MINI COMPILER

A COURSE PROJECT REPORT

By

KANISHKA GAUR (RA1911027010027)

DAKSH KAUSHIK (RA1911027010006)

SAGAR SHARDA

(RA1911027010031)

Under the guidance of

Dr. S. Sharanya

In partial fulfilment for the Course

of

18CSC304J – COMPILER DESIGN

in

Data Science and Business Systems

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chenpalpattu District

APRIL 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "Mini Compiler" is the bonafide work of DAKSH KAUSHIK (RA19110270100006), **KANISHKA GAUR (RA1911027010027), SAGAR SHARDA (RA1911027010031) who carried out the project work under my** supervision.

Dr. S. Sharanya,
Subject Staff
Associate Professor,
Data Science and Business Systems
SRM Institute of Science and Technology
Potheri, SRM Nagar, Kattankulathur,
Tamil Nadu 603203

Dr. S. Sharanya,
Course Cordinator
Associate Professor,
Data Science and Business Systems
SRM Institute of Science and Technology
Potheri, SRM Nagar, Kattankulathur,
Tamil Nadu 603203

TABLE OF CONTENTS

CHAPTER 1 – ABSTRACT

Auto-complete is a key feature for many web services. When you type in some phrases in Google, it presents a list of search suggestions. Sometimes these results use your input as prefix and sometimes not.

We have various phases to our project, starting from preprocessing the data, splitting data into smaller parts for purpose of training and testing, cleaning the data and finally building the model.

The first phase is preprocessing the data, where in we do the following things

- split the dataset by the \n character
- remove leading and trailing spaces
- drop empty sentences
- Tokenize sentences using nltk.word_tokenize

We shall discuss the phases in detail below.

Our project mainly looks to build an autocomplete system, that can provide the next word with relatively high accuracy and also give the probability of the word being the next one. We use n-gram models, which is a probabilistic model that is trained on a corpus of text, which is given to it as input.

CHAPTER 2 – MOTIVATION

Compiler Design is a part of our core subjects, and is also an important one. We learn about the compilers and how they process programs, but also equally important is the processing of grammar, which came under FLAT and is also a base reference to Compiler Design. So we also must know how the grammar is being processed.

In general, grammar processing comes under the topic of NLP. Natural language processing (NLP) refers to the branch of computer science concerned with giving computers the ability to understand text and spoken words in much the same way human beings can. We process our grammar, send it to our functions and then perceive the best possible output.

N-Gram models are Statistical(Probabilistic) Language models that aim to assign probabilities to a given sequence of words. Any N-gram is just a sequence of "n" words. For example, "Alex" is a unigram and "Hi There" is a bigram.

The task is to find out if we can compute P(w|h) given a word w and some history h . One could say that we can compute the probability of a given next word, using all the previous words in the sentence. For example using the last sentence, we could calculate:

P(word|One could say that we can compute the probability of a given next)

One such approach could be to use relative frequency counts to compute this probability, i.e. ,Out of the times we saw the history h , how many times was it followed by the word w

P(word | One could say that we can compute the probability of a given next)=

$$\frac{\text{C(One could say that we can compute the probability of a given next word)}}{\text{C(One could say that we can compute the probability of a given next)}}$$

