Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

A MINI PROJECT REPORT

On

TEXT PREDICTOR USING N-GRAMS

Submitted in partial fulfillment of the requirement of
University of Mumbai for the Course

In

Computer Engineering (IV SEM)

Submitted By
Janhavi Anap
Riddhi Narkar

Subject Incharge
Prof. Merlin Priya Jacob

Parshvanath Charitable Trust's

A. P. SHAH INSTITUTE OF TECHNOLOGY

(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)

(Religious Jain Minority)

CERTIFICATE

This is to certify that the requirements for the project report entitled '**Text Predictor using n-grams '** have been successfully completed by the following students:

Name	Moodle Id
Janhavi Anap	19102043
Riddhi Narkar	19102003

In partial fulfilment of the course Python Programming (CSL405) in Sem: IV of Mumbai University in the Department of Computer Engineering during academic year 2020-2021.

Sub-in-Charge

Parshvanath Charitable Trust's

A. P. SHAH INSTITUTE OF TECHNOLOGY

(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

PROJECT APPROVAL

The project entitled '**Text Predictor using n-grams**' by **Janhavi Anap**, and **Riddhi Narkar** are approved for the course of Python Programming (CSL405) in Sem: IV of Mumbai University in the Department of Computer Engineering.

Subject-in-Charge

Date:

Place: Thane

Table of Contents

Abstract

One of the major technological marvels of this tech-infused 21st century is the ability of us to connect digitally. With the advent of sophisticated communication tools, being able to express oneself through a digital medium effectively and efficiently needed a reliable mechanism. A very popular method of expressing thoughts, ideas or sharing information, is texting a message or posting a typed text piece on the internet. Often, typing can be prone to errors and not much efficient and quick when it comes to long typing sessions.

A word predictor is a piece of software which suggests words after processing your input. The input here could be a string of any number of words, and it normally depends on the algorithm used. An algorithm could be processing in a static way, which doesn't learn, no matter how much it is trained and tested, or it can be dynamic, and thus, adapting the changes in diction, grammar and style of the user. A dynamic word predictor may use a corpus or a database that changes every time a user uses the service and thus updates itself with the changes and advancements of the user. Also, such a system can very easily survive the changing language style, new words, new phrases, or new slangs and hence is versatile and needs less maintenance. In addition to texting and using this software as just a medium for faster and more efficient typing, a slightly differently built model can also be used to enhance the communication experience of persons with disabilities. Such a model would use the Augmentative and Alternative Communication devices as an API.

In this Python project, we tried to build a text predictor using the concept of n-grams which could be deployed for use where one would want to type something using a hardware or software keyboard. It would process the user input using tri-grams and suggest 3 words which would make the sentence grammatically correct in the order of the decreasing frequency which in turn, is based on the typing habits of the user.

List of figures

List of tables

Problem Definition

One important aspect of effective and fast communication is proper grammar, and choice of words, spellings and more meaningful sentences. In this fast age of electronics and internet, racing our fingers with the speed of our thoughts is error prone.

A text predictor not only focuses on suggesting next words which suit your style, but also to the fact that how much the suggested word would make a complete sense in the context of the information being conveyed through the sentence being typed. In addition to its listed expectations, if a software would be trained to your style of grammar and your diction, and which would adapt the changes of the language you try to adapt in yourself would be an extremely powerful tool and would cater to a wide range of users. Such a tool if provided with the communication service itself would be an added benefit. Also, a text predictor can be an extremely powerful tool when it comes to communication of people with disability. Such a system, allows them to communicate without much effort, and seamlessly and it enhances their current ability to communicate.

A n- gram word predictor solves this problem of less efficient communication by suggesting grammatically sound, and contextually relevant words which are predicted on the basis on the words earlier typed but the user. The degree of n here is 3, hence this is a trigrams implementation which is considered as one of the best values for n, as it is computationally less expensive, easier to develop and maintain, and highly efficient in implementation.

Introduction

1) WORD PREDICTION

Since we started to document and type things more often than we wrote, a lot of documentation required less time to create. Eventually this generated a need for faster and more error prone documentation. Word prediction has been a part of this change; in fact word prediction was responsible to drive this change.

Word prediction is used on a lot of platforms: right from your usual keyboard to well-developed apps and cloud services like Gmail, Microsoft Word, Grammarly, Google Docs, etc. A peculiar thing about these services is that you get suggestions which are grammatically precise. To achieve this, large platforms use a huge number of business and literature corpora to train their models. These models get better the more you use them, i.e., they get tailored to your personal writing and documenting style. They train themselves slowly according to the user's diction, grammar style, spellings. This not only allows you to reduce errors, but over time, also maintains your writing technique. Today, text prediction is used on almost all platforms.

