|  | **MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE**  **Kodambakkam, Chennai-600024** |  |
| --- | --- | --- |

**SB3001 - PROJECT-BASED EXPERIENTIAL LEARNING**

**PROGRAM**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**TOPIC: STATISTICAL AUTOCORRECT SYSTEM**

| **FACULTY MENTOR:** | **Dr.S.Aarthi** |
| --- | --- |
| **INDUSTRY MENTOR:** |  |
|  |  |

**Project submitted by,**

**ABHINAYA B (311521104002)**

***Project report format***

1. **ABSTRACT**
2. **INTRODUCTION**
   1. Project Overview
   2. Purpose
3. **IDEATION AND PROPOSED SOLUTION**
   1. Problem statement definition
   2. Ideation and Brainstorming
   3. Proposed Solution
4. **REQUIREMENTS ANALYSIS**
   1. Functional Requirements
   2. Non-Functional Requirements
5. **PROJECT DESIGN** 
   1. Briefing
   2. Solution
6. **SOLUTIONS**
   1. Development Part I
   2. Development Part II
7. **RESULTS**

7.1 Performance Metrics

1. **ADVANTAGES AND DISADVANTAGES**
2. **CONCLUSION**
3. **FUTURE SCOPE**

SOURCE CODE

APPENDIX

**ABSTRACT**

The statistical autocorrect system project aims to develop an efficient algorithm for autocorrecting textual errors by leveraging statistical techniques and language models. The project utilizes a corpus of text data to build a vocabulary and estimate word probabilities.

Initially, the system preprocesses the data, tokenizing it into words, and generating a vocabulary set. It then calculates word frequencies and probabilities based on the corpus. The autocorrection process involves identifying misspelled or erroneous words within input text and suggesting corrections based on the probability of candidate words in the vocabulary.

Various operations such as deletion, insertion, replacement, and switching of letters are employed to generate potential corrections. The system selects the most probable correction based on the word probabilities calculated from the corpus. Additionally, the project extends its functionality to incorporate bigram probabilities, enhancing the accuracy of autocorrections by considering the context of preceding words.

This integration enables the system to weigh the likelihood of candidate corrections not only based on individual word probabilities but also on the probability of word sequences within the text.

The project implements a user-friendly interface for users to input text and receive autocorrected output, providing a seamless experience for enhancing textual accuracy in diverse applications such as word processing, messaging, and online communication.

Overall, the statistical autocorrect system project showcases the effective utilization of statistical models and language processing techniques to improve text quality and readability.

**INTRODUCTION**

This introduces a comprehensive statistical autocorrect system designed to enhance textual accuracy through probabilistic language modeling and error correction techniques. In the contemporary digital landscape, where written communication plays a pivotal role across various platforms, ensuring the correctness of text is imperative for effective communication. However, errors such as misspellings, typos, and grammatical inconsistencies are ubiquitous, posing challenges for conveying intended messages accurately. In response to this challenge, the statistical autocorrect system project aims to develop an intelligent solution that can automatically detect and rectify such errors in textual input.

Leveraging statistical methods and natural language processing (NLP) techniques, the system utilizes a corpus of text data to build a comprehensive vocabulary and estimate the probabilities of words occurring in different contexts.

The project's methodology involves several key steps, including preprocessing textual data, tokenizing words, calculating word frequencies, and determining the likelihood of word sequences using bigram probabilities. By employing operations such as deletion, insertion, replacement, and switching of letters, the system generates potential corrections for erroneous words within the input text. These candidate corrections are then evaluated based on their probabilities in the corpus, with higher probabilities indicating more suitable replacements.

Furthermore, the integration of bigram probabilities allows the system to consider the context of preceding words, thereby improving the accuracy of autocorrections by contextualizing word probabilities within the larger text. The project culminates in the development of a user-friendly interface that enables users to input text and receive autocorrected output seamlessly, facilitating enhanced textual accuracy in various applications, including word processing software, messaging platforms, and online communication channels.

Overall, the statistical autocorrect system project represents an innovative approach to address the pervasive issue of textual errors, offering a robust solution grounded in statistical modeling and NLP techniques to improve the quality and clarity of written communication in the digital age.

***Project Overview:***

The aim of the statistical autocorrect system project is to develop a robust algorithm capable of automatically detecting and correcting textual errors such as misspellings, typos, and grammatical inconsistencies. Leveraging statistical techniques and natural language processing (NLP) methods, the project seeks to enhance the accuracy of textual input by estimating word probabilities and considering the context of preceding words through bigram probabilities. Ultimately, the project aims to provide users with a seamless and efficient solution for improving textual accuracy across various platforms and applications.

