

SUGGESTIVE TEXT USING NATURAL LANGUAGE PROCESSING

**Project report submitted
in partial fulfilment of the requirements for the Degree of
Bachelor of Engineering
in
Computer Science & Engineering**

By

Prashant Kumar (BE/6074/17)

Project Supervisor

Dr Sounak Paul

Dept. of CSE, BIT Mesra



BIRLA INSTITUTE OF TECHNOLOGY, MESRA

CERTIFICATE

This is to certify that Mr **Prashant Kumar** has developed the project titled “**SUGGESTIVE TEXT USING NATURAL LANGUAGE PROCESSING**” under my supervision and guidance.

To the best of my knowledge, the project work is original.

(Signature of Guide with Date)

Dr Sounak Paul

Department of Comp. Sc. & Engg.
Birla Institute of Technology, Mesra

ACKNOWLEDGEMENT

We would like to take the opportunity to express our sincere thanks to our guide **Dr SOUNAK PAUL**, Department of Computer Science and Engineering, Birla Institute of Technology, Mesra for his invaluable support and guidance throughout our project research work. Without his kind guidance & support, this was not possible.

We are grateful to him for his timely feedback which helped us track and schedule the process effectively. His time, ideas and encouragement that he gave helped us to complete our project efficiently. We would also like to thank him for providing an outstanding academic environment, also for providing the adequate facilities.

We are also thankful to **Prof. Dr K N Mishra**, Prof. incharge of, Department of Computer Science and Engineering, Birla Institute Of Technology, Mesra and all our B.E. teachers for providing advice and valuable guidance.

We would also like to extend our sincere thanks to all the faculty members and the non-teaching staff and friends for their cooperation.

Last but not the least, We are thankful to the family members whose constant support and encouragement in every aspect helped us to complete our project.

CONTENTS

<u>TOPIC NAME</u>	<u>PAGE NO.</u>
1. ABSTRACT	5
2. PROJECT OVERVIEW	6-7
3. REQUIREMENT SPECIFICATION	8
4. PROBLEM DESIGN Flow Chart & Algorithm	9-12
5. IMPLEMENTATION	13-21
6. RESULTS & DISCUSSIONS	22-25
7. CONCLUSION	26
8. REFERENCES	27

1. ABSTRACT

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using **natural language**. The ultimate objective of NLP is to read, decipher, understand, and make sense of human languages in a manner that is valuable. They are growing in popularity for their use in ML technologies like self-driving cars and speech recognition software.

Natural Language Processing is the driving force behind the following common applications:

- i) Language translation applications such as Google Translate.
- ii) Word Processors such as Microsoft Word and Grammarly that employ NLP to check grammatical accuracy of texts.
- iii) Interactive Voice Response (IVR) applications used in call centres to respond to certain users' requests.
- iii) Personal assistant applications such as OK Google, Siri, Cortana, and Alexa.

The fundamentals of text generation can be easily broken down into a simple supervised machine learning problem, wherein, there exists certain features (called x) with their corresponding labels (called y), and using these we can create our own prediction function which will then generate our predicted labels (called \hat{y} or y_{hat}). We then map these predicted labels to the actual labels to determine the cost and optimise it using an optimisation algorithm such as Gradient Descent, RMSprop or even the Adam optimiser.

And transitively we have reduced the task of text generation to a simple prediction problem. Hence, through this, we can have a corpus of text within which we may have several sentences. We extract each sentence (say it has n -words) and within each of these sentences, we mark the starting $n-1$ words as the features (called x) and the n th words as the label for it (called y).

Now, let's say we have the sentence, "Deep learning has automated our world," here, the phrase "Deep learning has automated our" can be the feature (called x) and the last word "world" can be the label (called y). So in the future, whenever the device encounters the text "Deep learning has automated our" it will know to predict "world" as the next word.

2. PROJECT OVERVIEW

2.1 Introduction

Suggestive text is popular across the board and in every industry, especially for mobile, app, and data science. Even journalism uses text generation to aid writing processes.

Text generation is popular across the board and in every industry, especially for mobile, app, and data science. Even journalism uses text generation to aid writing processes. We should have probably encountered text generation technology in your day-to-day life. Auto-correction or text prediction in various keyboards , Google search, and its Smart Compose on Gmail are just a few examples. These skills are valuable for any aspiring data scientist.

We should have probably encountered text generation technology in your day-to-day life. Auto-correction or text prediction in various keyboards , Google search, and Google's Smart Compose on Gmail are just a few examples. These skills are valuable for any aspiring data scientist.

We are going to build a text generator **using Natural Language Processing**. This will be a character based model that takes the previous character of the chain and predicts the next letter in the sequence.

By training our program with sample words, our text generator will learn common patterns in character order. The text generator will then apply these patterns to the input, of an incomplete word, and output the character with the highest probability to complete that word.

The text generator project relies on word prediction, a subdivision of natural language processing that predicts and generates next characters based on previously observed patterns in language.

Without NLP, we'd have to create a table of all words in the English language and match the passed string to an existing word. There are two problems with this approach.

- i) It would be very slow to search thousands of words
- ii) The predictor could only complete words that it had seen before.

NLP allows us to dramatically cut runtime and increase versatility because the generator can complete words it hasn't even encountered before. NLP can be expanded to predict words, phrases, or sentences if needed!

2.2 Problem Definition & Objectives

We are going to build a text predictor using Natural Language Processing. This will be a character based model that takes the previous character of the chain and predicts the next letter in the sequence.

By training our program with sample words, our text generator will learn common patterns in character order. The text generator will then apply these patterns to the input, of an incomplete word, and output the character with the highest probability to complete that word.

The text generator project relies on word prediction, a subdivision of natural language processing that predicts and generates next characters based on previously observed patterns in language.

NLP allows us to dramatically cut runtime and increase versatility because the generator can complete words it hasn't even encountered before. NLP can be expanded to predict words, phrases, or sentences if needed.

For this project, we will specifically be using Markov chains to complete our text. Markov processes are the basis for many NLP projects involving written language and simulating samples from complex distributions. Markov processes are so powerful that they can be used to generate superficially real-looking text with only a sample document.

3. REQUIREMENT SPECIFICATION

Technical Requirements

Various tools and techniques were used to implement the project model, the entire project has been developed using python and its library, while the front-end is designed using HTML.

Markov Chain

The Markov chain is a perfect model for our text generator because our model will predict the next character using only the previous character. Here The probability of each shift depends only on the previous state of the model, not the entire history of events.

Natural language Processing and Machine learning

Natural language processing used various tools like open NLP sentences and boundary detector and Google Language Detector.

Domain requirement

Our model is able to access the data freely without any complexity. Our model grants the file access of the device.

4. PROBLEM DESIGN

The design activities are of main importance in this phase, because in this activity, decisions ultimately affecting the success of the software implementation and its ease of maintenance are made. These decisions have the final bearing upon reliability and maintainability of the system. Design is the only way to accurately translate the customers requirements into finished software or asystem.Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software. Software design is conducted in two steps. Preliminary design is concerned with the transformation of require- ments into data.