2) USING N-GRAM MODEL TO PREDICT TEXT

N-grams is a very effective model used for text prediction. It retains the grammatical sense while suggesting words, and hence, maintains the semantics of a sentence. It does this by comparing pairs of words and processing which words is followed by what string of word(s). This allows n-grams to predict grammatically correct words, if it is trained under

good quality corpora. The more the quantity and quality of the corpora, better the prediction.

The idea behind the n-grams lies in its formal definition is “a contiguous sequence of n items from a given sample of text”. A n-gram with n = 1 is a unigram, which means it just a single word. A bi-gram predicts the second word on the basis of its previous word. So, the degree of n here is 2. A tri-gram model, which we have implemented here, predicts the third word, on the basis of the first and second word. It predicts the third word by scanning the first two words together, as a single entity. This mere fact, of processing the previous bunch of words together gives it an edge and allows it to maintain semantic and grammatical sense.

The answer of how it predicts that lies in the corpus and the concept of Markov chains. Markov chains, named after Andrey Markov, are mathematical systems that hop from one "state" (a situation or set of values) to another.

For example, if you made a small Markov chain model of 3 sentences :
‘I like to…’,
‘I like that’, and
‘I am a…’,
you would include "I", "like", "am" and "like", “to”, “that”, “a” as states, which together with other behaviors could form a 'state space': a list of all possible states. In addition, on top of the state space, a Markov chain tells you the probabilitiy of hopping, or "transitioning," from one state to any other state---e.g., the chance that a you’ll get a “to” after “like”.

The following diagram illustrates a complete version of the above example.