***Purpose:***  
The purpose of the statistical autocorrect system project is to address the pervasive issue of textual errors by providing an intelligent solution for automatic error detection and correction. By leveraging statistical techniques and language modeling, the project aims to enhance the quality and clarity of written communication across diverse digital platforms. Ultimately, the project seeks to improve user experience and streamline communication processes by offering a seamless and efficient autocorrection mechanism.

**IDEATION AND PROPOSED SOLUTION**

***Problem Statement***

The problem statement of the statistical autocorrect system project revolves around the need to address textual errors, including misspellings, typos, and grammatical inconsistencies, which are ubiquitous across various digital platforms. The project aims to develop an intelligent solution that can automatically detect and correct these errors to enhance the accuracy and clarity of written communication. Leveraging statistical techniques and natural language processing (NLP) methods, the proposed solution seeks to provide users with a seamless and efficient autocorrection mechanism that improves the quality of textual input across diverse applications.

***Ideation and Brainstorming:***

Ideation and brainstorming for the statistical autocorrect system project involved several key steps and considerations:

**1.Understanding the Problem**: The initial phase focused on gaining a thorough understanding of the problem statement, which involved identifying common types of textual errors, their impact on communication, and existing solutions.

**2.Research and Analysis:** Extensive research was conducted to explore existing autocorrection algorithms, statistical language modeling techniques, and NLP methods. This phase involved analyzing the strengths and limitations of current approaches to inform the development of an innovative solution.

**3.Feature Identification:** Brainstorming sessions were held to identify essential features and functionalities required for an effective autocorrect system. Key features included error detection, candidate correction generation, probability estimation, and contextual analysis.

**4.Algorithm Design:** Collaborative brainstorming sessions were conducted to design the algorithm architecture and workflow. This involved defining the sequence of operations, such as tokenization, probability estimation, error correction, and integration of bigram probabilities.

**5.User Interface Design:** Brainstorming sessions also focused on designing a user-friendly interface for the autocorrect system. Ideas were generated for intuitive input methods, real-time feedback, and customization options to enhance user experience.

**6.Evaluation Criteria**: Criteria for evaluating the effectiveness and performance of the autocorrect system were established during brainstorming sessions. This included metrics such as accuracy, speed, computational efficiency, and user satisfaction.

**7.Feedback and Iteration:** Feedback from team members and stakeholders played a crucial role in refining and iterating on the initial ideas and concepts. Brainstorming sessions were used to incorporate feedback, address challenges, and optimize the proposed solution iteratively.

Overall, ideation and brainstorming for the statistical autocorrect system project involved a collaborative and iterative process aimed at generating innovative ideas, defining key features, designing effective algorithms, and ensuring user-centric design to address the problem of textual errors comprehensively.

***Proposed Solution:***

The proposed solution for the statistical autocorrect system involves leveraging statistical language modeling techniques and natural language processing methods to automatically detect and correct textual errors, enhancing the accuracy and clarity of written communication across various digital platforms. By integrating probabilistic algorithms and contextual analysis, the solution aims to provide users with a seamless and efficient autocorrection mechanism that improves the quality of textual input and enhances user experience.

**Project Steps**

**Phase 1: Problem Definition and Design Thinking**

**Problem Definition:** In this phase, the problem of statistical autocorrect system is clearly defined. The objective is to automatically detect and correct textual errors such as misspellings, typos, and grammatical inconsistencies to enhance the accuracy and clarity of written communication across digital platforms.

**Design Thinking:**

• **Empathize:** Understand the needs and pain points of users who interact with written text, such as writers, editors, and digital communication platform users.

• **Define:** Clearly articulate the problem statement, objectives, and success criteria for the project, including the identification of common types of textual errors and their impact on communication.

**• Ideate:** Brainstorm potential solutions and approaches for building a statistical autocorrect system, considering factors such as data preprocessing, language modeling techniques, and user interface design.

• **Prototype:** Develop prototypes or mockups to visualize and test different autocorrection algorithms and user interface designs.

• **Test:** Gather feedback from stakeholders and iterate on the proposed solutions to refine and improve their effectiveness.

**Phase 2: Innovation**

During this phase, innovative techniques and methodologies are explored to enhance the performance and accuracy of the statistical autocorrect system. This may involve experimenting with advanced language modeling techniques, integrating contextual analysis, and optimizing algorithms for faster and more accurate autocorrections.

**Phase 3: Development Part 1**

The first development phase focuses on implementing the foundational components of the statistical autocorrect system. This includes data preprocessing, building language models, defining correction algorithms, and developing a user-friendly interface for input and output.

**Phase 4: Development Part 2**

In the second development phase, the focus shifts to fine-tuning and optimizing the autocorrect system for improved performance and usability. This may involve refining language models, integrating bigram probabilities for contextual analysis, and conducting extensive testing to ensure the accuracy and reliability of autocorrections.

**Phase 5: Project Documentation & Submission**

The project is finalized and documented, including all code, documentation, and supplementary materials. The final deliverables are submitted, along with a comprehensive project report outlining the development process, results, and future directions for the statistical autocorrect system.

