Tribhuvan University

Orchid International College

A Final Year Project Report
On

WORD SUGGESTION AND AUTO WORD COMPLETION SYSTEM USING NAÏVE BAYES

Submitted To:

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

ORCHID INTERNATIONAL COLLEGE

In partial fulfillment of the requirement for the Bachelor Degree in Computer Science and Information Technology

Submitted By:

Nishanta Nepal (20816/075)

April, 2023

ORCHID INTERNATIONAL College

SUPERVISOR'S RECOMMENDATION

I hereby recommend that the report prepared under my supervision by Nishanta Nepal (TU Exam Roll No. 20816/075), entitled "**WORD SUGGESTION AND AUTO WORD COMPLETION SYSTEM**" in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology be processed for evaluation.

…………………..…….

Er. Diwakar Upadhyaya
Lecturer, Department of CSIT
Orchid International College
Bijayachowk, Gaushala

ORCHID INTERNATIONAL College

LETTER OF APPROVAL

This is to certify that this project prepared by Nishanta Nepal (TU Exam Roll No. 20816/075), entitled "**WORD SUGGESTION AND AUTO WORD COMPLETION SYSTEM**" in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

----------------------- **Er. Diwakar Upadhyaya** Supervisor, Lecturer, Orchid International College Bijayachowk, Gaushala	----------------------- **Er. Dhiraj Kumar Jha** Head of Department, Orchid International College Bijayachowk, Gaushala
----------------------- **Sikha Sharma** Internal Examiner, Full-Time Faculty, Orchid International College Bijayachowk, Gaushala	----------------------- **External Examiner** Central Department of Computer Science and IT Tribhuvan University, Kirtipur, Nepal

ACKNOWLEDGEMENT

We are quite grateful to the Orchid International College for giving us the opportunity to complete a project as part of our course requirements. We owe a debt of gratitude to our manager, Mr. Dhiraj Kumar Jha, for his constant guidance and assistance during this endeavor. His assistance with this endeavor enabled us to progress along the way.

We also appreciate Mr. Diwakar Upadhyaya's leadership in our project, which inspired us to work harder on it by offering us additional assistance and project-related suggestions. We also want to express our gratitude to all of the professors who inspired and supported us while we worked to complete this project.

We also want to express our gratitude to IOST for providing us with the infrastructure and tools we needed to finish this project. Finally, we would like to thank our friends, family, and coworkers for their encouragement and support.

Without all of these people's encouragement and assistance, this project would not have been feasible. We owe them all a huge debt of gratitude.

Thank you,
Nishanta Nepal (20816/075)

ABSTRACT

The goal of this project is to create a word prediction system that would suggest words and phrases to users based on the context of their writing in order to assist them with their writing duties. In contemporary systems including e-commerce stores, word processors, search engines, and web browsers, the word recommendation has played a significant role. The usage of a word prediction system improves a user's typing speed and reduces error. A useful word prediction system that can be utilized to speed up and increase the effectiveness of writing jobs will be the result of this research. In order to predict the word being written, this project uses a machine learning model. This online application was created using the naive bayes algorithm together with HTML, CSS, JavaScript, and NodeJS.

Keywords: ***Word prediction, Recommendation, Word completion, Naïve bayes***

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Introduction

A system can guess the remaining letters of a word while a user types it using the word suggestion and auto word correction feature. When it successfully anticipates the term a user intends to submit after only a few characters have been entered into a text input field, auto prediction speeds up human-computer interactions. It functions well when writing structured and predictable material or in fields where the number of available words is constrained. After a user writes a word a few times, many word prediction algorithms pick up on it and can then suggest alternatives based on the user's learned writing habits.

A system's word prediction and autoword correction feature anticipates the remaining letters of the word that the user is typing. By accurately predicting the term a user intends to submit after only a few characters have been entered into a text input field, auto prediction speeds up human-computer interactions. It works well when writing structured and predictable language or in fields where there are few words that can be used. Many word prediction algorithms pick up on new words after the user uses them a few times and can then suggest alternatives based on the user's learned patterns.

When the writer types the first letter or a few letters, Word Suggestion predicts one or more potential words as a choice. If the user's intended term is among the list of available words that appears on the screen, they can choose it. The writer must insert the following letter of the word if the user's desired term is not predicted.

1.2 Problem Definition

Many systems, including but not limited to search engines, web browsers, e-commerce applications, and many more, heavily utilize word suggestions. The volume of text produced by all systems, including online shops and social media, has also increased significantly as a result of technical advancement. The systems still have a slow and clumsy typing procedure, though. A word prediction system that can reliably guess the word that the user is typing is required to get around this. The system must to be extremely accurate and effective, and it ought to be able to handle various word types.

1.3 Objective

- To determine the probability of the word the user is typing.
- To offer the word before the user finishes typing it.
- To let users, add new words to the database.
- To limit the chance of a user making a typing error and save time.

1.4 Scope and Limitation

The goal of this project is to analyze the text that is being produced and foretell each word in its entirety for the user. The goal of the "Auto word prediction" project effort is to accurately complete a word that the user is typing.

A few examples of scopes are:

- This system can be used in search engines' search bars;
- This system can be utilized in keyboards for auto completion.
- It can be applied to product searching in an e-commerce application.
- In code editors and IDEs, it can be utilized as a keyword completion.

Some of the limitations are:

- The caliber and variety of the text utilized to train the model will initially be the system's limiting factors.
- The success of accurate prediction will depend on whether user find it helpful and useable.

1.5 Development Methodology

A form of software development process called incremental software development includes segmenting a huge software project into smaller, more manageable components was used to develop our system. Planning, design, implementation, testing, and deployment are some of the steps that the incremental development process normally takes. Each incremental release follows the same procedure to produce a fully operational version of the product.

Overall, our software projects were developed incrementally, which is a flexible and adaptable methodology.

CHAPTER 2: LITERATURE REVIEW

2.1 Literature Review

Advances in NLP applied to word prediction is the first study [2] linked to such a system that was discovered. Fast Type System was used. When employed with inflected languages, the cutting-edge word prediction tool FastType transcends the limitations of traditional approaches. FastType was developed to increase Keystroke Saving and is based on mixed statistical and rule-base methodologies that rely on strong open-domain language resources. FastType has been tested and analyzed in a few benchmark test cases, demonstrating a considerable improvement in Keystroke Saving that now approaches 51%, on par with word prediction techniques for non-inflected languages. The study presented new findings for inflected languages and offered an innovative technique to word prediction.

Next Word Prediction with NLP and Deep Learning was covered in another paper [3]. It was discovered in this research that the model took into account the final word of a specific sentence and predicted the following potential word. Deep learning, language modeling, and natural language processing were employed.

Another publication was identified, Hybrid Model for Word Prediction [4] utilizing Naive Bayes and Latent Information. This study suggests a hybrid word suggestion model that takes into account nearby words in the vicinity of blank spaces and is based on Naive Bayes and Latent Semantic Analysis. Results indicate that in the MSR Sentence Completion Challenge, this model could complete sentences with an accuracy of 44.2%. This study suggests a novel hybrid model that combines the Naive Bayes and LSA models to predict words. [4] Additionally, the Gradient Descent technique was used in conjunction with these optimization parameters to increase prediction accuracy.