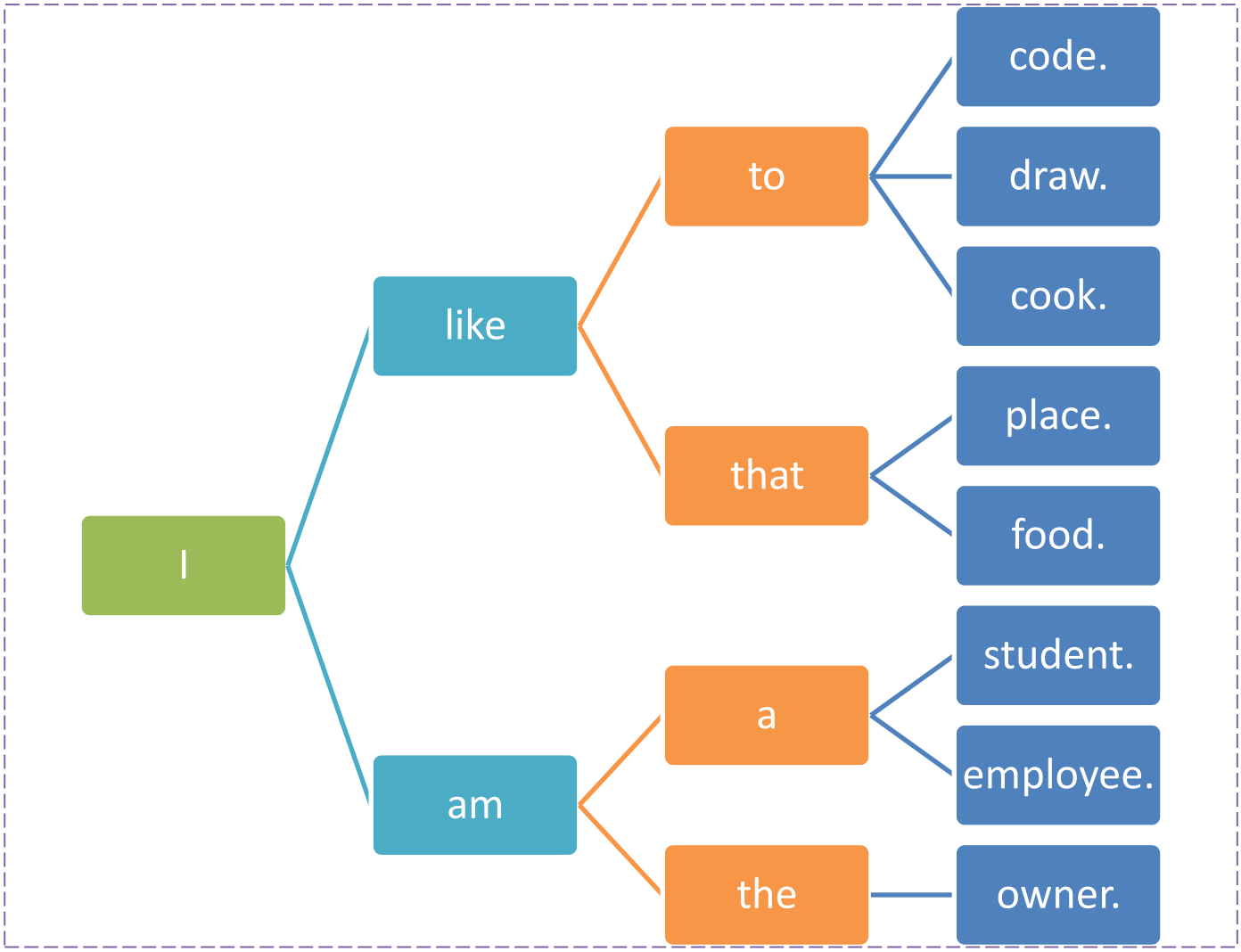


Figure 1: Markov chains

Here are some dummy probabilities, as to how the Markov model can favour one word over the other and make a choice leading to the prediction it makes.

Current word	Next possible words	Probability
I	(like, am)	(0.25, 0.75)
like	(to, that)	(0.90, 0.10)
am	(a, the)	(0.80, 0.20)
a	(student, employee)	(0.50, 0.50)
the	owner	1
to	(code, draw, cook)	(0.40, 0.20, 0.40)
that	(place, food)	(0.40, 0.60)

Table 1: Markov chain probability

Description of the Modules

This project consists of 2 modules, one the GUI application part, and other, the core logic part.

GUI module:

We used Tkinter for designing this.

Sr.no	Method	Description
1	update	Updates the values of predictions in the GUI. It takes a sorted list as a parameter, and based on its length displays the first 3 contents in proper probability order. It puts a "-" for each missing prediction (if the length of list is less than 3, i.e. when the corpus doesn't have any further predictions). The following my_tracer method calls it.

2	my_tracer	It fetches the suggestions passed from the Markov chain module and sends them to update to be displayed on the buttons
3	btnClear	Clears the input text field by setting its value to an empty string
4	btnClick	Adds the word corresponding to the predictions' button in the input text field and sets the cursor to end of the newly updated sentence
5	btnAdd	Adds the current sentence in the input text box to the history as well as the corpus. While updating the history and the corpus, it scans for spelling errors using gingerit library. If the history is greater than 3 sentences, it displays the latest 3 sentences, and drops the remaining from the application window for better and clutterless presentation.

Table 2: Methods of GUI module

Main driver code:

The rest of the code is mainly the UI part, which calls these above methods.
The GUI consists of a input field, followed by the three buttons consisting of top 3 predictions and then followed by a clear button and an add button.
The remaining window consists of a history section where all the sentences which are added in the corpus after pressing add are displayed.

Markov Chain module:

We initialize three dictionaries here, first_words (contains all the first words as keys and their frequencies as values), second_words (contains all the second words as values and first words as keys) and transitions(contains all the subsequent words in a list as the value and the tuple of first two words in order as the key).

Sr.no	Method	Description
1	add_to_dictionary	This adds a key value pair to the specified dictionary.

2	calculate_probab	Returns dictionary of words and their corresponding frequencies from the given list.
3	train_markov	The main markov chain implementation. Runs a nested for loop, the outer one scans every sentence of the corpus and the inner one scans every word of the sentence. Makes the word devoid of any whitespaces or punctuation. According to the words' position in the sentence, it places them in the correct dictionary. Then using that, it calculates the frequencies for second_words and transition dictionary and updates the dictionary for a final time with frequencies.