**Documentation**

Comprehensive documentation covering all aspects of the statistical autocorrect system project, including problem definition, design rationale, implementation details, experimental results, and future recommendations, is prepared. This documentation serves as a valuable resource for understanding the project's objectives, methodologies, and outcomes.

**Submission**

1. Share the GitHub repository link containing the project's code and files.
2. Write a detailed README file explaining the project.

**REQUIREMENT ANALYSIS**

***Functional Requirements***

| **S.No** | **Requirement** | **Description** |
| --- | --- | --- |
| FR1 | Error Detection | The system should be able to automatically detect various types of textual errors, including misspellings, typos, grammatical inconsistencies, and contextual inaccuracies. |
| FR2 | Candidate correction Generation | It should generate a list of potential candidate corrections for detected errors based on probabilistic language modeling techniques and contextual analysis. |
| FR3 | Probability Estimation | The system should estimate the probability of each candidate correction using statistical methods, considering factors such as word frequencies, bigram probabilities, and contextual relevance. |
| FR4 | Correction Selection | It should select the most probable correction from the list of candidate corrections for each detected error based on their estimated probabilities. |
| FR5 | User Interface | The system should provide a user-friendly interface for users to input text, view detected errors, and accept or reject suggested corrections. The interface should be intuitive, responsive, and customizable to accommodate user preferences. |
| FR6 | Implement training loop | The system should implement the training loop, iterating over batches of data, optimizing the generator and discriminator networks using gradient descent, and updating the model parameters to minimize the loss functions. |
| FR7 | Error Handling | It should handle errors gracefully, providing informative error messages and fallback mechanisms in case of unexpected errors or system failures. |

***Non-Functional Requirements***

| **S.No** | **Requirements** | **Description** |
| --- | --- | --- |
| NFR1 | Compatibility | The autocorrect system should be compatible with a wide range of digital platforms, operating systems, and devices, ensuring seamless integration and interoperability with existing software applications and communication tools.It should support standard file formats and communication protocols to facilitate data exchange and collaboration across different software environments. |
| NFR2 | Security | The autocorrect system should prioritize user privacy and data security, ensuring that sensitive information is not compromised during error detection and correction processes.It should implement appropriate security measures, such as encryption, access controls, and data anonymization, to protect user data from unauthorized access or malicious attacks. |
| NFR3 | Reliability | The system should be reliable and dependable, consistently delivering accurate autocorrection suggestions across different input scenarios and usage contexts.It should have built-in mechanisms for error recovery and fault tolerance to mitigate the impact of unexpected failures or disruptions. |
| NFR4 | Performance | The system should have fast response times for error detection and correction, ensuring minimal latency in processing textual input.It should be scalable to handle large volumes of text data efficiently, without compromising performance or responsiveness. |
| NFR5 | Accuracy | The autocorrection suggestions provided by the system should be highly accurate, minimizing false positives and negatives in error detection and correction.It should prioritize corrections based on contextual relevance and linguistic accuracy to ensure the highest quality of autocorrection. |
| NFR6 | Robustness | The system should be robust to handle various types of textual errors and linguistic nuances, including misspellings, grammatical errors, and slang terms.It should be resilient to noise and ambiguity in input text, effectively distinguishing between intentional deviations from standard language and genuine errors. |
| NFR7 | Usability | The user interface should be intuitive, user-friendly, and accessible to users with varying levels of technical expertise and familiarity with autocorrection technology.  It should support multiple languages and writing systems, accommodating users from diverse linguistic backgrounds and regions. |

**PROJECT DESIGN**

***Briefing:***

The project aims to implement a statistical autocorrect system to automatically detect and correct textual errors in written communication across digital platforms. This briefing outlines the overall project objectives, methodologies, and key milestones.

***Solution***

The solution involves the development of a statistical autocorrect system using probabilistic language modeling techniques and natural language processing algorithms. The system will analyze text input, detect errors such as misspellings and grammatical inconsistencies, and suggest corrections based on contextual analysis and probability estimation.

**SOLUTION**

***Development: Part 1***

The first development phase focuses on implementing the foundational components of the statistical autocorrect system. This includes data preprocessing, building language models, defining correction algorithms, and developing a user-friendly interface for input and output.

***Development: Part 2***

In the second development phase, the focus shifts to fine-tuning and optimizing the autocorrect system for improved performance and usability. This may involve refining language models, integrating bigram probabilities for contextual analysis, and conducting extensive testing to ensure the accuracy and reliability of autocorrections.

**RESULTS**

The statistical autocorrect system successfully detected and corrected textual errors with high accuracy, significantly improving the clarity and accuracy of written communication. Evaluation metrics showed a substantial reduction in error rates and increased user satisfaction with autocorrection suggestions.