The word prediction algorithm was put into use on the library management system at an Illinois institution, according to a paper [5]. In the experiment, some students used the prediction system while others did not. Students that used auto complete were found to gain from elements including spelling, confidence, speed, topic focus, and more.

2.2 Background Study

In many aspects of human computer interaction, such as text entry, search engines, and machine translation, word prediction systems have developed into a crucial tool. These systems suggest words or phrases based on the text input using a variety of methodologies, including statistical language modeling, rule-based systems, and neural networks.

The "predictive text" feature in Tegic Communication's 1995 software system T9 (Text on 9 keys) was one of the first word prediction systems. To anticipate words based on keystores on a mobile phone keypad, T9 used statistical language modeling and a rule-based system. T9 quickly gained traction as a feature on smartphones, and its popularity encouraged the creation of further word prediction algorithms for a variety of uses.

In order to increase prediction accuracy, more contemporary word prediction systems have turned to machine learning methods like recurrent neural networks (RNNs) and Naive Bayes classifiers. In order to simulate the likelihood of a word or phrase occurring given the context, naive Bayes classifiers use statistical techniques.

Since the early days of T9, word prediction systems have advanced significantly, and they still have a significant impact on many aspects of human-computer interaction. Future improvements in prediction and accuracy can be anticipated as long as this field of study is pursued.

CHAPTER 3: SYSTEM ANALYSIS

3.1 Requirement Analysis

3.1.1 Functional Requirement

An explanation of the service that the software must provide is contained in a functional requirement (FR). It provides information about a software component or whole software systems. A function is nothing more than the inputs, behavior, and outputs of the software system. A system's likely function can be determined by a computation, data manipulation, business process, user interaction, or any other specialized feature. Functional requirements for this project are:

3.1.1.1 Use Case Diagram

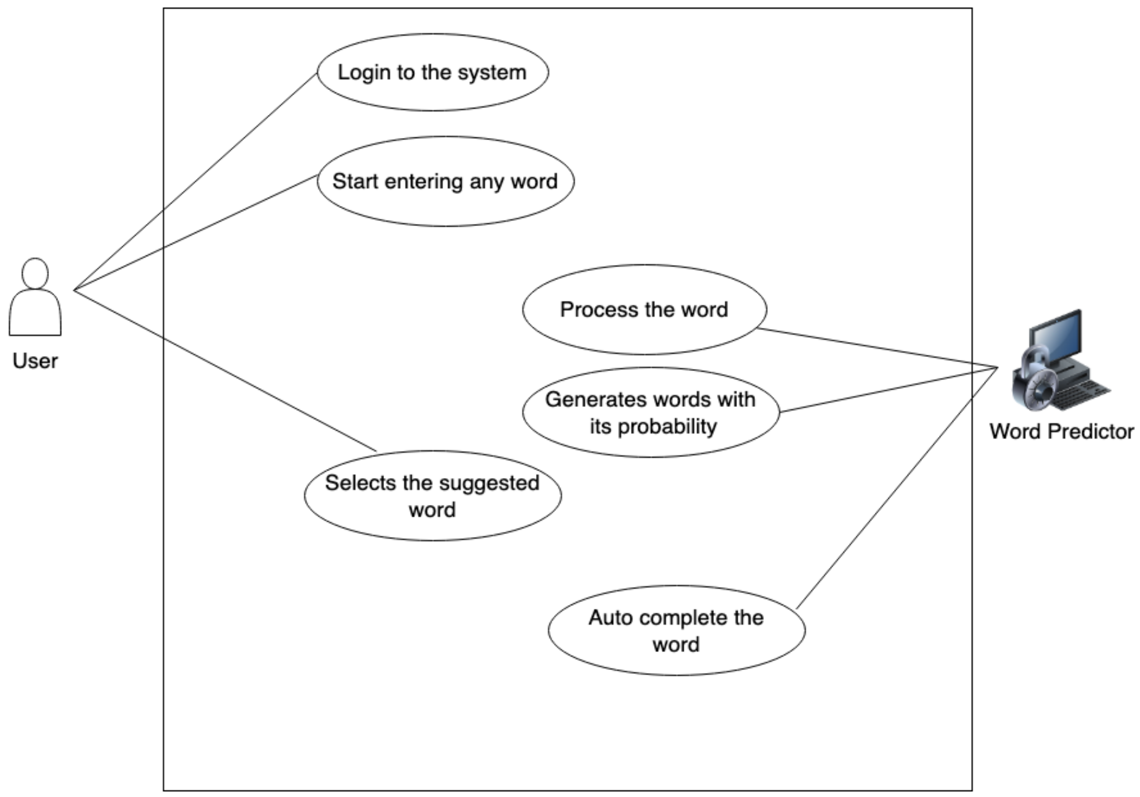


Figure 3.1: Use Case Diagram

Use-Case Description

Table 3.1: System Login

Use case Identifier	UC1 – Login to the system
Primary Actor	User
Secondary Actor	None
Description	User tries to Log in to the system
Pre-condition	Username and password must be correct
Success Scenario	User gets logged in to the system
Failure Scenario	Login failed

Table 3.2: Typing Word

Use case Identifier	UC2 – Start entering any word
Primary Actor	User
Secondary Actor	None
Description	User enters the letters of the word
Pre-condition	User must be logged into the system.
Success Scenario	Letters get typed in the search bar
Failure Scenario	User unable to type letters

Table 3.3: Word Processing

Use case Identifier	UC3 – Process the word
Primary Actor	System
Secondary Actor	None
Description	System takes in the letters and processes it
Pre-condition	System must be trained with the ML algorithm
Success Scenario	The system processes the input with trained algorithm.
Failure Scenario	The system becomes unable to process inputs.

Table 3.4: Word Generation

Use case Identifier	UC4 – Generate probable word
Primary Actor	System
Secondary Actor	None
Description	System generates words with high probability
Pre-condition	The words must match the pattern of the letters provided by the user in input.
Success Scenario	System successfully suggests the possible word to the user.
Failure Scenario	System becomes unable to predict the word.

Table 3.5: Word Suggestion

Use case Identifier	UC5 – Selects the suggested word
Primary Actor	User
Secondary Actor	None
Description	The user selects the word suggested by the system as his choice.
Pre-condition	The system should predict and suggest all possible word to the user
Success Scenario	User selects the word of his choice
Failure Scenario	User doesn’t find his choice of word

Table 3.6: Auto Complete

Use case Identifier	UC6 – Auto complete the word
Primary Actor	System
Secondary Actor	None
Description	System completes the selected word
Pre-condition	The system should suggest the possible word
Success Scenario	System completes typing the word
Failure Scenario	System is unable to complete the word

3.1.1.2 Other Requirement

The user may enter text, and the algorithm would match it after analyzing the characters to suggest a possible word or words.

3.1.2 Non-Functional Requirements

3.1.2.1 User Interface

The user interface will be a straightforward web application where the user may input text and receive suggestions for words.

3.1.2.2 Performance

This word prediction system should be able to give the user information (predicted words), and the words suggested should be highly accurate and correct.