Below is the proposed software model for our project:

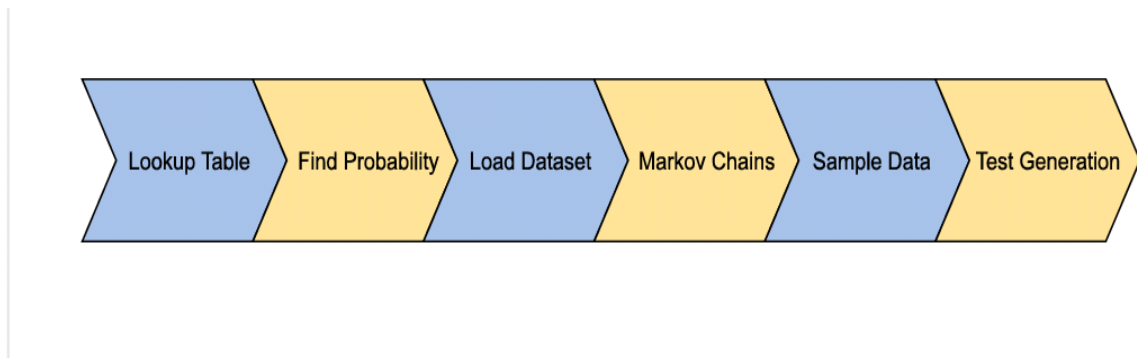
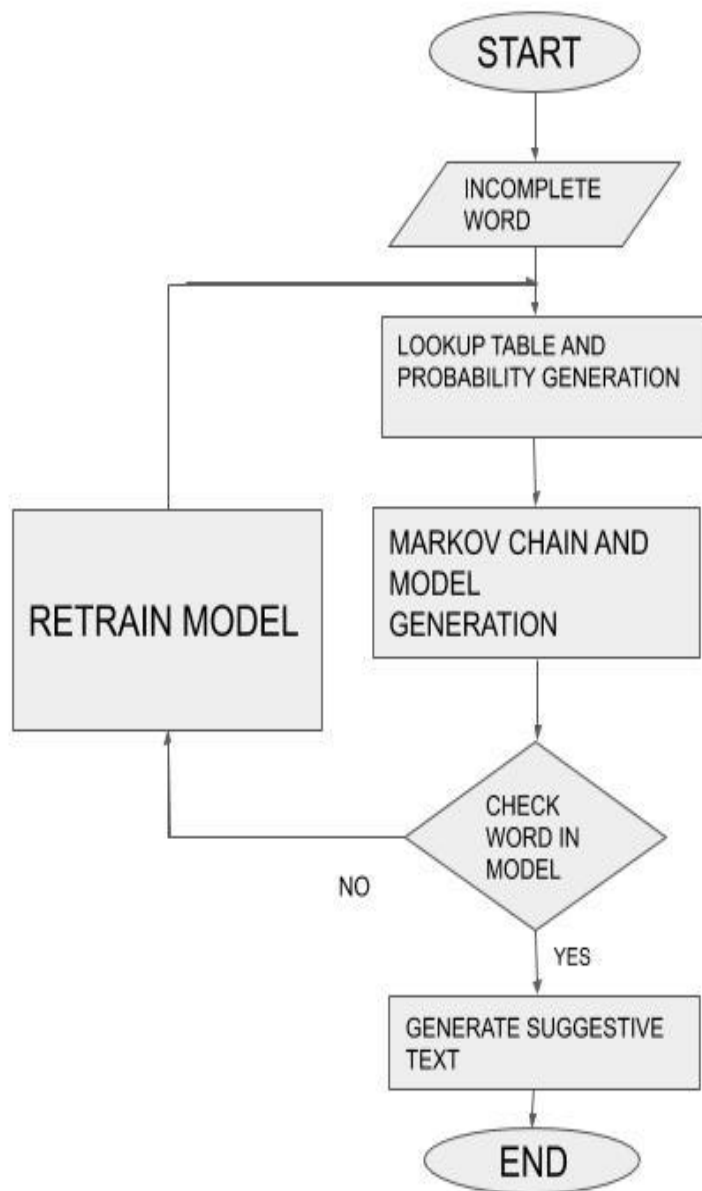


Figure 1.1: Model for Project

4.1 Flow Chart

The following flowchart shows the different processes in a sequential order:



4.2 Algorithm

Our suggestive text algorithm consists of five steps:

- **Step #1:** Creation of a dictionary for storing frequencies
- **Step #2:** Conversion of frequencies to probabilities.
- **Step #3:** Loading the dataset for training.
- **Step #4:** Building and Sampling our data.
- **Step #5:** Generation of suggestive text.

1. Generate the lookup table

First, we'll create a table that records the occurrences of each character state within our training corpus. We will save the last 'K' characters and the 'K+1' character from the training corpus and save them in a lookup table.

For example, imagine our training corpus contained, "the man was, they, then, the,the". Then the number of occurrences by word would be:

- "the" - 3
- "then" - 1
- "they" - 1
- "man" - 1

2. Convert frequencies to probabilities

Once we have this table and the occurrence, we'll generate the probability that an occurrence of Y will appear after an occurrence of a given X. Our equation for this will be:

Frequency of y with x/sum of total frequency

For example, if $x = \text{the}$ and $y = n$ our equation would look like this:

- ▶ Frequency when $y = n$ when $x = \text{the}$ is 2
- ▶ Total frequency in the table is 8
- ▶ Therefore $P = 2/8 = 0.125 = 12.5\%$

3. Loading the data sets for training

Next we'll load our real training corpus, we will use the long text (.txt) doc that you want. This data set will give our generator enough occurrences to make reasonably accurate predictions. As with all machine learning, larger training corpuses will result in more accurate predictions.

4. Building and Sampling our data

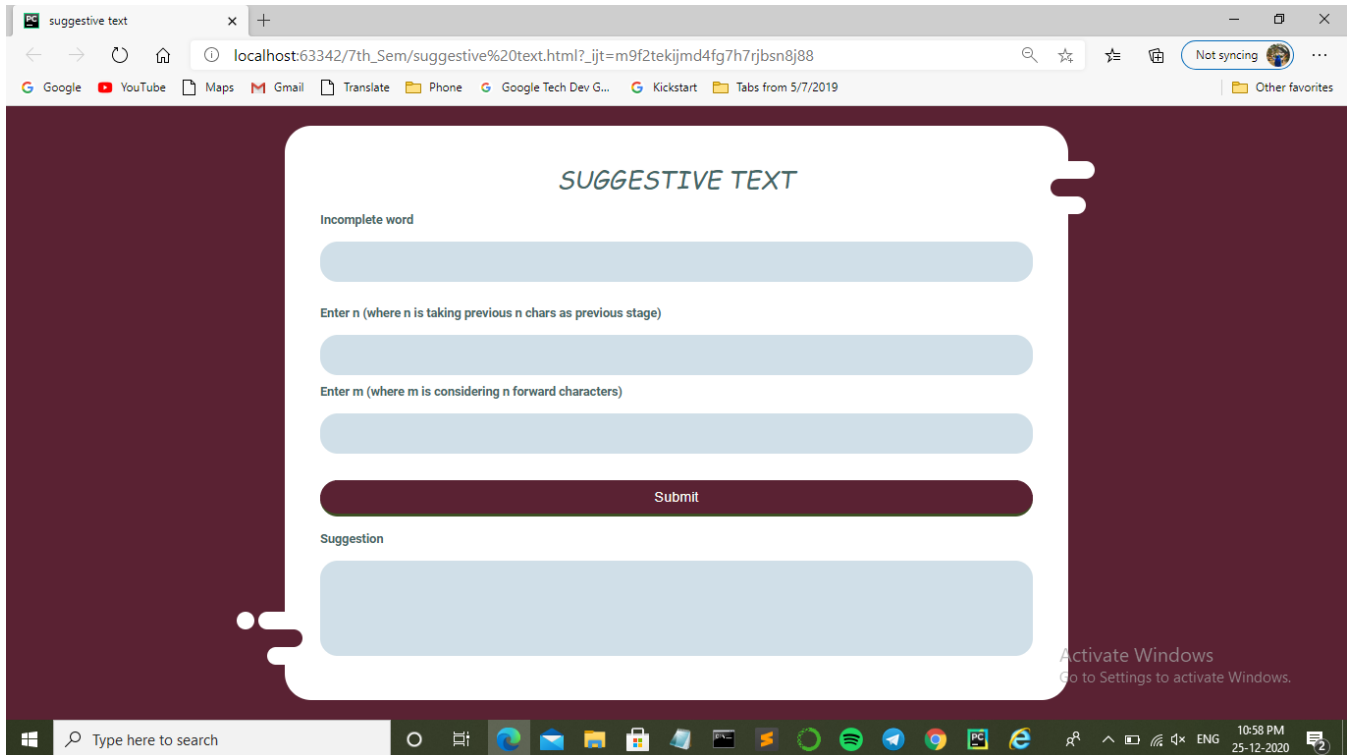
Now, we'll create a sampling function that takes the unfinished word, the Markov chains model, and the number of characters used to form the word's base. We'll use this function to sample passed context and return the next likely character with the probability it is the correct character.