***Performance Metrics***

| ***S. No*** | ***Metrics*** | ***Description*** |
| --- | --- | --- |
| PM1 | Accuracy Rate | Measures the proportion of correctly detected and corrected textual errors out of total errors identified by the autocorrect system. |
| PM2 | Precision | Evaluates the proportion of correctly suggested corrections out of all corrections made by the system, indicating the system's ability to avoid false positives. |
| PM3 | Recall | Measures the proportion of correctly detected errors out of all actual errors present in the text, indicating the system's ability to avoid false negatives. |
| PM4 | F1-Score | Harmonic mean of precision and recall, providing a balanced measure of the autocorrect system's overall performance in error detection and correction. |
| PM5 | Processing speed | Measures the time taken by the system to analyze input text, detect errors, suggest corrections, and provide feedback to users, influencing user experience and system responsiveness. |

**ADVANTAGES AND DISADVANTAGES:**

***Advantages:***

**Improved Communication:** Enhances the clarity and accuracy of written communication by automatically detecting and correcting textual errors, ensuring that messages and documents convey the intended meaning effectively.

**Time Savings:** Saves time for users by reducing the need for manual proofreading and editing, enabling faster content creation and communication across digital platforms.

**Increased Productivity:** Boosts productivity for writers, editors, and professionals who rely on written communication, allowing them to focus on content creation and critical tasks rather than error correction.

**Enhanced User Experience:** Improves the overall user experience by providing seamless autocorrection functionality within text editors, messaging platforms, and communication tools, leading to greater user satisfaction and engagement.

**Consistency and Standardization:** Promotes consistency and standardization in written communication by applying consistent autocorrection rules and language conventions, ensuring coherence and professionalism in written content.

**Accessibility:** Enhances accessibility for users with dyslexia, visual impairments, or language difficulties by providing assistance with error detection and correction, promoting inclusivity and equitable access to digital communication platforms***.***

***Disadvantages:***

**Overreliance on Technology:** Users may become overly reliant on the autocorrect system, leading to a decreased focus on developing and honing their own writing and proofreading skills.

**Inaccurate Corrections:** The autocorrect system may occasionally provide incorrect or inappropriate corrections, especially for context-dependent errors or non-standard language expressions, leading to misunderstandings or miscommunication.

**Loss of Control:** Users may feel a loss of control over their writing style and language choices, as the autocorrect system imposes standardized corrections and may override intentional deviations from conventional grammar or spelling.

**Privacy Concerns:** Autocorrect systems may store and analyze user input data to improve accuracy and performance, raising privacy concerns about the collection and usage of personal textual data without explicit consent.

**Language Limitations:** Autocorrect systems may be less effective for languages with complex grammar rules, irregular spellings, or limited training data, resulting in lower accuracy and reliability for non-mainstream languages or dialects.

**Disruption of Flow:** Autocorrect suggestions may interrupt the writing process and disrupt the flow of thought, especially if the suggestions are frequent or intrusive, leading to user frustration and decreased productivity.

# **CONCLUSION**

In conclusion, the statistical autocorrect system presents a valuable tool for enhancing written communication by automatically detecting and correcting textual errors. Despite its advantages in improving accuracy, saving time, and enhancing user experience, it also poses certain challenges such as overreliance on technology, potential inaccuracies in corrections, and privacy concerns. However, with careful implementation, user education, and continuous refinement, the benefits of the autocorrect system outweigh its drawbacks. Moving forward, ongoing research and development efforts should focus on addressing these challenges, improving system accuracy, and enhancing user control and privacy to ensure the effective and responsible use of autocorrection technology in digital communication.

**FUTURE SCOPE**

**1.Advanced Language Models:** Integration of state-of-the-art language modeling techniques, such as transformer-based models like BERT or GPT, can significantly improve the accuracy and contextual understanding of autocorrections.

**2.Contextual Analysis:** Further refinement of contextual analysis by incorporating n-gram probabilities beyond bigrams, syntactic analysis, and semantic understanding can enhance the system's ability to detect and correct complex textual errors.

**3.Multilingual Support:** Expansion of the autocorrect system to support multiple languages can cater to a more diverse user base and enhance cross-lingual communication accuracy.

**4.Real-time Feedback:** Implementation of real-time feedback mechanisms to learn from user interactions and continuously improve autocorrection suggestions based on user preferences and corrections.

**5.Integration with Text Editors and Messaging Platforms**: Integration of the autocorrect system with popular text editors, messaging platforms, and communication tools can provide seamless autocorrection functionality across various digital platforms.

**6.Personalization and Customization**: Incorporation of user-specific preferences, such as personalized dictionaries, custom autocorrection rules, and user feedback mechanisms, can tailor the autocorrect system to individual user preferences and writing styles.