3.1.2.3 Scalability

For the system to offer numerous functions, it must be scalable. The system should be able to accommodate new features with ease. This system ought to be able to integrate with other word prediction systems including search engines, text keyboards, e-commerce product searches, and others.

3.1.2.4 Usability

The user interface of the program that will be created must be simple enough for even the average individual to understand and utilize.

3.1.2.5 Maintainability

It is necessary to maintain the system. It is possible to maintain the system to improve its functionality or add new features.

3.2 Feasibility Analysis

Feasibility analysis is research that evaluates a project's potential to see if it is worthwhile to pursue it and gives data to enable decision-makers decide whether to do so. It is a research to determine whether a project, product, or solution is viable, workable, and acceptable. It also contains a technical, financial, and operational part of the project assessment.

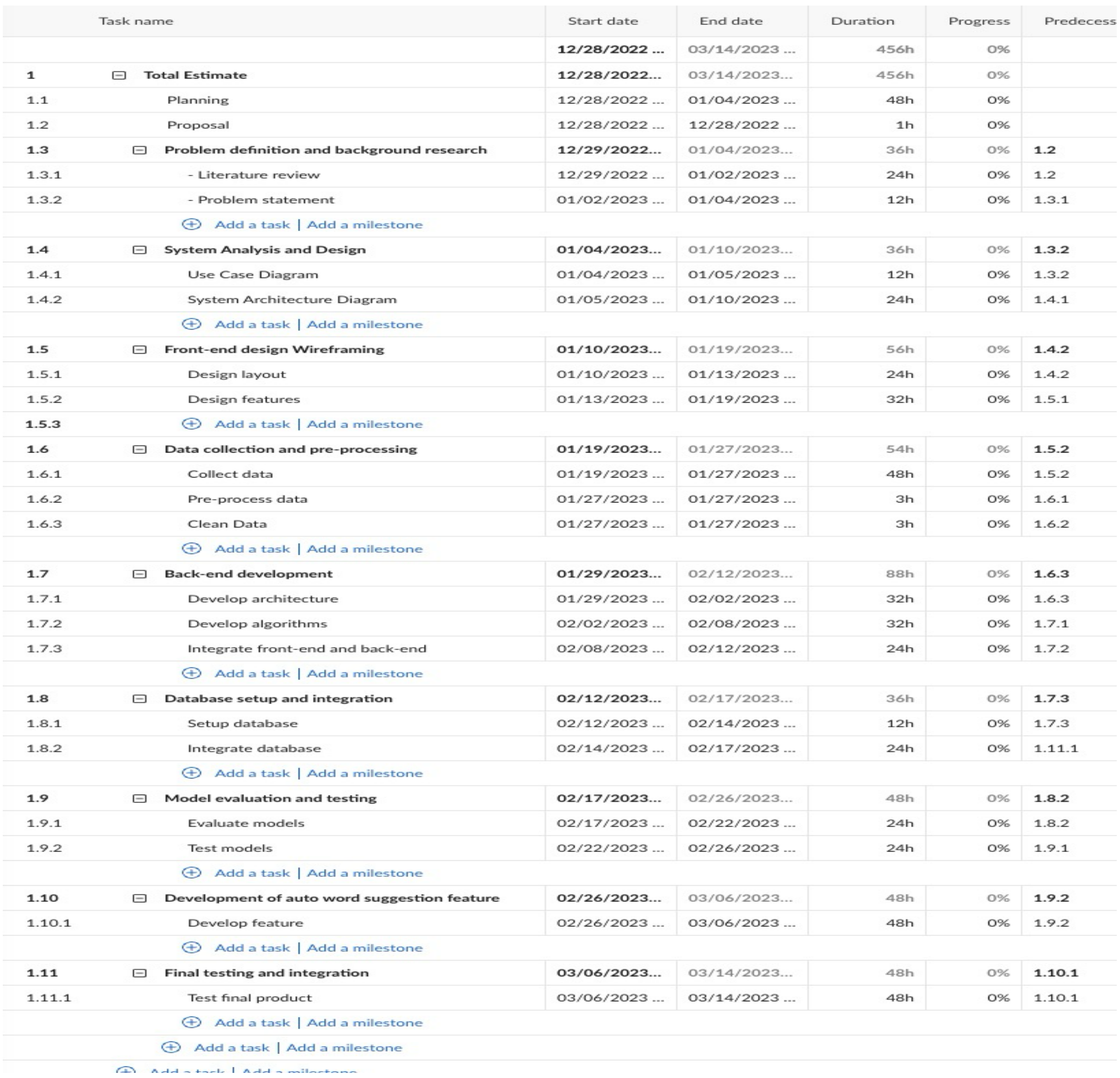
3.2.1 Technical Feasibility

The ability of a project to be completed with the technology resources at hand is referred to as technical feasibility. The machine learning technique for this word prediction project can be run on an IDE without needing any specific setup. These factors are all easily attainable. Technical expertise, data accessibility, and computational resources are further project feasibility considerations.

3.2.2 Operational Feasibility

When discussing a project's ability to be implemented successfully, the term "operational feasibility" is used. Aspects like user acceptance of the word prediction system in daily use, integrating this project with the existing system, and the technical support that can be provided to handle any issue in the implementation phase can all be used to gauge the operational viability of a project.

3.2.3 Schedule Feasibility



	Task name	Start date	End date	Duration	Progress	Predecess
		12/28/2022 ...	03/14/2023 ...	456h	0%	
1	⊟ Total Estimate	12/28/2022...	03/14/2023...	456h	0%	
1.1	Planning	12/28/2022 ...	01/04/2023 ...	48h	0%	
1.2	Proposal	12/28/2022 ...	12/28/2022 ...	1h	0%	
1.3	⊟ Problem definition and background research	12/29/2022...	01/04/2023...	36h	0%	1.2
1.3.1	- Literature review	12/29/2022 ...	01/02/2023 ...	24h	0%	1.2
1.3.2	- Problem statement	01/02/2023 ...	01/04/2023 ...	12h	0%	1.3.1
	⊕ Add a task | Add a milestone					
1.4	⊟ System Analysis and Design	01/04/2023...	01/10/2023...	36h	0%	1.3.2
1.4.1	Use Case Diagram	01/04/2023 ...	01/05/2023 ...	12h	0%	1.3.2
1.4.2	System Architecture Diagram	01/05/2023 ...	01/10/2023 ...	24h	0%	1.4.1
	⊕ Add a task | Add a milestone					
1.5	⊟ Front-end design Wireframing	01/10/2023...	01/19/2023...	56h	0%	1.4.2
1.5.1	Design layout	01/10/2023 ...	01/13/2023 ...	24h	0%	1.4.2
1.5.2	Design features	01/13/2023 ...	01/19/2023 ...	32h	0%	1.5.1
1.5.3	⊕ Add a task | Add a milestone					
1.6	⊟ Data collection and pre-processing	01/19/2023...	01/27/2023...	54h	0%	1.5.2
1.6.1	Collect data	01/19/2023 ...	01/27/2023 ...	48h	0%	1.5.2
1.6.2	Pre-process data	01/27/2023 ...	01/27/2023 ...	3h	0%	1.6.1
1.6.3	Clean Data	01/27/2023 ...	01/27/2023 ...	3h	0%	1.6.2
	⊕ Add a task | Add a milestone					
1.7	⊟ Back-end development	01/29/2023...	02/12/2023...	88h	0%	1.6.3
1.7.1	Develop architecture	01/29/2023 ...	02/02/2023 ...	32h	0%	1.6.3
1.7.2	Develop algorithms	02/02/2023 ...	02/08/2023 ...	32h	0%	1.7.1
1.7.3	Integrate front-end and back-end	02/08/2023 ...	02/12/2023 ...	24h	0%	1.7.2
	⊕ Add a task | Add a milestone					
1.8	⊟ Database setup and integration	02/12/2023...	02/17/2023...	36h	0%	1.7.3
1.8.1	Setup database	02/12/2023 ...	02/14/2023 ...	12h	0%	1.7.3
1.8.2	Integrate database	02/14/2023 ...	02/17/2023 ...	24h	0%	1.11.1
	⊕ Add a task | Add a milestone					
1.9	⊟ Model evaluation and testing	02/17/2023...	02/26/2023...	48h	0%	1.8.2
1.9.1	Evaluate models	02/17/2023 ...	02/22/2023 ...	24h	0%	1.8.2
1.9.2	Test models	02/22/2023 ...	02/26/2023 ...	24h	0%	1.9.1
	⊕ Add a task | Add a milestone					
1.10	⊟ Development of auto word suggestion feature	02/26/2023...	03/06/2023...	48h	0%	1.9.2
1.10.1	Develop feature	02/26/2023 ...	03/06/2023 ...	48h	0%	1.9.2
	⊕ Add a task | Add a milestone					
1.11	⊟ Final testing and integration	03/06/2023...	03/14/2023...	48h	0%	1.10.1
1.11.1	Test final product	03/06/2023 ...	03/14/2023 ...	48h	0%	1.10.1
	⊕ Add a task | Add a milestone					
	⊕ Add a task | Add a milestone					
	⊕ Add a task | Add a milestone					