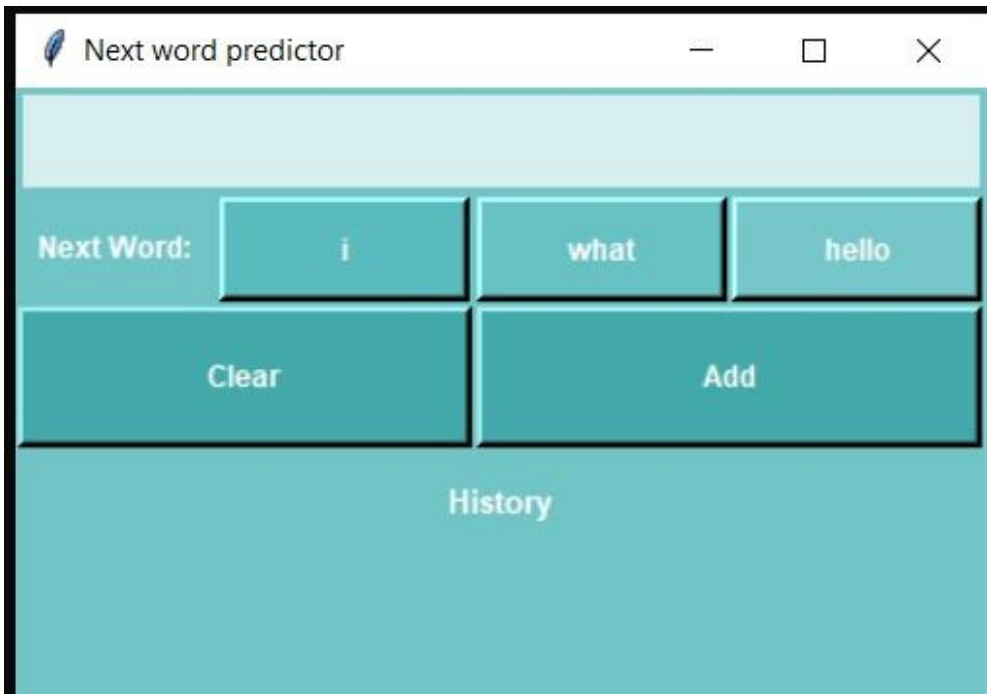
4	suggest_first_words	Returns the sorted first_words dictionary to the GUI module to showcase some predicted sentence startings when the application starts and before the user enters any word.
5	update_corpus	Updates the corpus with the newly added sentences from the GUI.
6	suggestions	Selects a dictionary, either second_words or transitions according to the number of words user types, and returns the corresponding sorted dictionary.

Table 3: Methods of Markov Chain module

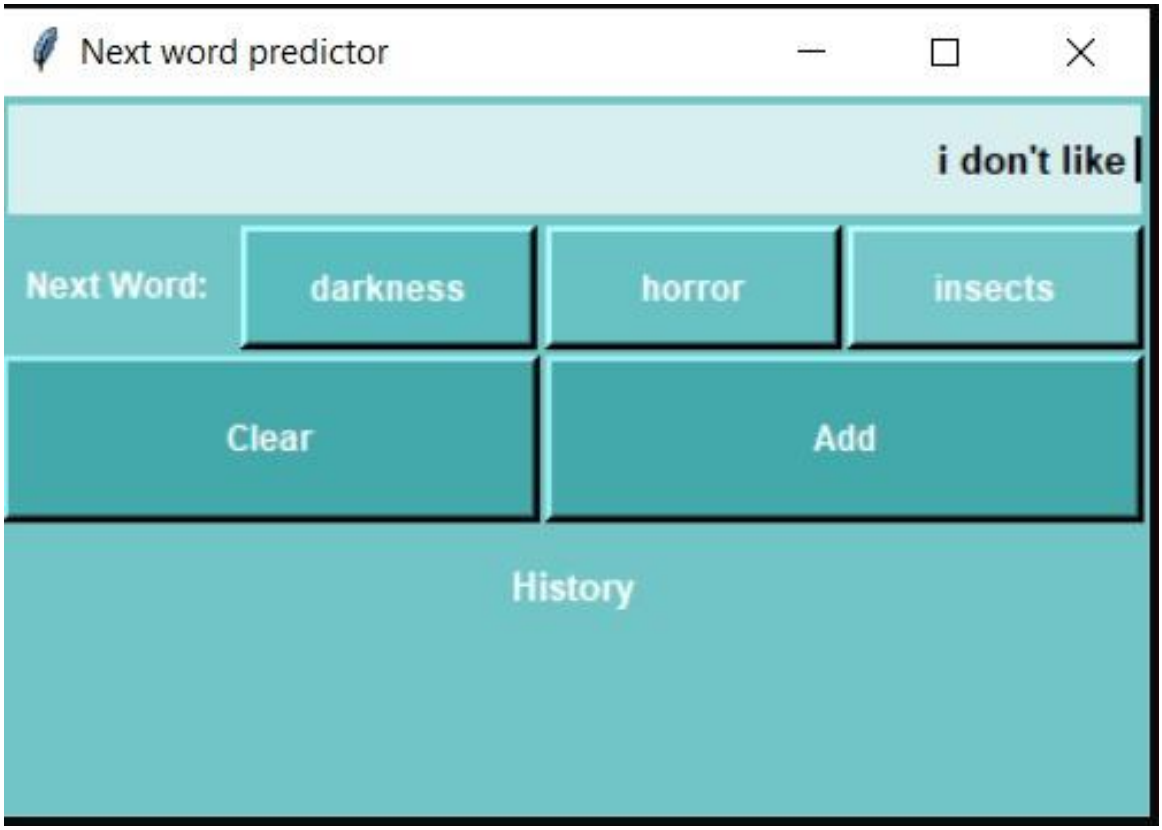
This module more or less contains only methods, which are either called in the same module itself, or in the GUI to fetch processed data from here.