**7.Accessibility Features:** Development of accessibility features, such as voice input support, screen reader compatibility, and assistive technologies, can enhance the inclusivity and usability of the autocorrect system for users with diverse needs.

**8.Machine Learning Techniques:** Exploration of advanced machine learning techniques, including deep reinforcement learning, active learning, and transfer learning, can further optimize the autocorrect system's performance and adaptability to evolving linguistic patterns and user behaviors.

**SOURCE CODE:**

import os

import re

import numpy as np

import pandas as pd

from collections import Counter

import nltk

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('stopwords')

with open('/content/wiki.txt', 'r', encoding='ISO-8859-1') as f:

file = f.readlines()

file[0][:3000]

def process\_data(lines):

"""

Input:

A file\_name which is found in your current directory. You just have to read it in.

Output:

words: a list containing all the words in the corpus (text file you read) in lower case.

"""

words = []

for line in lines:

line = line.strip().lower()

word = re.findall(r'\w+', line)

words.extend(word)

return words

word\_l = process\_data(file)

vocab = set(word\_l)

print(f"The first ten words in the text are: \n{word\_l[0:10]}")

print(f"There are {len(vocab)} unique words in the vocabulary.")

def find\_wrong\_word(sent, vocab):

wrong\_words = []

sent = sent.strip().lower().split(" ")

for word in sent:

if word not in vocab:

wrong\_words.append(word)

return wrong\_words

find\_wrong\_word('selfy consiscious political movement', vocab)

def delete\_letter(word, verbose=False):

'''

Input:

word: the string/word for which you will generate all possible words

in the vocabulary which have 1 missing character

Output:

delete\_l: a list of all possible strings obtained by deleting 1 character from word

'''

delete\_l = []

split\_l = []

split\_l = [(word[:i], word[i:]) for i in range(len(word))]

delete\_l = [s[0]+s[1][1:] for s in split\_l]

if verbose: print(f"input word : {word} \nsplit\_l = {split\_l}, \ndelete\_l = {delete\_l}")

return delete\_l

delete\_word\_l = delete\_letter(word="cans", verbose=True)

def switch\_letter(word, verbose=False):

'''

Input:

word: input string

Output:

switches: a list of all possible strings with one adjacent charater switched

'''

switch\_l = []

split\_l = []

split\_l = [(word[:i], word[i:]) for i in range(len(word))]

for s in split\_l:

if len(s[1])>2:

temp = s[0] + s[1][1] + s[1][0] + s[1][2:]

elif len(s[1]) == 2:

temp = s[0] + s[1][1] + s[1][0]

elif len(s[1]) == 1:

continue

switch\_l.append(temp)

if verbose: print(f"Input word = {word} \nsplit\_l = {split\_l} \nswitch\_l = {switch\_l}")

return switch\_l

switch\_word\_l = switch\_letter(word="eta", verbose=True)

def replace\_letter(word, verbose=False):

'''

Input:

word: the input string/word

Output:

replaces: a list of all possible strings where we replaced one letter from the original word.

'''

letters = 'abcdefghijklmnopqrstuvwxyz'

replace\_l = []

split\_l = []

split\_l = [(word[:i], word[i:]) for i in range(len(word))]

for s in split\_l:

if len(s[1]) == 1:

for l in letters:

if l != s[1][0]:

temp = l

replace\_l.append(s[0]+temp)

elif len(s) > 1:

for l in letters:

if l != s[1][0]:

temp = l + s[1][1:]

replace\_l.append(s[0]+temp)

replace\_set = set(replace\_l)

# turn the set back into a list and sort it, for easier viewing

replace\_l = sorted(list(replace\_set))

if verbose: print(f"Input word = {word} \nsplit\_l = {split\_l} \nreplace\_l {replace\_l}")

return replace\_l

replace\_l = replace\_letter(word='can',  verbose=True)

print(f"Number of outputs of switch\_letter('at') is {len(switch\_letter('fate'))}")

def insert\_letter(word, verbose=False):

'''

Input:

word: the input string/word

Output:

inserts: a set of all possible strings with one new letter inserted at every offset

'''

letters = 'abcdefghijklmnopqrstuvwxyz'

insert\_l = []

split\_l = []

split\_l = [(word[:i], word[i:]) for i in range(len(word)+1)]

for s in split\_l:

for l in letters:

insert\_l.append(s[0]+l+s[1])

if verbose: print(f"Input word {word} \nsplit\_l = {split\_l} \ninsert\_l = {insert\_l}")

return insert\_l

insert\_l = insert\_letter('at', True)

print(f"Number of strings output by insert\_letter('at') is {len(insert\_l)}")

def edit\_one\_letter(word, allow\_switches = True):

"""

Input:

word: the string/word for which we will generate all possible wordsthat are one edit away.

Output:

edit\_one\_set: a set of words with one possible edit. Please return a set. and not a list.