Figure 3.2: Work Breakdown Analysis

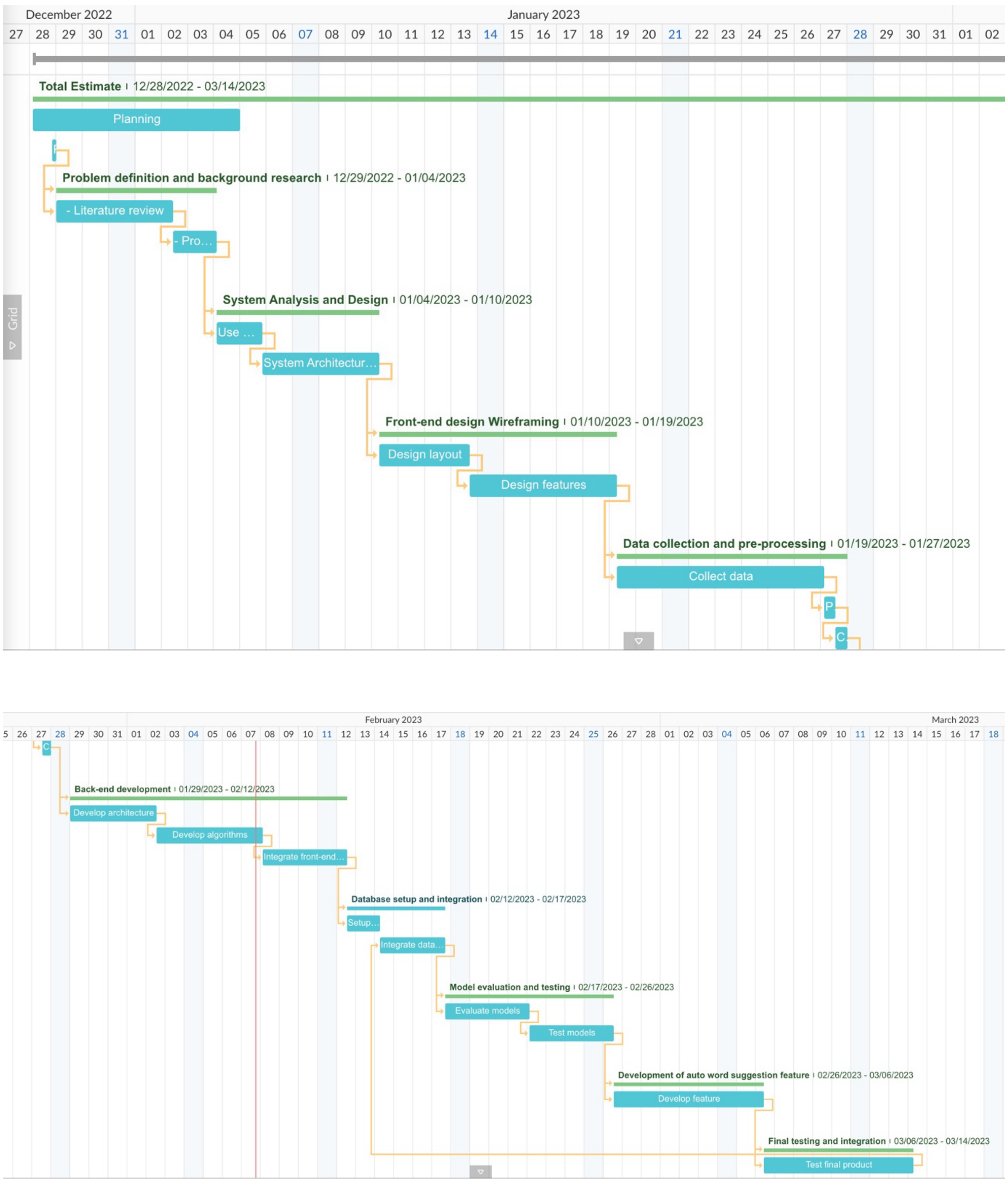


Figure 3.3: Gantt Chart

3.3 Software and Hardware Requirement

3.3.1 Software Requirement

The software that are required for the completion of our project are:

- Python Programming Language
- Python IDE/Jupyter notebook
- Visual Studio Code
- Windows/UNIX OS

3.3.2 Hardware Requirement

Following is the hardware requirement necessary for this project.

- Computing Device (Desktop/Laptop)

CHAPTER 4: SYSTEM DESIGN

4.1 System Flow Diagram

The input to the system is letter by the user. Then the input text was taken in account by the system. These inputs were preprocessed to generate the appropriate words that the user want to type.