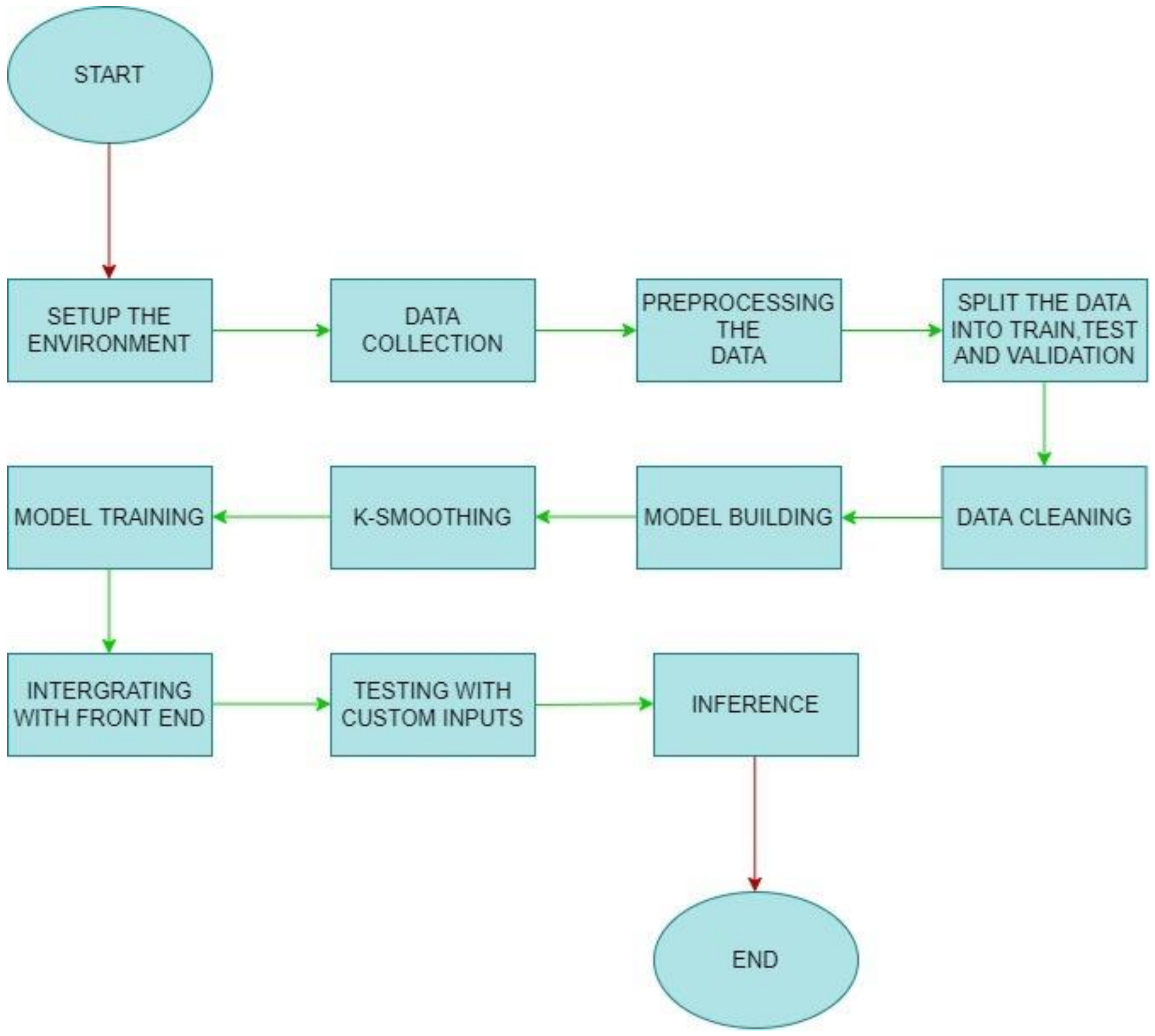
CHAPTER 3 – LIMITATIONS OF EXISTING METHODS

The current method takes a lot of time to compile the probabilities for one sentence, which is not ideal as we would like less waiting time for getting the next word.

It also can be resource intensive sometimes, which may be a deterrent for people having less than ideal devices, which discourages people from using it.

We need a large dataset to work on as the model might overfit if we use a smaller dataset. This means we have to have access to larger datasets, which is not possible every time. Overfitting is generally discouraged as we do not want our predictions to be biased.