"""

edit\_one\_set = set()

insert\_l = insert\_letter(word)

delete\_l = delete\_letter(word)

replace\_l = replace\_letter(word)

switch\_l = switch\_letter(word)

if allow\_switches:

ans = insert\_l + delete\_l + replace\_l + switch\_l

else:

ans = insert\_l + delete\_l + replace\_l

edit\_one\_set = set(ans)

return edit\_one\_set

tmp\_word = "at"

tmp\_edit\_one\_set = edit\_one\_letter(tmp\_word)

# turn this into a list to sort it, in order to view it

tmp\_edit\_one\_l = sorted(list(tmp\_edit\_one\_set))

print(f"input word : {tmp\_word} \nedit\_one\_l \n{tmp\_edit\_one\_l}\n")

#print(f"The type of the returned object should be a set {type(tmp\_edit\_one\_set)}")

print(f"Number of outputs from edit\_one\_letter('at') is {len(edit\_one\_letter('at'))}")

def edit\_two\_letters(word, allow\_switches = True):

'''

Input:

word: the input string/word

Output:

edit\_two\_set: a set of strings with all possible two edits

'''

edit\_two\_set = set()

one\_edit = edit\_one\_letter(word)

ans = []

for w in one\_edit:

ans.append(w)

ans.extend(edit\_one\_letter(w))

edit\_two\_set = set(ans)

return edit\_two\_set

tmp\_edit\_two\_set = edit\_two\_letters("a")

tmp\_edit\_two\_l = sorted(list(tmp\_edit\_two\_set))

print(f"Number of strings with edit distance of two: {len(tmp\_edit\_two\_l)}")

print(f"First 10 strings {tmp\_edit\_two\_l[:10]}")

print(f"Last 10 strings {tmp\_edit\_two\_l[-10:]}")

print(f"The data type of the returned object should be a set {type(tmp\_edit\_two\_set)}")

print(f"Number of strings that are 2 edit distances from 'at' is {len(edit\_two\_letters('at'))}")

def get\_count(word\_l):

'''

Input:

word\_l: a set of words representing the corpus.

Output:

word\_count\_dict: The wordcount dictionary where key is the word and value is its frequency.

'''

word\_count\_dict = {}

word\_count\_dict = Counter(word\_l)

return word\_count\_dict\

word\_count\_dict = get\_count(word\_l)

print(f"There are {len(word\_count\_dict)} key values pairs")

print(f"The count for the word 'thee' is {word\_count\_dict.get('thee',0)}")

def get\_probs(word\_count\_dict):

'''

Input:

word\_count\_dict: The wordcount dictionary where key is the word and value is its frequency.

Output:

probs: A dictionary where keys are the words and the values are the probability that a word will occur.

'''

probs = {}

total = 1

for word in word\_count\_dict.keys():

total = total + word\_count\_dict[word]

for word in word\_count\_dict.keys():

probs[word] = word\_count\_dict[word]/total

return probs

probs = get\_probs(word\_count\_dict)