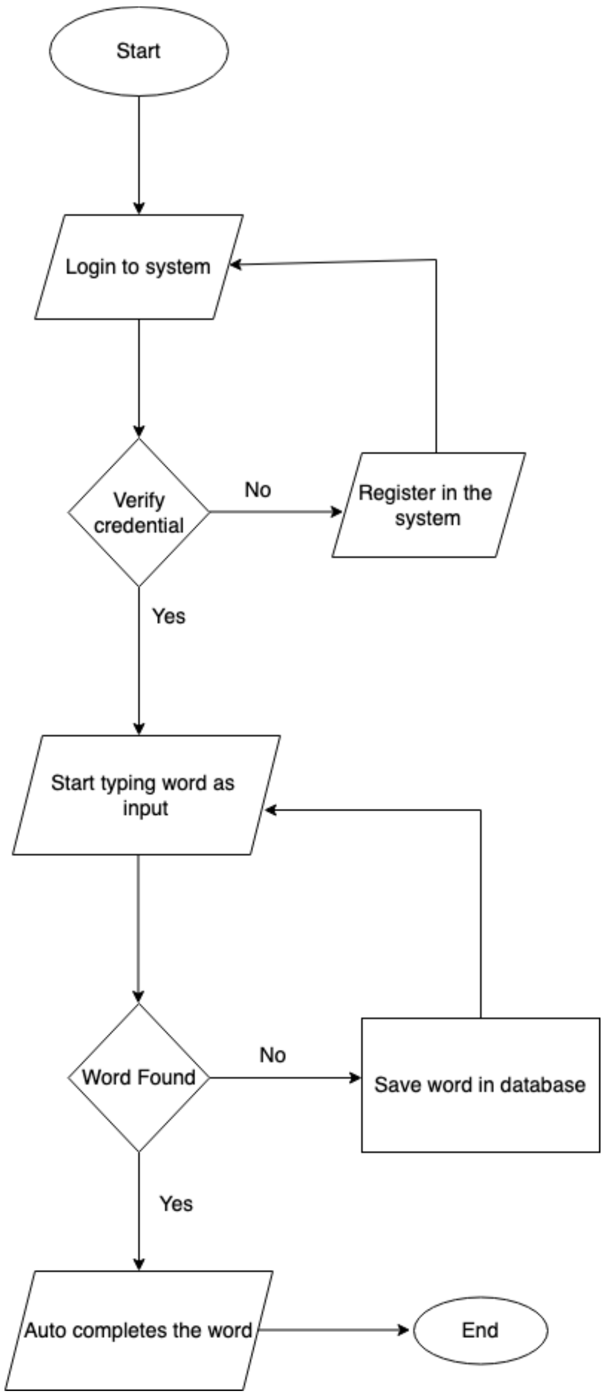


Figure 4.1: System Flow Diagram

4.3 Activity Diagram

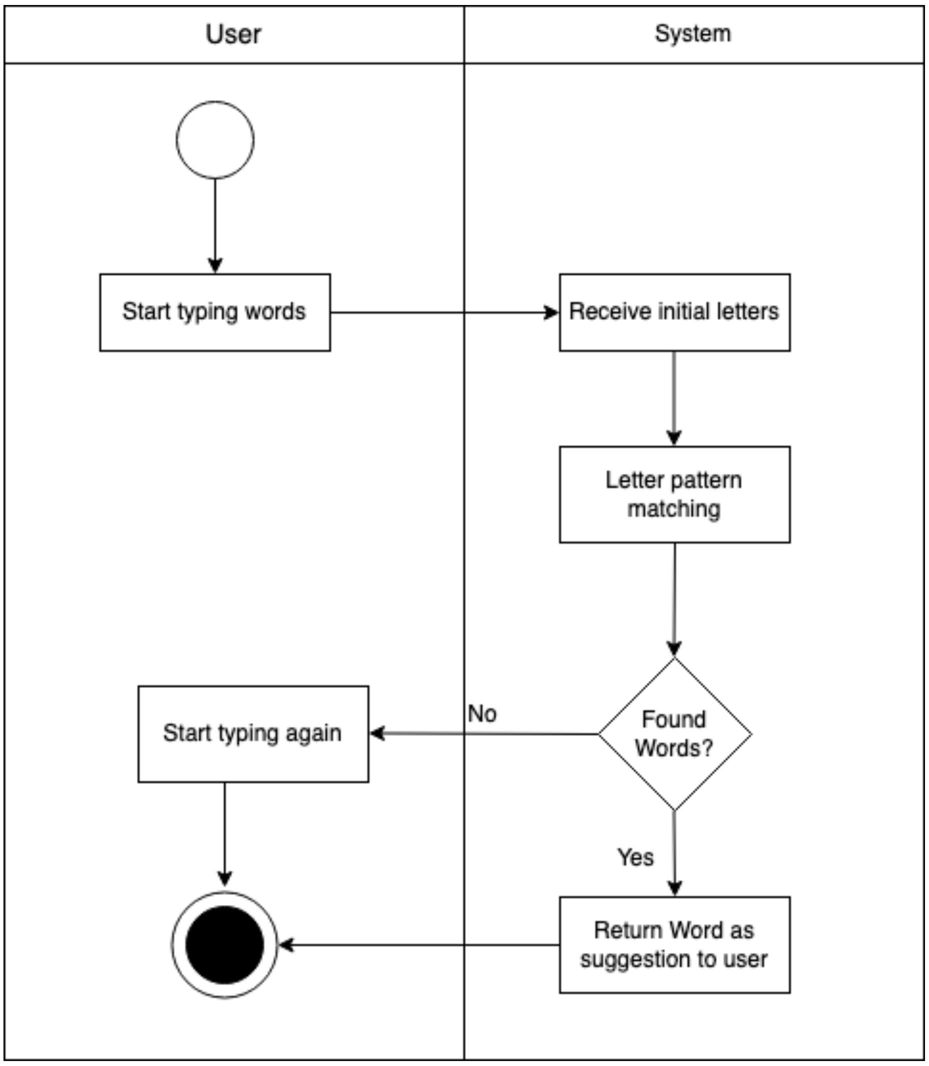


Figure 4.2: Activity Diagram

Take into consideration the word that suggests activity of the system for the activity diagram. The user begins entering a word into the input field. The system receives the word's prefix or first few letters. To see if there are any terms that match, the machine consults the dictionary. If there are no words that match, the word predictor produces a null value, the input box shows no predictions, and the user must start typing again. If there are matching words, the word predictor determines the likelihood of each potential next word based on the frequency of each word in the dictionary. The user receives the most likely following word from the word predictor.