Implementation

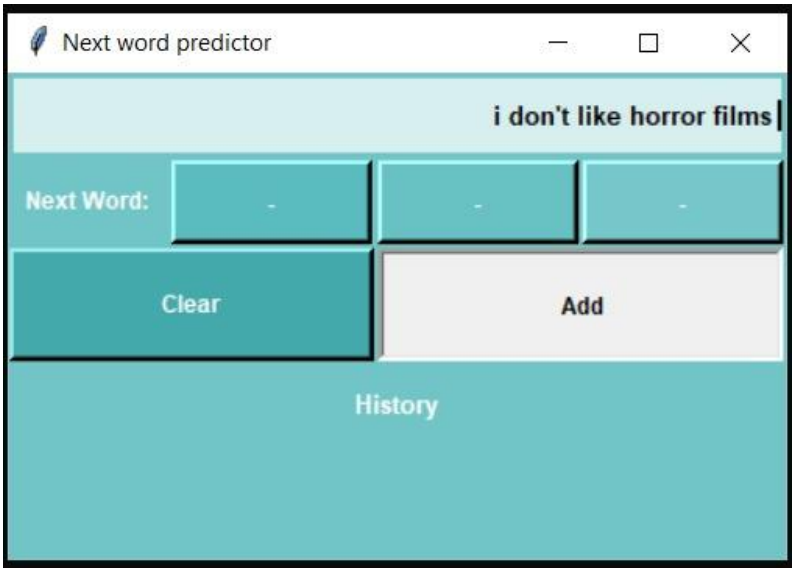
1) When we start the application for the very first time, the first words are displayed on the buttons.