5. Generation of suggestive text

Finally, we'll combine all the above functions to generate some text.

5. IMPLEMENTATION

FRONT END:



The front end of the project was designed with HTML and styling was done with CSS. The above image shown is the the front end implementation of our project.

The front end of our project was connected with the backend with the help of FLASK module of the of python.

FLASK:

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

Coding Environment:

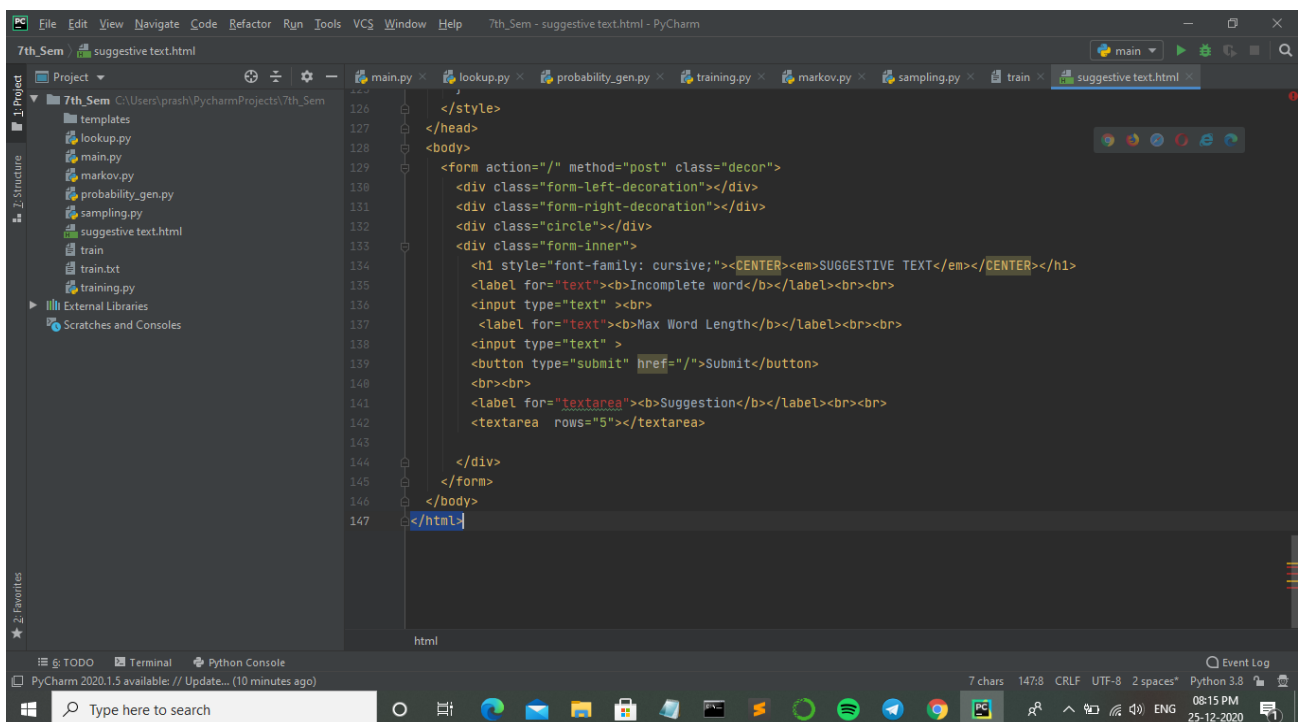


The project was coded and implemented in an integrated development environment (IDE) known as PyCharm.

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as data science with Anaconda.

PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also Professional Edition with extra features – released under a proprietary license.

Code:



```
126 </style>
127 </head>
128 <body>
129 <form action="/" method="post" class="decor">
130 <div class="form-left-decoration"></div>
131 <div class="form-right-decoration"></div>
132 <div class="circle"></div>
133 <div class="form-inner">
134 <h1 style="font-family: cursive;"><CENTER><em>SUGGESTIVE TEXT</em></CENTER></h1>
135 <label for="text"><b>Incomplete word</b></label><br><br>
136 <input type="text" ><br>
137 <label for="text"><b>Max Word Length</b></label><br><br>
138 <input type="text" >
139 <button type="submit" href="/">Submit</button>
140 <br><br>
141 <label for="textarea"><b>Suggestion</b></label><br><br>
142 <textarea rows="5"></textarea>
143
144 </div>
145 </form>
146 </body>
147 </html>
```

The above screenshot shows the body of the webpage, what we have implemented for our web page.