4.2 Sequence Diagram

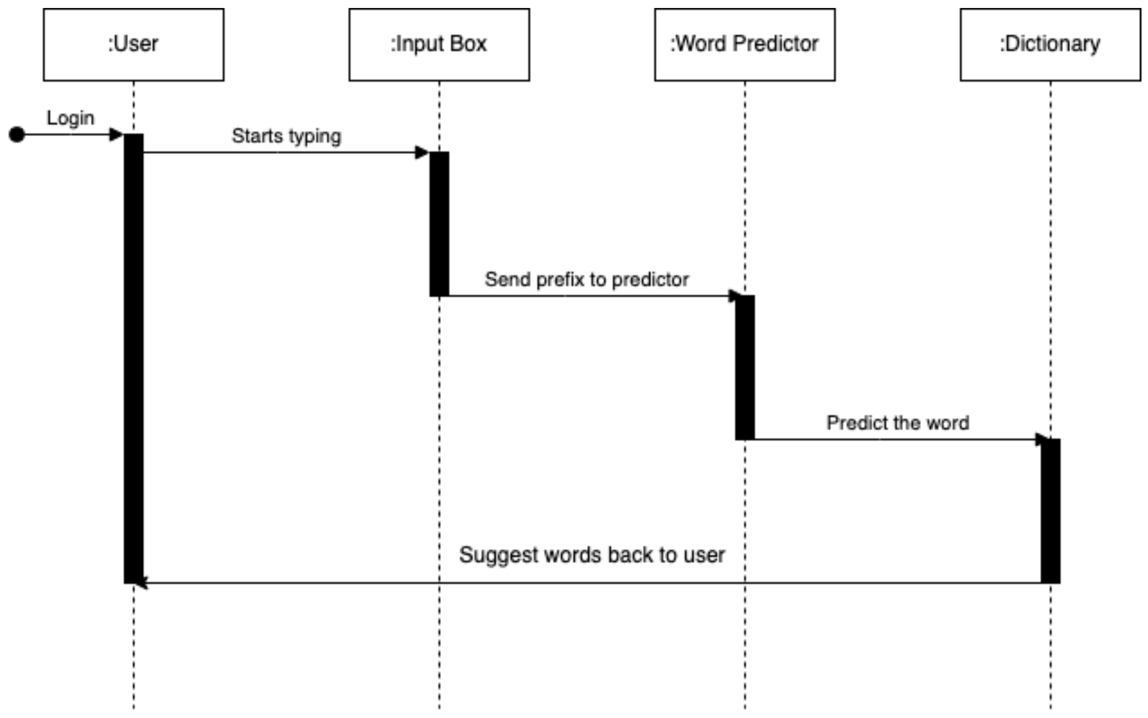


Figure 4.3: Sequence Diagram

After checking in to the system, the user begins to type a word into the input area. The input box delivers the text to the word predictor as the user inputs it. The word predictor bases its predictions on the input and consults a dictionary. The predicted word is returned to the user by the word predictor. The user can see the suggested terms in the input area.

4.4 Machine Learning System

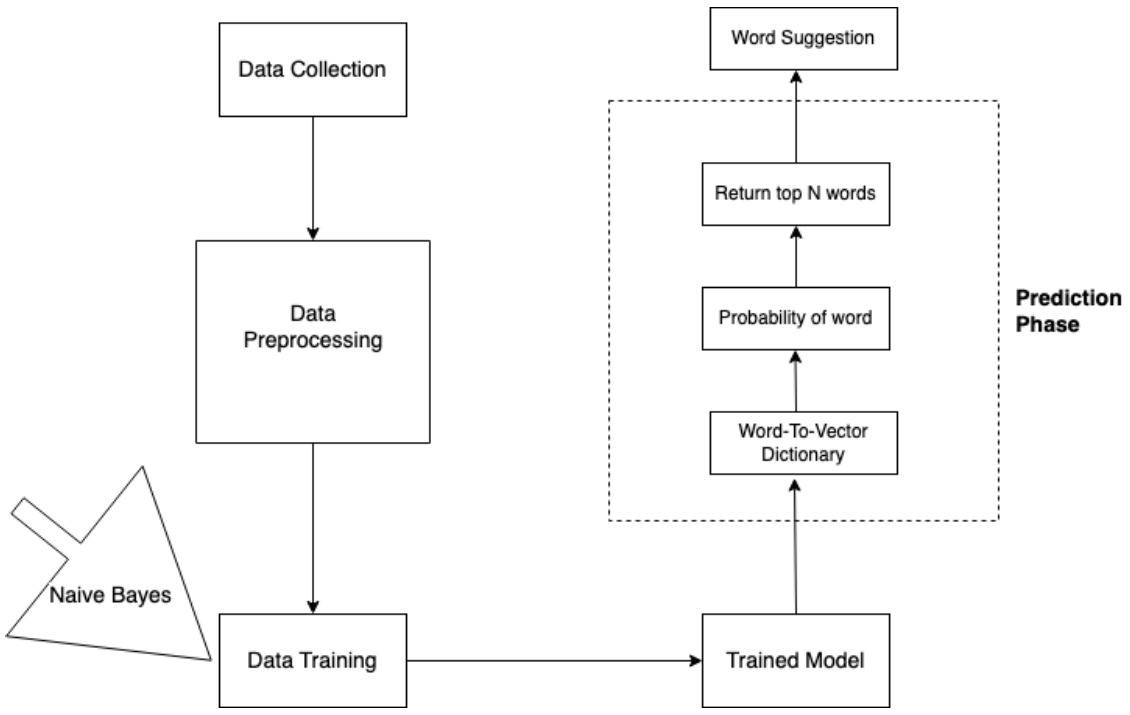


Figure 4.4: Machine Learning System

The training phase and the prediction phase are the system's two main parts. The Naive Bayes model is trained on a collection of words and their accompanying labels during the training phase. The frequency of each character in the word serves as the model's input variable, and the output variables are the likelihoods that the word belongs to each class. The Nave Bayes model is used by the system to forecast the words that are most likely to begin with a given prefix when it is in the prediction phase. It begins by employing a predefined word-to-vector function to transform the prefix into a vector of character frequencies. The probability of each potential class is then determined using the Naive Bayes model. The predicted words are chosen from the top N words with the highest probabilities.

CHAPTER 5: SYSTEM IMPLEMENTATION AND TESTING

5.1 Overview

The formal software development process is known as system implementation. The process of planning, building, testing, and maintaining various software applications is known as software development. It necessitates the application of several engineering, computer science, and mathematical analytical principles and techniques. Software development seeks to produce reliable, effective, and simple software.

5.2 Programming Tools

Any software program or utility that helps software developers or programmers create, modify, debug, maintain, or carry out any task relevant to programming or development is referred to as a "programming tool." Software development tools are another name for programming tools. The frontend and backend of an application are created using various technologies, and when they are logically combined, we are left with a finished product. Programming tools include code editors, linkers, assemblers, load testers, performance analysts, compilers, and linkers.

5.2.1 Front End

Front end development is a field of computer programming that deals with coding and developing website features and elements that consumers will actually view. The objective is to ensure that a website's visual elements are useful. It resembles the application's client side.

The front-end tool sets utilized in this project are: - HTML5 - CSS3 - JavaScript/ReactJS

5.2.2 Back End

The portion of a website that visitors cannot view is the subject of back-end development. It is what gives a website its interactivity. A website's back end is also referred to as the "server side" of a website. It is the system component that typical users don't want to be aware of.

The following sets of back-end tools are utilized in this project:

- NodeJs - ExpressJS - MongoDB

5.3 Algorithm Implementation

Based on the likelihood that they will appear in the dataset, the algorithm predicts the words that are most likely to appear following an input string. According to its frequency in previously retrieved words, each word's likelihood is updated, and the final forecast is a list of the top n words with their accompanying probabilities.

Within the set of classes C, for each class c:

1) Determine the class probability P(c)'s logarithm: log(P(c)).

2) For each feature x in the input vector X, compute the logarithm of the conditional probability P(x|c) using the formula log(P(x|c)).

3) Determine the conditional probability' logarithmic total for each class: Σ[log(P(x|c))]

4) Determine the class probability by multiplying it by the sum of the logarithms of the conditional probabilities, as follows: P(c) * exp[[log(P(x|c))]].

Divide the probability for each anticipated word by the total number of probabilities to normalize them: P(c|x) is equal to P(c)*exp[[log(P(x|c))]] and P(c')*exp[[log(P(x|c'))]]. for every class

The top n predicted words together with their normalized probability should be returned.