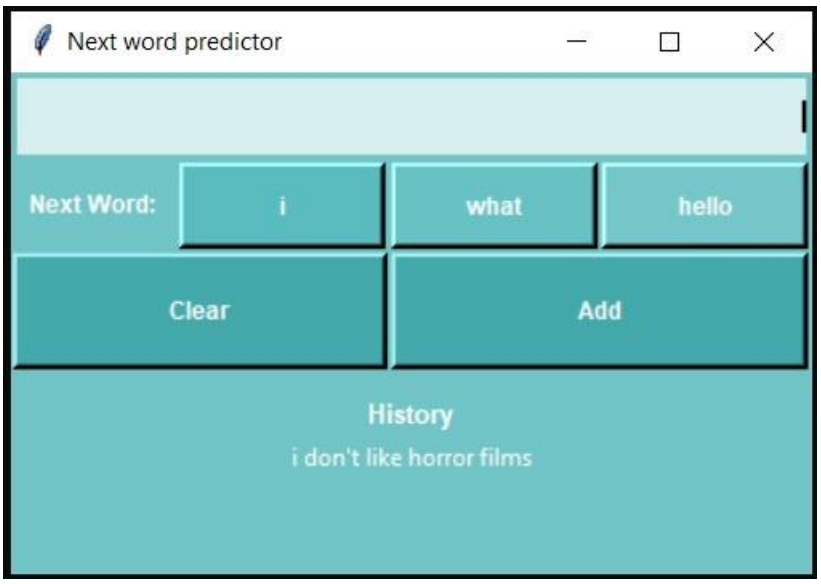


2) How the predictions change corresponding to the input

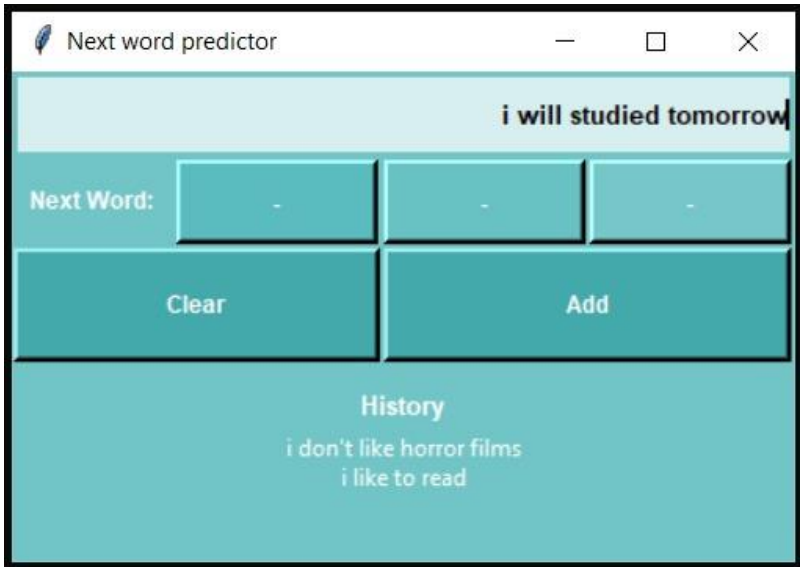


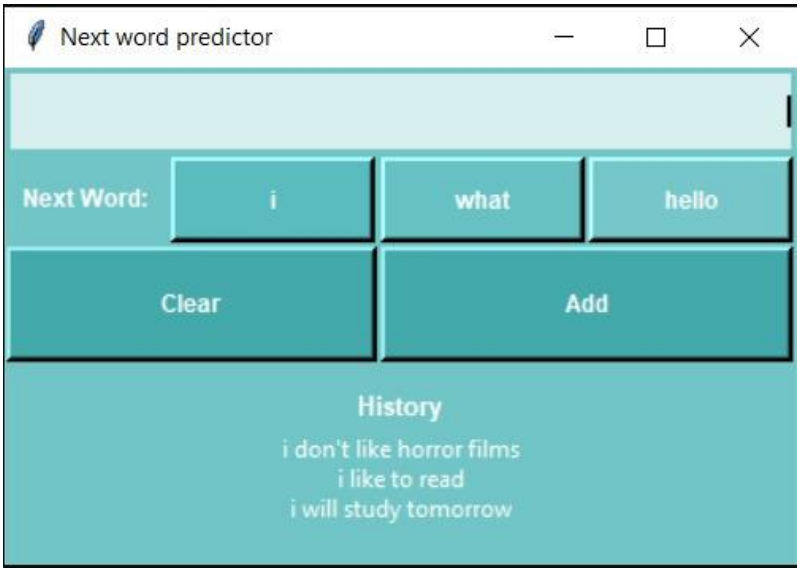
3) Add to history (this by default adds the sentence to corpus too)