CHAPTER 4 – PROPOSED METHOD WITH ARCHITECTURE



CHAPTER 5 – MODULES WITH DESCRIPTION

Setting up the environment:

Under this module we import and setup all the required libraries and dependencies which are needed for the project to run correctly without any hiccups. It also ensures that we do not keep importing or installing files and modules in between and break our flow of code.

Preprocess Pipeline:

The preprocess pipeline consists of a simple function, which makes our data usable to us. It contains the following sub functions:

- split the dataset by the \n character
- remove leading and trailing spaces
- drop empty sentences.
- Tokenize sentences using nltk.word_tokenize

These sub functions make it much better to work with the data as all the unwanted characters which can later cause problems.

Splitting the data into train, validation and test:

We split the data into train, validation and test subsets so that we can train our model and test out the accuracy it gives and ensure that on production of new random data, it does not overfit or underfit the model.

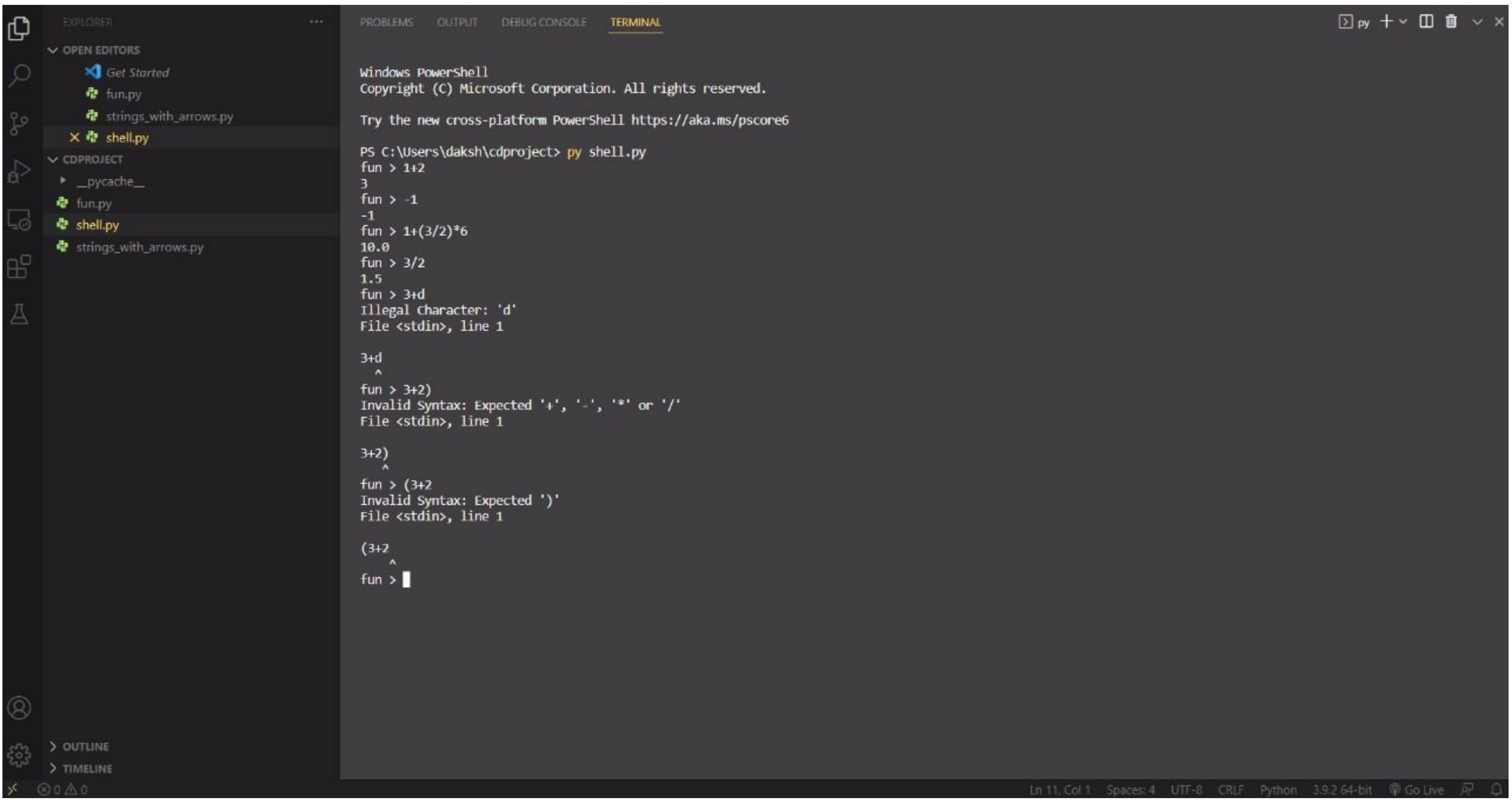
Cleaning Data:

In this part, we first create a frequency dictionary, which ensures that we use only certain words that repeat upto a certain number and beyond. This helps us in narrowing down the next word and to not have too many suggestions which will reduce the accuracy of the model. We also have to take care of out-of-vocabulary and unknown words, which we do. We then finally return new data after all these processes.

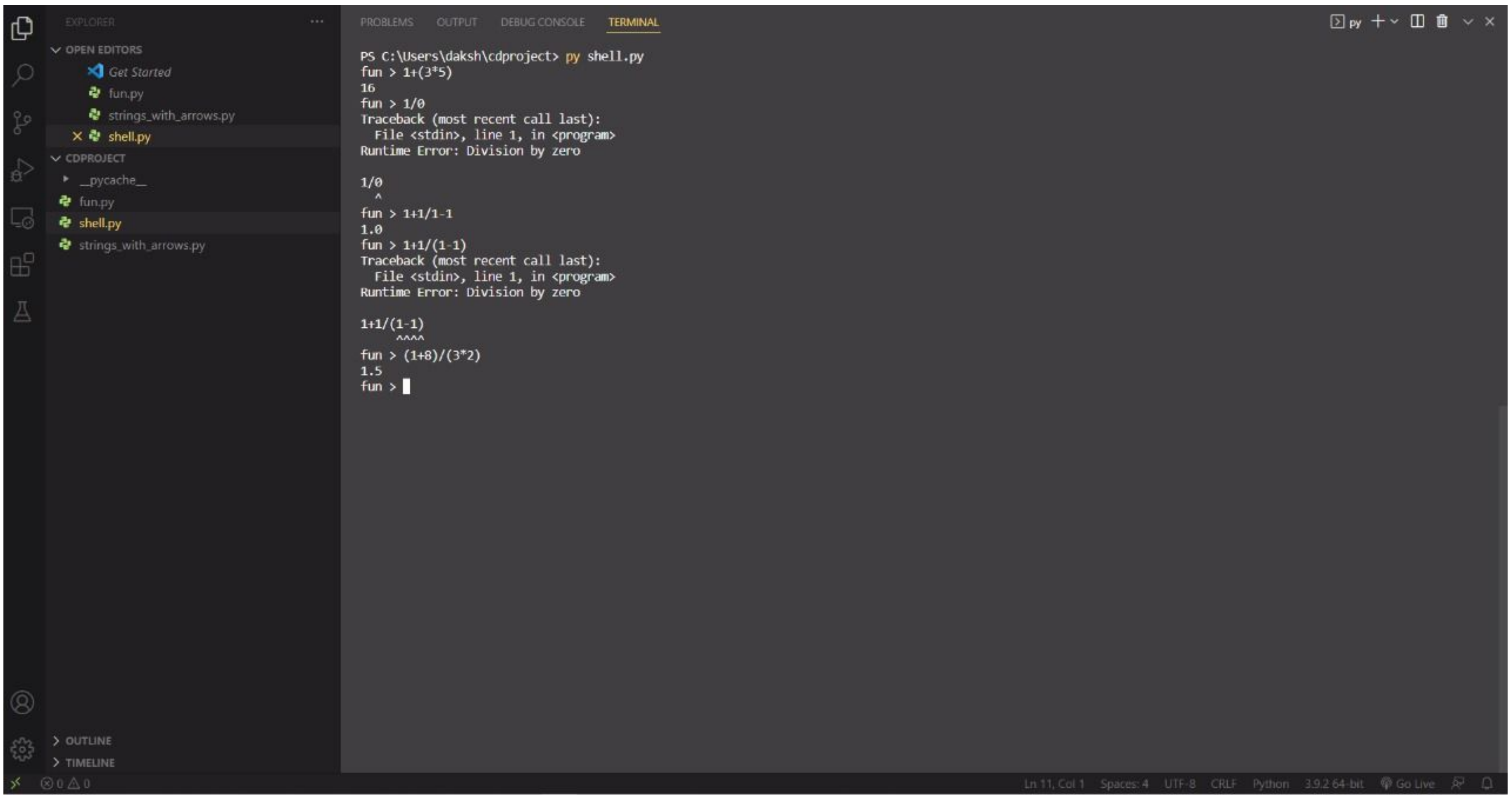
Building the model:

We make a helper function which maps the n-grams to their respective frequency within our dataset. We also need to include k-smoothing, as, if we encounter a n-gram which is not present, our probability will turn to 0, which will render our function useless. We then get the probability computed and finally move to build the autocomplete function

CHAPTER 6 – SCREENSHOTS



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\daksh\cdproject> py shell.py
fun > 1+2
3
fun > -1
-1
fun > 1+(3/2)*6
10.0
fun > 3/2
1.5
fun > 3+d
Illegal Character: 'd'
File <stdin>, line 1

3+d
  ^
fun > 3+2)
Invalid Syntax: Expected '+', '-', '*' or '/'
File <stdin>, line 1

3+2)
   ^
fun > (3+2
Invalid Syntax: Expected ')'
File <stdin>, line 1

(3+2
    ^
fun > █
```



```
PS C:\Users\daksh\cdproject> py shell.py
fun > 1+(3*5)
16
fun > 1/0
Traceback (most recent call last):
  File <stdin>, line 1, in <program>
Runtime Error: Division by zero

1/0
  ^
fun > 1+1/1-1
1.0
fun > 1+1/(1-1)
Traceback (most recent call last):
  File <stdin>, line 1, in <program>
Runtime Error: Division by zero

1+1/(1-1)
    ^^^^
fun > (1+8)/(3*2)
1.5
fun >
```

CHAPTER 7 – CONCLUSION

The autocomplete feature which we set out to make works and displays the probability of the next occurring word in a straightforward manner.

We have all our functions working in sync one after the other to check and produce the required outputs for us in the form of the next word and its probability.

Spelling correction was the one universally acknowledged use. For subject-based searching, confidence in the topic searched and the stage of research emerged as indicators of the likelihood of autocomplete suggestions being taken. The use and effectiveness of providing subject suggestions requires further study.

The further scope of this project can be expanding it into various other forms, than only using our website. We have opportunities to bring it into a smaller form factor and also make it open source so that we have other suggestions also pouring in and we are not limited to only one set of ideas.

CHAPTER 8 – REFERENCES

https://necromuralist.github.io/Neurotic-Networking/posts/nlp/auto-complete-building-the-auto-complete-system/#org0c4ae44
https://www.analyticsvidhya.com/blog/2021/09/what-are-n-grams-and-how-to-implement-them-in-python/
https://www.kaggle.com/code/sauravmaheshkar/auto-completion-using-n-gram-models/notebook#%E2%9C%82%EF%B8%8F-Splitting-into-Train,-Valid-and-Test
https://devpost.com/software/auto-completion-using-n-gram-models-jdweg3