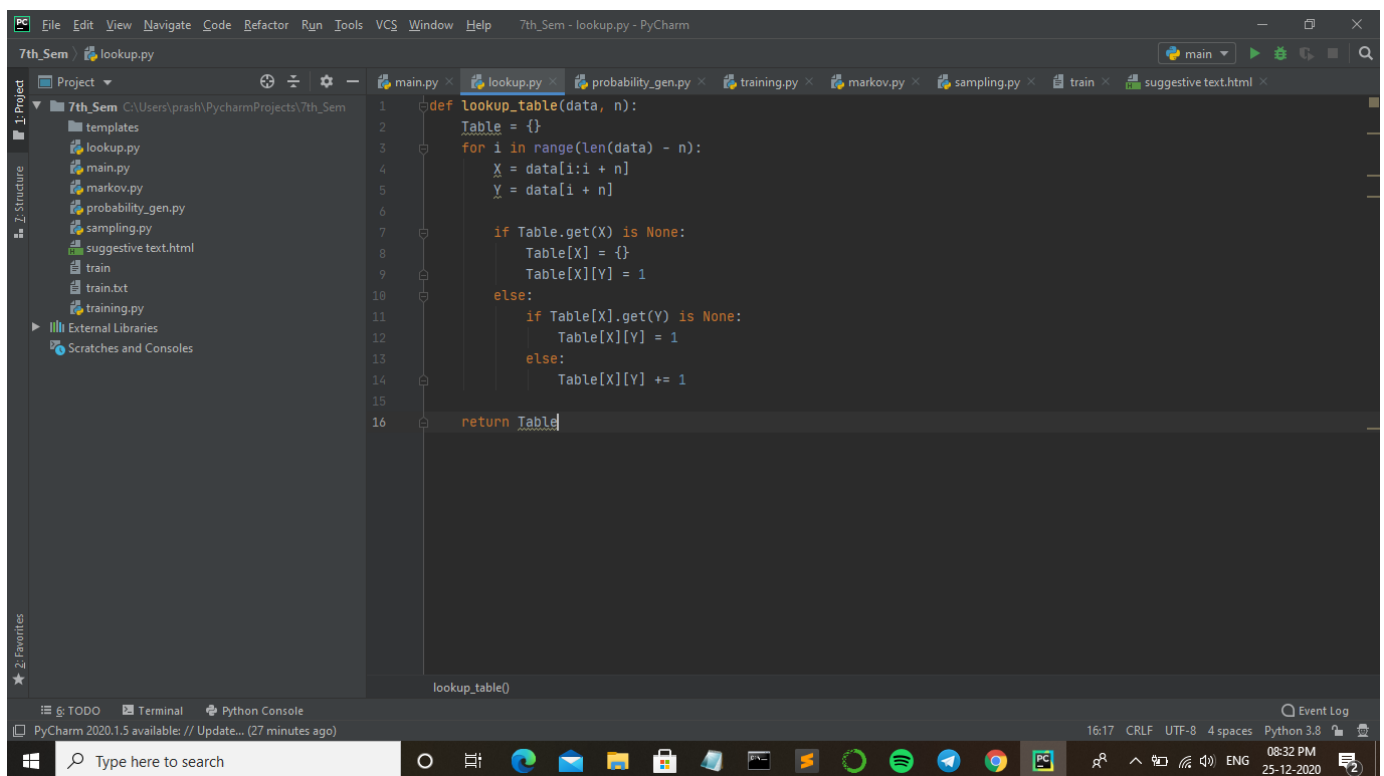
BACK END:

As mentioned above the back end of the project was carried out in Pycharm itself, where each step of the algorithm was separately coded as a single module for our ease and convenience.

Our project consisted of five modules based on the steps of algorithm, the screenshot and implementation is mentioned below.

- **Step #1:** Creation of a dictionary for storing frequencies

Module : lookup.py



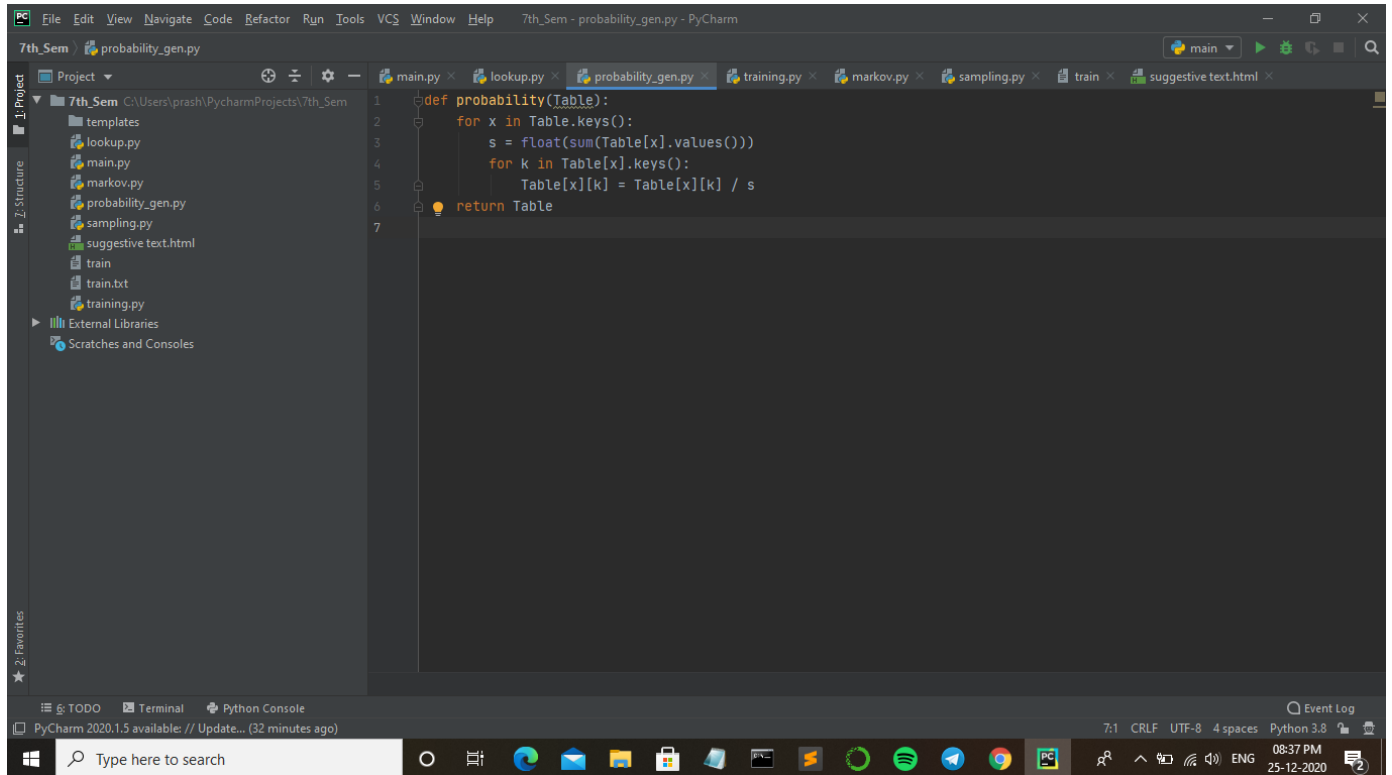
This was the module which contains the code for our lookup table generation.

Here, we created a dictionary that is going to store our X and its corresponding Y and frequency value.

We checked for the occurrence of X and Y, and, if we already have the X and Y pair in our lookup dictionary, then we just increment it by 1.

- **Step #2:** Conversion of frequencies to probabilities.

Module : probability_gen.py



This was the module which contains the code for converting frequencies to the probability which will be further helpful in generation of markov chains.