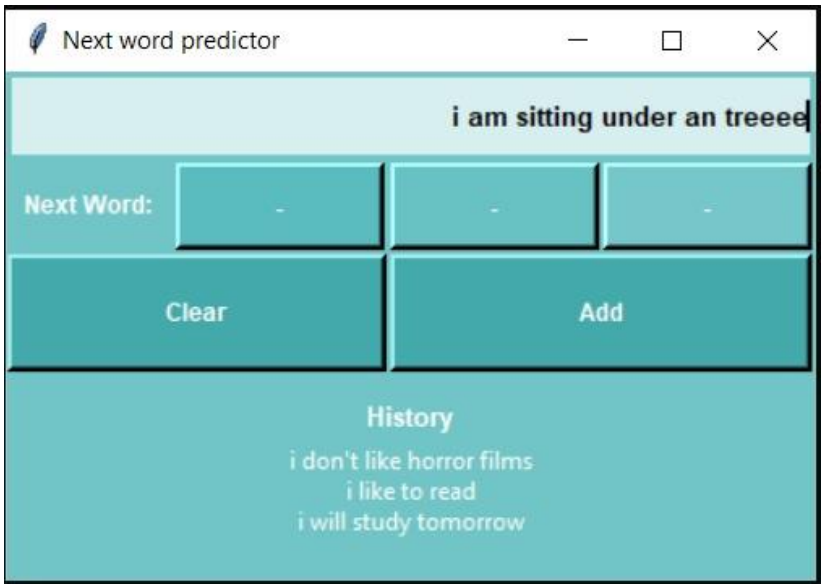


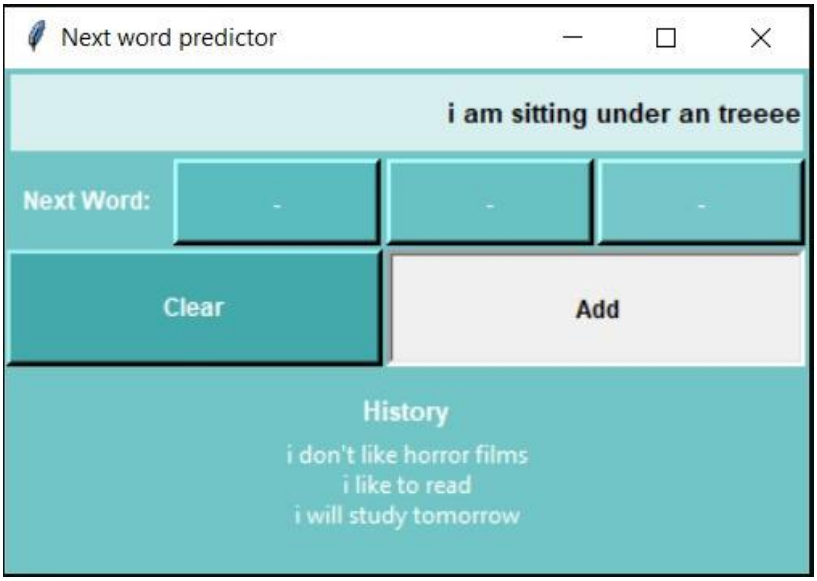


4) Grammar and spelling check

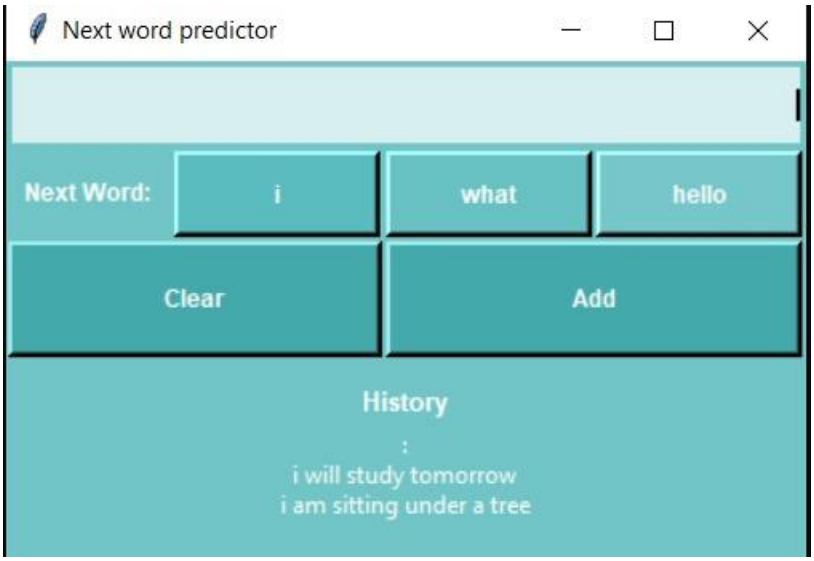




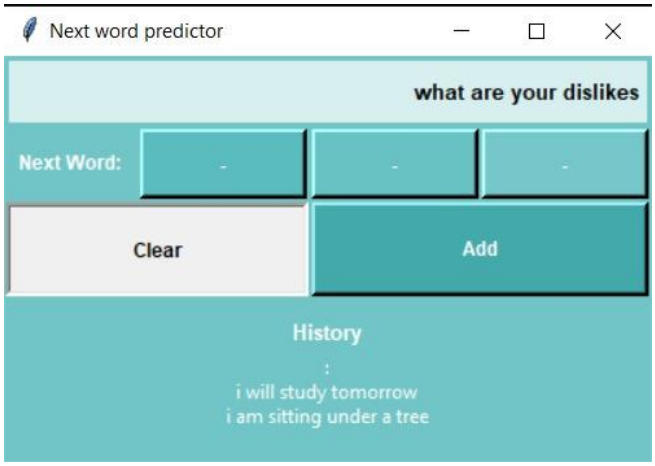


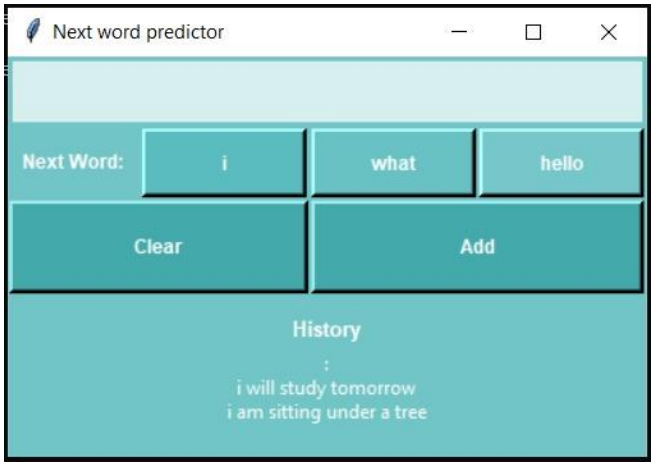


(Adding the sentence in the History corrects the grammar)



5) Clear button





Conclusion and Future scope

In the project we created, our database was updated with new sentences each time we made a choice based on the predictions, or even if we added a new word of our own choice, thus ignoring the predictions. But, the changes reflect in the corpus after we restart the application. Since, we are using file handling, Python doesn't update the corpus text file until the program is terminated. This can be, of course, eliminated by developing a database using MySQL. This will make the prediction more fast and robust, as the corpus would immediately be reflected with the changes.