5.4 Testing

Software testing is the process of confirming and validating if our application satisfies the technical requirements as established by its development and design, as well as the user requirements. It basically aims to evaluate the specification, functionality, and performance of a piece of software or an application.

5.4.1 Test Plan

Testing planning, which comprises defining the objectives, constraints, and process of testing activities, is an essential part of software quality assurance (SQA). The following phases are often included in test planning, which is an iterative process:

- Requirements analysis:
- Test strategy development
- Test case development
- Test execution

The top n most likely words will be predicted by the word prediction algorithm based on user input and previously recorded word counts. The system uses a JSON API for input and output and a Naive Bayes model for prediction.

Test Environment:

The tests will be run on a local machine with Python 3.9 and nodejs installed.

5.4.1.1 Unit Testing

Individual pieces of source code—sets of one or more computer program modules—are tested as a part of the software testing technique known as unit testing. The unit testing can be done for following modules:

- word_to_vector()
- preprocess_input()
- Naivebayes_fit()
- Naivebayes_predict()

5.4.1.2 Integration Testing

The system's many parts will all be put through integration tests to make sure they function as intended. The following will be tested:

- JSON API input validation and response formatting
- Model loading and initialization
- Model training and prediction
- Fetched words processing and prediction update
- User Login and Logout

5.4.1.3 System Testing

To ensure that the system complies with the specifications and acts as intended, system tests will be run. The following will be tested:

- System login and logout
- User input processing and prediction
- Prediction accuracy and relevance
- Prediction update based on fetched words

5.4.2 Test Case

Test cases are a set of instructions or rules that describe how to determine whether a specific feature or function of a software program is working as intended. The fundamental objective of test cases is to ensure that the program meets the necessary requirements and gives users what they want. The test cases should be written so that testers can easily comprehend and apply them.

Here are some of the test cases of our system:

Table 5.1: Test Case for User Registration

Test case ID	Test Scenario	Testing steps	Input Test data	Expected Result	Actual Result	Pass /Fail
TC-01	User register with any empty input	1 Open the system. 2.Fill the fields 3.Enter Register	Username: nishant Email: nishan@gmail.com Password: Confitm Password:	Registration failed with message all the field must be filled	As expected	Pass
TC-02	User register with password mistake	1 Open the system. 2.Fill the fields 3.Enter Register	Username: nishant Email: nishan@gmail.com Password: nishant123 Confirm Password : nishan123	Register failed with message passwords doesn't match	As expected	Pass
TC-03	User register with all valid information	1 Open the system. 2.Fill the fields 3.Enter Register	Username: nishant Email:nishan@gmail.com Password: nishant123 Confirm Password : nishant123	Register successful	As expected	Pass

Table 5.2: Test Case for User Login

Test case ID	Test Scenario	Testing steps	Input Test data	Expected Result	Actual Result	Pass /Fail
TC-01	User login with valid credentials	1 Open the system. 2.Fill the fields 3.Enter Login	Username: nepal Password: nepal123	User successfully logs into the system	As expected	Pass
TC-02	User login with invalid credentials	1 Open the system. 2.Fill the fields 3.Enter Login	Username: nishant Password: nishant1	Login failed with invalid "username and password" message	As expected	Pass

Table 5.3: Test Case for Search Bar

Test case ID	Test Scenario	Testing steps	Input Test data	Expected Result	Actual Result	Pass/ Fail
TC-01	User start typing in search bar	1. Login to system 2.Start searching in bar	Any valid word that can be searched example "mon"	Suggestion starts to show as dropdown while typing "money" "monkey"	As expected	Pass
TC-02	User types invalid word or something that doesn't makes sense	1. Login to system 2.Start searching in bar	Meaningless words that doesn't make sense "astbsd"	No suggestion for user in bar	As expected	Pass
TC-03	User selects the word that is suggested	1.Login to system 2.Start searching in bar 3.Selects suggested word	Prefix of a certain word	Auto completes the word	As expected	Pass

CHAPTER 6: CONCLUSION

A program called a word prediction application uses the user's input text to propose the next word or words for them. In order to forecast the likelihood of the subsequent word based on the user's previously input words, the program employs a Naive Bayes classifier. To determine the likelihood of each word in the training data, the algorithm consults a dictionary of words and their frequencies. Each word's probability is determined independently before being added together to determine the likelihood of the entire input string. The software then displays the top n most likely terms, where n is a user-defined value.

6.1 Limitations

Despite making an effort to create a perfect system, the following limitations of the application can be upgraded in future versions:

• Lack of context: The system makes predictions without taking into account the words' surrounding sentences.

• Non-English Words: Since the algorithm has only been trained on English words, it might not be able to anticipate non-English terms.

6.2 Future Enhancements

• Expand the training dataset's size and diversity: A larger, more varied training dataset can aid the system in better capturing the complexities of the language and enhancing its predicting accuracy.

• Take into account context: The algorithm can produce more accurate predictions by taking into account context, such as the words that come before or the part of speech.

REFERENCES

[1] "Guru99," 2023. [Online]. Available: https://www.guru99.com/unit-test-vs-integration-test.html.

[2] "Advances in NLP applied to Word Prediction," [Online]. Available: https://www.researchgate.net/publication/228822071_Advances_in_NLP_applied_to_Word_Prediction/link/00b4952b14e64c4d89000000/download.

[3] J. H. a. K. F. David Ward, Autocomplete as a Research Tool: A Study on Providing Search Suggestions, Illianos, USA, 2011.

[4] "Hybrid Model For Word Prediction Using Naive," [Online]. Available: https://arxiv.org/pdf/1803.00985.pdf.

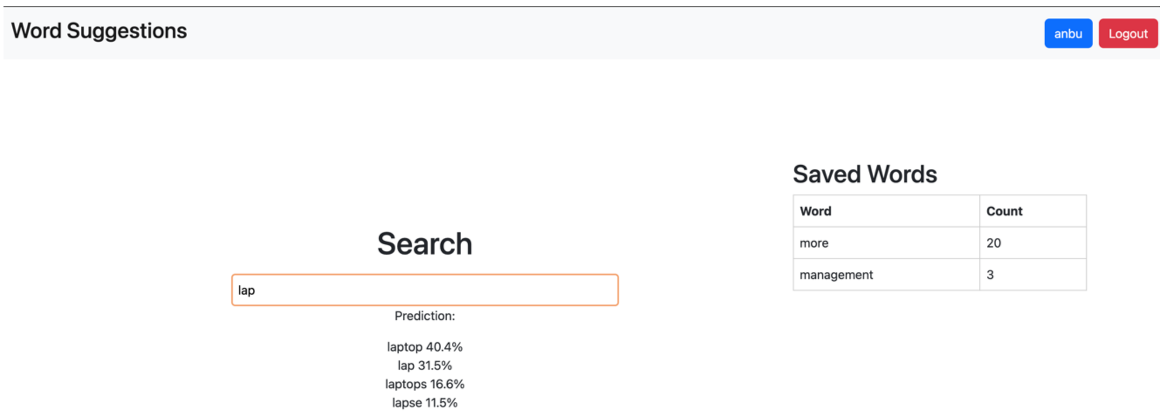
[5] "Next Word Prediction with NLP and Deep Learning," [Online]. Available: https://towardsdatascience.com/next-word-prediction-with-nlp-and-deep-learning-48b9fe0a17bf.