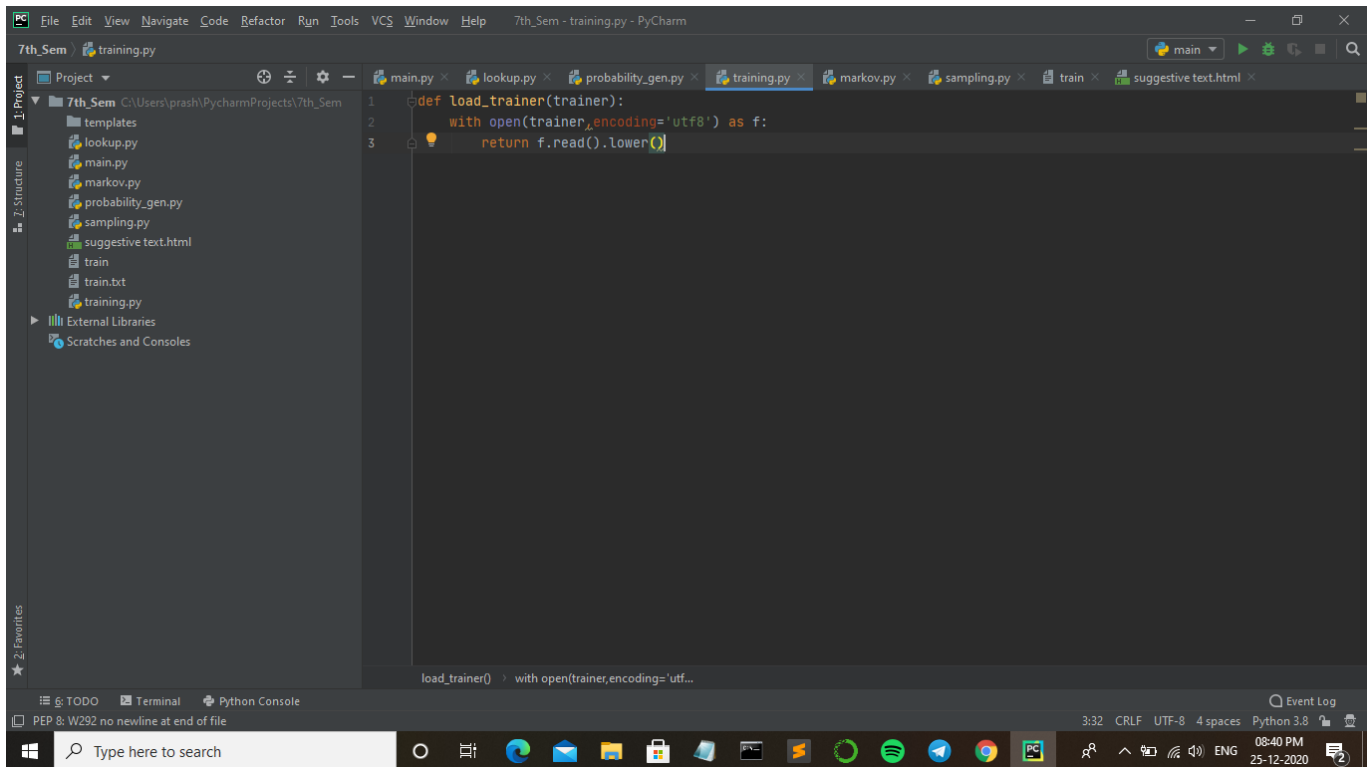
Here we summed up the frequency values for a particular key and then divided each frequency value of that key by that summed value to get our probabilities.

Frequency of Y with X

Sum of Total Frequencies

- **Step #3:** Loading the dataset for training.

Module : training.py



We'll load our real training set with the help of the load_trainer function, with the help of our training set we will feed the text file so that our model can learn from there which will further help the model to predict the next word.

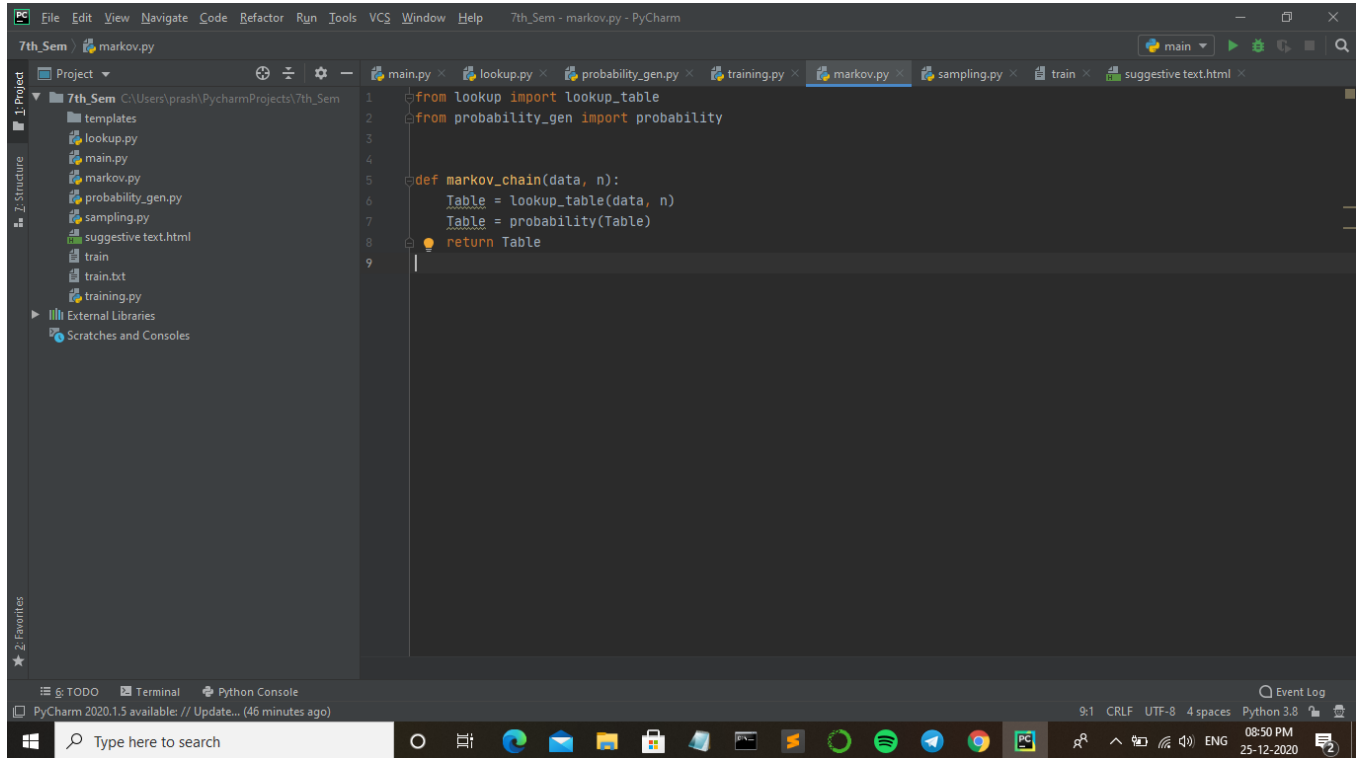
The trainer file will be attached in the reference section. The data set will give our generator enough occurrences to make reasonably accurate predictions. As with all machine learning, larger training corporuses will result in more accurate predictions.

Significance of training Sets:

While working with AI and ML model, giving the priority to training data will definitely help you to acquire the best quality of data sets to get best results. Cogito is one of the companies, providing the machine learning training data with labeling and annotation for AI model development into various fields, while ensuring the quality and accuracy at best level.

