurable foundation for assessing the model's success. The division of the dataset into training, validation, and test sets ensures a rigorous evaluation methodology, preventing biases and overfitting.

Baseline models and rule-based systems provide essential points of reference for understanding the advancements brought by the NLP model. By comparing against simpler methods, we can gauge the value added and unique contributions of the auto-sentencing approach.

The training process, accompanied by meticulous monitoring on both training and validation sets, allows for the fine-tuning of the model. Hyperparameter tuning further refines the model's configuration, optimizing its performance. The chosen metrics are then applied to the test set, offering quantitative insights into precision, recall, and overall effectiveness.

Qualitative assessments are equally crucial, involving an examination of the generated sentences for coherence, context relevance, and grammatical correctness. Error analysis sheds light on common mistakes, guiding future model refinements. Generalization to new data and user feedback contribute to a holistic understanding of the model's real-world applicability.

Ethical considerations, including bias detection and fairness, are integral parts of the analysis process. Addressing these aspects ensures responsible deployment and mitigates potential societal impacts.

The result analysis is an iterative journey. Based on findings, adjustments are made to the model architecture, hyperparameters, or dataset. This continuous improvement loop is vital for adapting the auto-sentencing model to evolving requirements and ensuring its sustained success in real-world applications.

In summary, the analysis of an auto-sentencing project in NLP is not merely a static assessment but a dynamic process of refinement and enhancement, driven by a commitment to delivering reliable, fair, and effective automated sentence generation.