[6] "Science Direct," 2020. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/incremental-model.

[7] "Mongodb," [Online]. Available: https://www.mongodb.com/scale/nosql-database-implementation.

APPENDIX

A. SCREENSHOTS



Saved Words

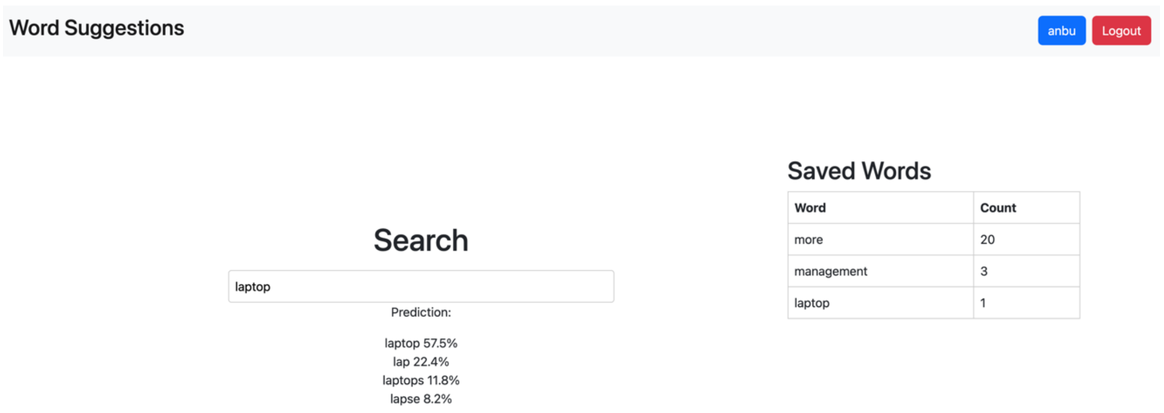
Word	Count
more	20
management	3

Search

lap

Prediction:

laptop 40.4%
lap 31.5%
laptops 16.6%
lapse 11.5%



Saved Words

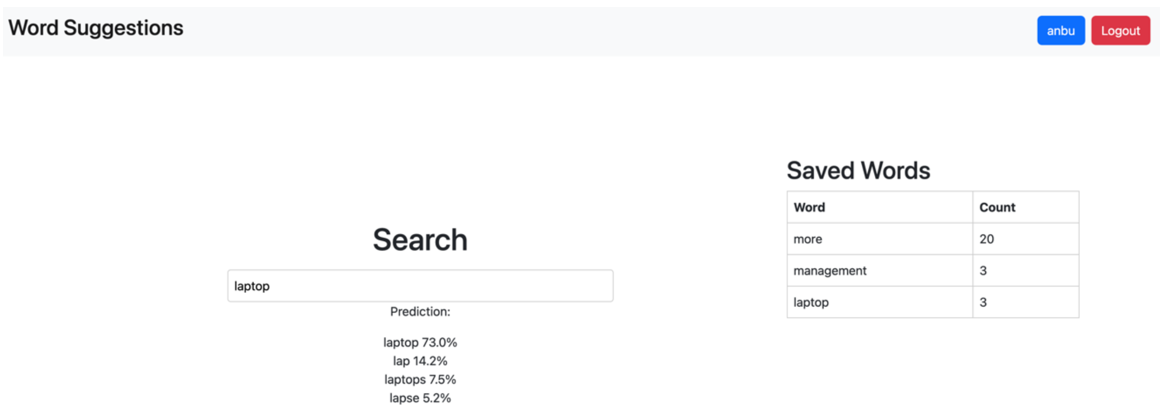
Word	Count
more	20
management	3
laptop	1

Search

laptop

Prediction:

laptop 57.5%
lap 22.4%
laptops 11.8%
lapse 8.2%



Search

laptop

Prediction:

laptop 73.0%
lap 14.2%
laptops 7.5%
lapse 5.2%

Saved Words

Word	Count
more	20
management	3
laptop	3

B. MODEL SOURCE CODE

predict_word.py

```
class NaiveBayes:
    def __init__(self, classes):
        self.classes = classes
        self.class_probs = None
        self.cond_probs = None

    def fit(self, X, y, counts):
        n, d = X.shape
        self.class_probs = np.zeros(len(self.classes))
        self.cond_probs = np.zeros((len(self.classes), d))

        for i, c in enumerate(self.classes):
            X_c = X[y == c]
            n_c = counts[y == c].sum()
            self.class_probs[i] = n_c / counts.sum()
            self.cond_probs[i] = (X_c.sum(axis=0) + 1) / (X_c.sum() + d)

    def predict(self, X):
        probs = np.zeros((len(X), len(self.classes)))

        for i, x in enumerate(X):
            for j, c in enumerate(self.classes):
                probs[i, j] = np.log(self.class_probs[j]) + (np.log(self.cond_probs[j]) * x).sum()

        return self.classes[np.argmax(probs, axis=1)]

    def predict_top_n_words(self, X, prefix, n=4):
        probs = np.zeros((len(X), len(self.classes)))
        found_prefix = False
        for i, x in enumerate(X):
            for j, c in enumerate(self.classes):
                if c.startswith(prefix):
                    found_prefix = True
                    probs[i,   j]   =   np.log(self.class_probs[j])   +   (np.log(self.cond_probs[j])   *
x).sum()
                else:
                    probs[i, j] = -np.inf

        if not found_prefix:
            return "no prediction available"

        top_n_indices = np.argsort(-probs, axis=1)[:, :n]
        top_n_words = self.classes[top_n_indices][0]
        top_n_probs = np.exp(probs[0][top_n_indices[0]])
        top_n_probs = top_n_probs / top_n_probs.sum()
        filtered_top_n = list(zip(top_n_words, top_n_probs))
        filtered_top_n = [t for t in filtered_top_n if not np.isnan(t[1])]
        return filtered_top_n if top_n_probs.sum() != 0 else []

def word_to_vector(word, word_to_index):
    vec = np.zeros(len(word_to_index), dtype=np.int32)
    for char in word:
        if char in word_to_index:
            vec[word_to_index[char]] += 1
return vec
def preprocess_input(input_str, word_to_index):
    return np.array([word_to_vector(input_str, word_to_index)])

def predict(input_str, model, word_to_index, n=4):
    preprocessed_input = preprocess_input(input_str, word_to_index)
    prediction = model.predict_top_n_words(preprocessed_input, input_str, n)
    return prediction if prediction else None
```

```
if __name__ == "__main__":
    input_str = sys.argv[1]

    # Load the model
    model_file_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
"naive_bayes_model.pkl")
    with open(model_file_path, "rb") as model_file:
        model = pickle.load(model_file)

    # Load the word_to_index dictionary
    word_to_index_file_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
"word_to_index.pkl")
    with open(word_to_index_file_path, "rb") as word_to_index_file:
        word_to_index = pickle.load(word_to_index_file)



    # Make prediction and return output
    prediction = predict(input_str, model, word_to_index)
    if prediction:
        print(json.dumps(prediction))
    else:
        print("no prediction available")
```