- **Step #4: Building and Sampling our data.**

Module : markov.py



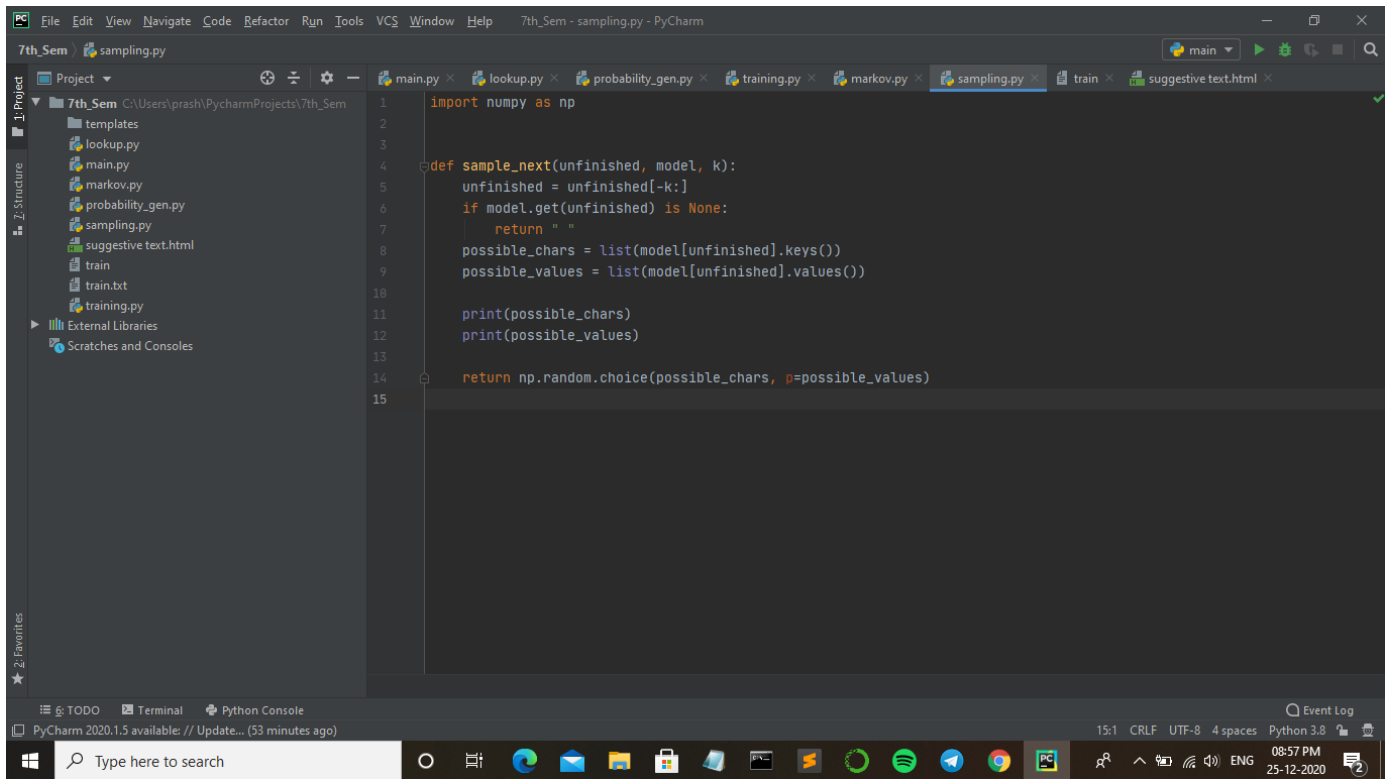
In this module markov.py, we imported lookup_table and probability from our previous created modules which was used to create our markov_chain function. This will play an important role in implementation the the project.

Here we created a method to generate the Markov model. This method accepts the text corpus and the value of n, which is the value telling the Markov model to consider n characters and predict the next character.

We generated our lookup table and we converted the frequencies into the probabilistic values by using the methods defined above.

- **Step #5: Building and Sampling our data.**

Module : sampling.py



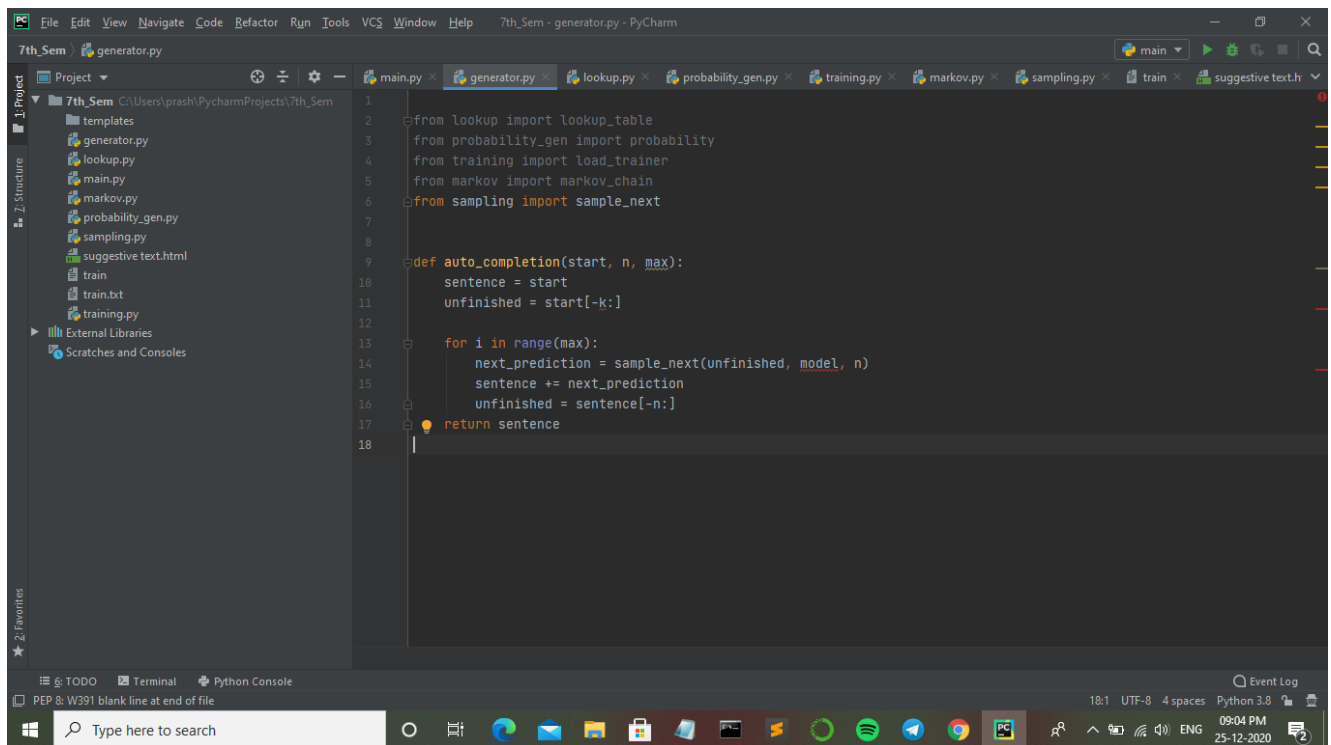
```
1 import numpy as np
2
3
4 def sample_next(unfinished, model, k):
5     unfinished = unfinished[-k:]
6     if model.get(unfinished) is None:
7         return " "
8     possible_chars = list(model[unfinished].keys())
9     possible_values = list(model[unfinished].values())
10
11     print(possible_chars)
12     print(possible_values)
13
14     return np.random.choice(possible_chars, p=possible_values)
15
```

Here we created sampling module for sampling our data, we took help our numpy module of python for generation of feasible characters and probability.

The function, `sample_next(unfinished,model,k)`, accepts three parameters: the context, the model, and the value of K. The unfinished is nothing but the text that will be used to generate some new text. We printed the possible characters and their probability values, which are also present in our model. We returned a sampled character according to the probabilistic values as we discussed above.

- **Step #6:** Generation of suggestive text.