Word prediction, thus, can have a major impact when it comes to using a language in its correct sense. It can be deployed to use for teaching motor skills to children who are learning a new language, but allowing them to grasp the core semantics intuitively, without having the need to write. It allows one to use and understand the language, and document anything with a very less error margin.

References

ARTICLES AND BLOGS

1) Duke University. "Java Programming: Principles of Software Design". coursera.org. https://www.coursera.org/learn/java-programming-design-principles(accessed Feb. 13 2021).

2) GaganTalreja1. "Word Prediction using concepts of N – grams and CDF". geeksforgeeks.org. https://www.geeksforgeeks.org/word-prediction-using-concepts-of-n-grams-and-cdf/ (accessed Feb. 13 2021)

3) M. Mayo. "Building a Wikipedia Text Corpus for Natural Language Processing – Kdnuggets". kdnuggets.com. https://www.kdnuggets.com/2017/11/building-wikipedia-text-corpus-nlp.html (accessed Feb. 13 2021)

4) Omar14. "Estimating the best length of n-gram". stackexchange.com. https://stats.stackexchange.com/questions/155483/estimating-the-best-length-of-n-gram. (accessed Feb. 13 2021)

5) N. Bertangnolli. "Good Grams: How to Find Predictive N-Grams for your Problem". towardsdatascience.com. https://towardsdatascience.com/good-grams-how-to-find-predictive-n-grams-for-your-problem-c04a5f320b39 (accessed Feb. 13 2021)

6) U. Malik. "Python for NLP: Developing an Automatic Text Filler using N-Grams". stackabuse.com. https://stackabuse.com/python-for-nlp-developing-an-automatic-text-filler-using-n-grams/ (accessed Feb. 13 2021)

7) Tutorialspoint. "Python - Regular Expressions". tutorialspoint.com. https://www.tutorialspoint.com/python/python_reg_expressions.htm (accessed Feb. 13 2021)

8) W3resource. "Python: Sort (ascending and descending) a dictionary by value". w3resource.com. https://www.w3resource.com/python-exercises/dictionary/python-data-type-dictionary-exercise-1.php?passed=passed (accessed Feb. 13 2021)

9) O. Borisov. "Text Generation Using N-Gram Model". towardsdatascience.com. https://towardsdatascience.com/text-generation-using-n-gram-model-8d12d9802aa0 (accessed Feb. 13 2021)

10) Wikipedia contributors. "N-gram". wikipedia.org. https://en.wikipedia.org/wiki/N-gram (accessed Feb. 13 2021)

11) Wikipedia contributors. "Markov chain". wikipedia.org. https://en.wikipedia.org/wiki/Markov_chain (accessed Feb. 13 2021)

12) V. Powell. "Markov Chains". setosa.io. https://setosa.io/ev/markov-chains/ (accessed Feb. 13 2021)

13) S. Kapadia. “Language Models: N-Gram”. towardsdatascience.com. https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9 (accessed Feb. 13 2021)

14) S. Bhutani. “Markov chains - next word Prediction - Python Theory”. youtube.com. https://www.youtube.com/watch?v=Ls38CLOvazc (accessed Feb. 13 2021)

15) DataScienceDojo. “N-Grams in Natural Language Processing”. youtube.com. https://www.youtube.com/watch?v=E_mN90TYnlg (accessed Feb. 13 2021)

16) DecisionForest. “What are Unigrams, Bigrams & N-Grams N-Gram Analysis for Machine Learning Projects | NLP Projects”. youtube.com. https://www.youtube.com/watch?v=MZIm_5NN3MY (accessed Feb. 13 2021)

17) PlanetOjas, “n gram model”. youtube.com. https://www.youtube.com/watch?v=5263Xqr1YCc&t=464s (accessed Feb. 13 2021)

Acknowledgement

I, **Riddhi Narkar**, on behalf of my project team would like to extend a token of gratitude to **Prof. Merlin Priya Jacob,** our project guide who helped us throughout the duration of this project. Her suggestions, ideas helped us to make our code better, efficient and robust. She helped us identify the corner cases where our application might run into an error, and also suggested ways to tackle those. After working on this project, both of us developing a liking for natural language processing works and help us scratch the surface of this beautiful piece of technology. Furthermore, I'm immensely grateful to my project partner, **Janhavi Anap**, who supported me and kept me inspired throughout the course of this project. I got to learnt a lot from her during building our code. Her wonderful approaches made working on this project an exciting task.