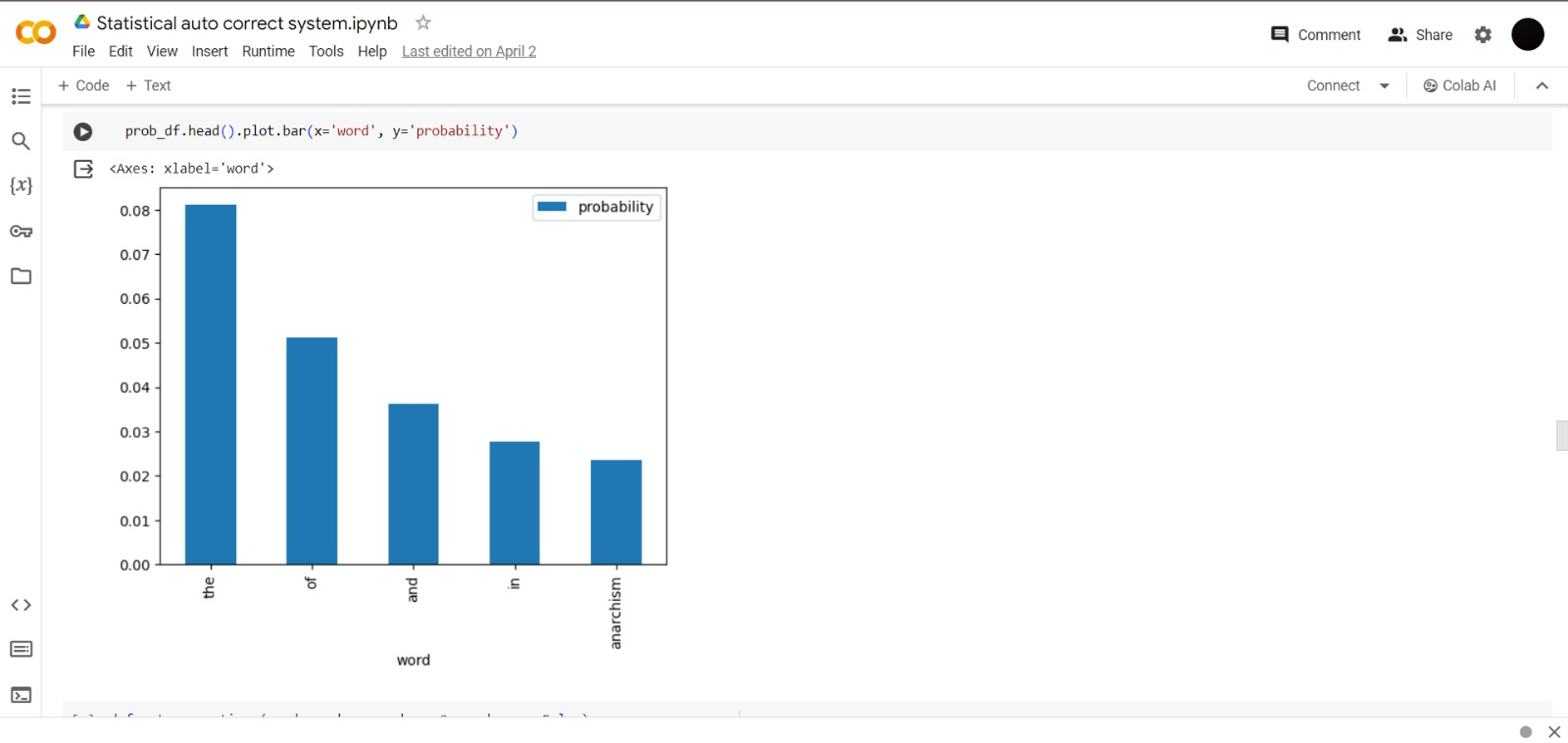
print(f"Length of probs is {len(probs)}")

prob\_df =

pd.DataFrame({'word':probs.keys(),'probability':probs.values()}).sort\_values(by='probability', ascending=False)

prob\_df.sample(5)

prob\_df.head().plot.bar(x='word', y='probability')



def get\_corrections(word, probs, vocab, n=2, verbose = False):

'''

Input:

word: a user entered string to check for suggestions

probs: a dictionary that maps each word to its probability in the corpus

vocab: a set containing all the vocabulary

n: number of possible word corrections you want returned in the dictionary

Output:

n\_best: a list of tuples with the most probable n corrected words and their probabilities.

'''

suggestions = []

n\_best = []

if word in probs.keys():

suggestions.append(word)

for w in edit\_one\_letter(word):

if len(suggestions) == n:

break

if w in probs.keys():

suggestions.append(w)

for w in edit\_two\_letters(word):

if len(suggestions) == n:

break

if w in probs.keys():

suggestions.append(w)

best\_words = {}

for s in suggestions:

best\_words[s] = probs[s]

best\_words = sorted(best\_words.items(), key=lambda x: x[1], reverse=True)

n\_best = best\_words

if verbose: print("entered word = ", word, "\nsuggestions = ", suggestions)

return n\_best

def get\_correct\_word(word, vocab, probs, n):

corrections = get\_corrections(word, probs, vocab, n, verbose=False)

# print(corrections)

if len(corrections) == 0:

return word

final\_word = corrections[0][0]

final\_prob = corrections[0][1]

for i, word\_prob in enumerate(corrections):

#print(f"word {i}: {word\_prob[0]}, probability {word\_prob[1]:.6f}")

if word\_prob[1] > final\_prob:

final\_word = word\_prob[0]

final\_prob = word\_prob[1]

return final\_word

get\_correct\_word('annd', vocab, probs, 100)

def autocorrect(sentence, vocab, probs):

print("Input sentence : ", sentence)

wrong\_words = find\_wrong\_word(sentence, vocab)

print("Wrong words : ", wrong\_words)

#print(wrong\_words)

correct\_words = []

for word in sentence.strip().lower().split(" "):

if word in wrong\_words:

correct\_word = get\_correct\_word(word, vocab, probs, 15)

#print(word, correct\_word)

word = correct\_word

correct\_words.append(word)

print("Output Sentence : ", " ".join(correct\_words).capitalize())

autocorrect("anarchsim is poliitcal philosophy and movement ", vocab, probs)

def count\_n\_grams(data, n, start\_token='<s>', end\_token = '<e>'):

# Initialize dictionary of n-grams and their counts

n\_grams = {}

for sentence in data:

# prepend start token n times, and append <e> one time

sentence = [start\_token]\*n + sentence + [end\_token]

sentence = tuple(sentence)

for i in range(len(sentence)-n):

n\_gram = sentence[i:i+n]

if n\_gram in n\_grams.keys():

n\_grams[n\_gram] += 1

else:

n\_grams[n\_gram] = 1

return n\_grams

def split\_to\_sentences(data):

#sentences = data.split("\n")

sentences = [s.strip() for s in data]

sentences = [s for s in sentences if len(s) > 0]

return sentences

def tokenize\_sentences(sentences):

tokenized\_sentences = []

for sentence in sentences:

sentence = sentence.lower()

tokenized = nltk.tokenize.word\_tokenize(sentence)

tokenized\_sentences.append(tokenized)

return tokenized\_sentences

def get\_tokenized\_data(data):

sentences = split\_to\_sentences(data)

tokenized\_sentences = tokenize\_sentences(sentences)

return tokenized\_sentences

tokenized\_data = get\_tokenized\_data(file)

bigram\_counts = count\_n\_grams(tokenized\_data, 2)

def get\_bigram\_prob(word, prev\_word, bigram\_counts, factor):

key = tuple([prev\_word, word])

#print(key)

ksum = 0

occ = 0

for k, v in bigram\_counts.items():

if k[0] == prev\_word:

ksum = ksum + v

occ = occ + 1

#print(ksum)

#print(occ)

count = 0

if key in bigram\_counts.keys():

count = bigram\_counts[key]

#print(type(occ))

smooth\_count = count + factor

smooth\_occ = ksum + occ\*factor

probability = smooth\_count / smooth\_occ

#print(probability)

return probability

get\_bigram\_prob('is', 'that', bigram\_counts, 1)

def get\_corrections\_bigram(word, prev\_word, probs, vocab, bigram\_counts, unigram\_weight=0.3, bigram\_weight=0.7, n=5, verbose = False):

'''

Input:

word: a user entered string to check for suggestions

probs: a dictionary that maps each word to its probability in the corpus

vocab: a set containing all the vocabulary

n: number of possible word corrections you want returned in the dictionary

Output:

n\_best: a list of tuples with the most probable n corrected words and their probabilities.

'''

suggestions = []

n\_best = []

if word in probs.keys():

suggestions.append(word)

for w in edit\_one\_letter(word):

if len(suggestions) == n:

break

if w in probs.keys():

suggestions.append(w)

for w in edit\_two\_letters(word):

if len(suggestions) == n:

break

if w in probs.keys():

suggestions.append(w)

best\_words = {}

for s in suggestions:

#best\_words[s] = probs[s]

unigram\_prob = probs[s]

#print(s)

try:

bigram\_prob = get\_bigram\_prob(s, prev\_word, bigram\_counts, 1)

except:

bigram\_prob = 0.0000000000000000001

final\_score = unigram\_weight\*unigram\_prob + bigram\_weight\*bigram\_prob

best\_words[s] = final\_score

best\_words = sorted(best\_words.items(), key=lambda x: x[1], reverse=True)

n\_best = best\_words

if verbose: print("entered word = ", word, "\nsuggestions = ", suggestions)

return n\_best

def get\_correct\_word\_bigram(word, prev\_word, probs, vocab, bigram\_counts, unigram\_weight, bigram\_weight, n):

corrections = get\_corrections\_bigram(word, prev\_word, probs, vocab,

bigram\_counts, unigram\_weight, bigram\_weight, n, verbose=False)

#print(corrections)

if len(corrections) == 0:

return word

final\_word = corrections[0][0]

final\_prob = corrections[0][1]

for i, word\_prob in enumerate(corrections):

#print(f"word {i}: {word\_prob[0]}, probability {word\_prob[1]:.6f}")

if word\_prob[1] > final\_prob:

final\_word = word\_prob[0]

final\_prob = word\_prob[1]

return final\_word

def autocorrect\_bigram(sentence, vocab, probs, bigram\_counts):

print("Input sentence : ", sentence)

wrong\_words = find\_wrong\_word(sentence, vocab)

print("Wrong words : ", wrong\_words)

#print(wrong\_words)

correct\_words = []

word\_list = sentence.strip().lower().split(" ")

for i, word in enumerate(word\_list):

#print(i, word)

#### Previous word

if i==0:

prev\_word = '<s>'

else:

prev\_word = word\_list[i-1]

if word in wrong\_words:

correct\_word = get\_correct\_word\_bigram(word, prev\_word, probs, vocab, bigram\_counts, 0.3, 0.7, 10)

#print(word, correct\_word)

word = correct\_word

correct\_words.append(word)

print("Output Sentence : ", " ".join(correct\_words).capitalize())

autocorrect\_bigram('anarchsim is poliitcal philosophy and movement ', vocab, probs, bigram\_counts)



**APPENDIX:**

Source code @github:

https://github.com/Abhinaya-cse/IBM-Gen-AI