Module : generator.py

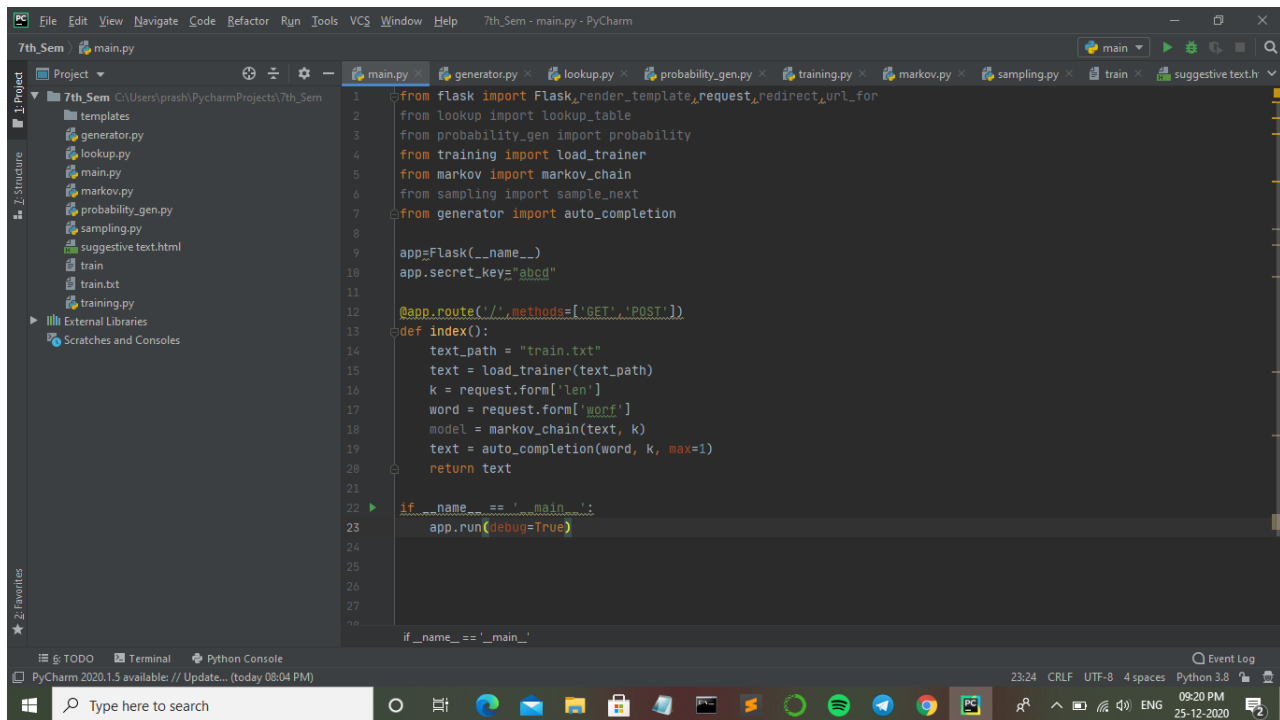


```
1
2 from lookup import lookup_table
3 from probability_gen import probability
4 from training import load_trainer
5 from markov import markov_chain
6 from sampling import sample_next
7
8
9 def auto_completion(start, n, max):
10     sentence = start
11     unfinished = start[-k:]
12
13     for i in range(max):
14         next_prediction = sample_next(unfinished, model, n)
15         sentence += next_prediction
16         unfinished = sentence[-n:]
17     return sentence
18
```

The function below takes in three parameters: the starting word from which you want to generate the text, the value of K, and the maximum length of characters up to which you need the text.

This is the most important module which contains the generator function which will generate the text for our incomplete word.

Module : main.py



```
1 from flask import Flask, render_template, request, redirect, url_for
2 from lookup import lookup_table
3 from probability_gen import probability
4 from training import load_trainer
5 from markov import markov_chain
6 from sampling import sample_next
7 from generator import auto_completion
8
9 app = Flask(__name__)
10 app.secret_key = "abcd"
11
12 @app.route('/', methods=['GET', 'POST'])
13 def index():
14     text_path = "train.txt"
15     text = load_trainer(text_path)
16     k = request.form['len']
17     word = request.form['word']
18     model = markov_chain(text, k)
19     text = auto_completion(word, k, max=1)
20     return text
21
22 if __name__ == '__main__':
23     app.run(debug=True)
```

In this module we imported all our previously created modules for generation of our predictive text. We connected the backend of our project with the frontend with the help of flask module.

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since we can build a web application quickly using only a single Python file.

6. RESULTS & DISCUSSIONS

What our project does ?

We build a text generator using Markov chains. This is a character based model that takes the previous character of the chain and generates the next letter in the sequence.

By training our program with sample words, our text generator will learn common patterns in character order. The text generator will then apply these patterns to the input, an incomplete word, and output the character with the highest probability to complete that word.

How our project does ?

The text generator project relies on text generation, a subdivision of natural language processing that predicts and generates next characters based on previously observed patterns in language, we specifically used Markov chains to complete our text. Markov processes are the basis for many NLP projects involving written language and simulating samples from complex distributions, they are so powerful that they can be used to generate superficially real-looking text with only a sample document.

What are Markov Chains ?

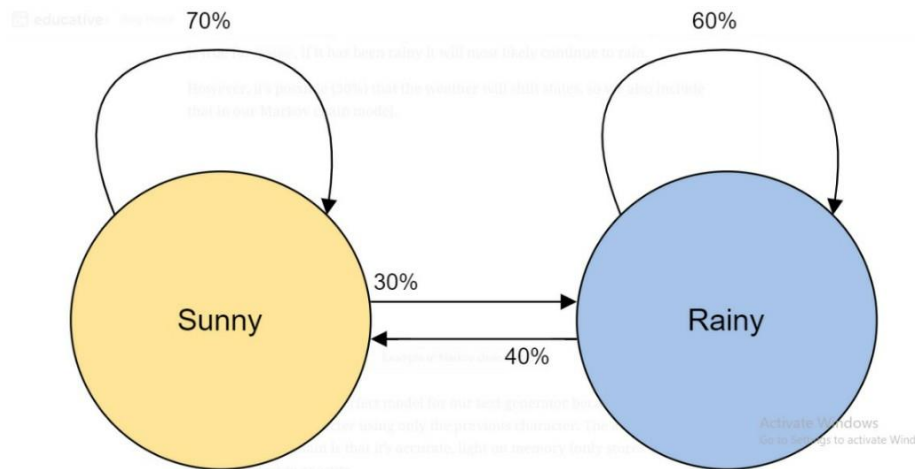
A Markov chain is a stochastic process that models a sequence of events in which the probability of each event **depends on the state of the previous event**. The model requires a finite set of states with fixed conditional probabilities of moving from one state to another.

For example, imagine you wanted to build a Markov chain model to predict weather conditions.

We have two states in this model, sunny or rainy. There is a higher probability (70%) that it'll be sunny tomorrow if we've been in the sunny state today. The same is true for rainy, if it has been rainy it will most likely continue to rain.

However, it's possible (30%) that the weather will shift states, so we also include that in our Markov chain model.

Example of Markov chain states:

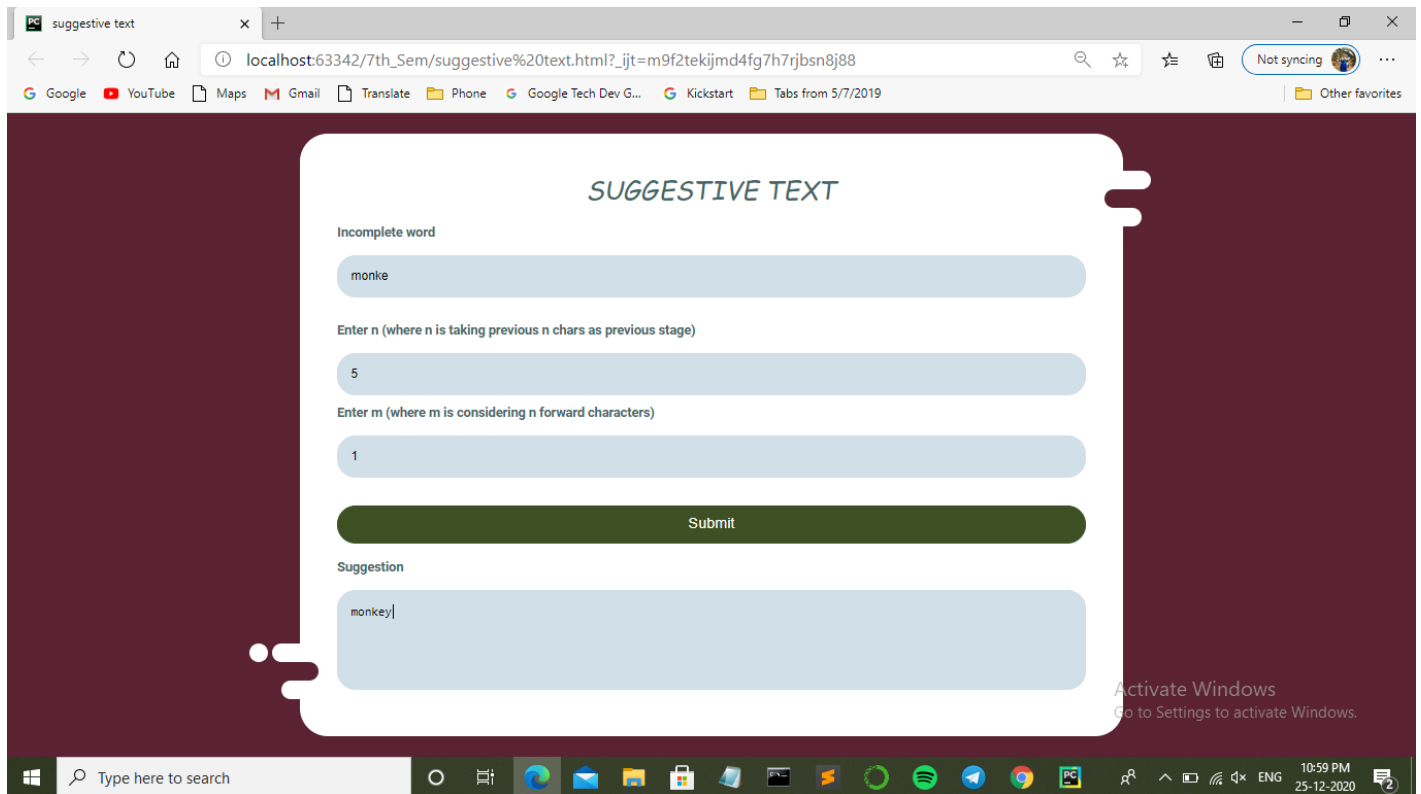


The probability of each shift depends only on the previous state of the model, not the entire history of events.

The Markov chain is a perfect model for our text generator because our model will predict the next character using only the previous character. The advantage of using a Markov chain is that it's accurate, light on memory (only stores 1 previous state), and fast to execute.

Outputs from our project:

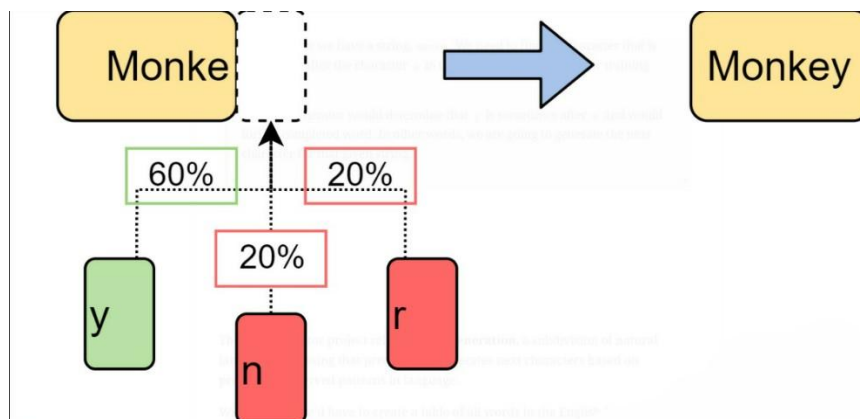
1.



Explanation:

Let's suppose we have a string, `monke`. We need to find the character that is best suited after the character `e` in the word `monke` based on our training corpus.

Our text generator would determine that `y` is sometimes after `e` and would form a completed word. In other words, we are going to generate the next character for that given string.



2.

suggestive text

localhost:63342/7th_Sem/suggestive%20text.html?_ijt=egfq0s1egfi41gjlq7umqekccd

Apps Google YouTube Maps Gmail Translate

SUGGESTIVE TEXT

Incomplete word

country

Enter n (where n is taking previous n chars as previous stage)

6

Enter m (where m is considering n forward characters)

1

Submit

Suggestion

country

Activate Windows
Go to Settings to activate Windows.

Type here to search

11:02 PM
25-12-2020

3.

suggestive text

localhost:63342/7th_Sem/suggestive%20text.html?_ijt=egfq0s1egfi41gjlq7umqekccd

Apps Google YouTube Maps Gmail Translate

SUGGESTIVE TEXT

Incomplete word

countr

Enter n (where n is taking previous n chars as previous stage)

6

Enter m (where m is considering n forward characters)

4

Submit

Suggestion

countrymen

Activate Windows
Go to Settings to activate Windows.

Type here to search

11:05 PM
25-12-2020

7. CONCLUSION

Suggestive text model is successfully implemented using Natural Language Processing. There are certain cases where the program might not return the expected result as each word is being considered only once. This will cause certain issues for particular word and will not receive the desired output. To improve the accuracy of the model we can consider trying out larger training sets or even feeding our model with dictionaries. We are able to develop a high-quality next word prediction for our dataset. The next word prediction model which we have developed is fairly accurate on the provided dataset. The overall quality of the prediction is good. However, certain pre-processing steps and certain changes in the model can be made to improve the prediction of the model.

Without NLP, we'd have to create a table of all words in the English language and match the passed string to an existing word. There are problems with this approach as it would be very slow to search thousands of words. NLP allows us to dramatically cut runtime and increase versatility because the generator can complete words it hasn't even encountered before. NLP can be expanded to predict words, phrases, or sentences if needed!

8. REFERENCES

- NLP at Work: The Essence of Excellence, 3rd Edition (People Skills for Professionals).
- Natural Language Processing Fundamentals by Sohom Ghosh, Dwight Gunning.
- Python Natural Language Processing by Jalaj Thanaki.
- Words That Change Minds: Mastering the Language of Influence.