A Project Report

on

AUTOMATED WORD PREDICTION IN TELUGU LANGUAGE USING STATISTICAL APPROACH

submitted in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

17WH1A0514	Ms. G.KIRTHANA
17WH1A0548	Ms. K.VAISHNAVI
18WH5A0508	Ms. S.DEVI SREE

under the esteemed guidance of

Dr. L. LAKSHMI
Professor

Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

May, 2021

DECLARATION

We hereby declare that the work presented in this project entitled **"AUTOMATED WORD PREDICTION IN TELUGU LANGUAGE USING STATISTICAL APPROACH"** submitted towards completion of Project Work in IV year of B.Tech. CSE at 'BVRIT HYDERABAD College of Engineering For Women', Hyderabad is an authentic record of our original work carried out under the guidance of Dr.L.Lakshmi, Professor, Department of CSE.

Sign. with date:
Ms. G.KIRTHANA
(17WH1A0514)

Sign. with date:
Ms. K.VAISHNAVI
(17WH1A0548)

Sign. with date:
Ms. S.DEVI SREE
(18WH5A0508)

BVRIT HYDERABAD

College of Engineering for Women

(NBA Accredited – EEE, ECE, CSE and IT)

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering

Certificate

This is to certify that the Project Work report on "**AUTOMATED WORD PREDICTION IN TELUGU LANGUAGE USING STATISTICAL APPROACH**" is a bonafide work carried out by Ms. G. KIRTHANA (17WH1A0514); Ms. K.VAISHNAVI (17WH1A0548); Ms.S.DEVI SREE (18WH5A0508) in the partial fulfillment for the award of B.Tech. Degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department	**Guide**
Dr. K. Srinivasa Reddy,	**Dr. L. Lakshmi**
Professor and HoD,	**Professor**
Department of CSE	

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal**, **BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. K. Srinivasa Reddy, Professor**, Department of CSE, **BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, Dr**. L. Lakshmi, Professor**, Department of CSE**, BVRIT HYDERABAD College of Engineering for Women** for his constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. G.KIRTHANA

(17WH1A0514)

Ms. K.VAISHNAVI

(17WH1A0548)

Ms. S.DEVI SREE

(18WH5A0508)

Contents

Abstract

Word prediction is one of the important phenomena in typing that benefit various types of users who use a keyboard to write. This can be an onscreen keyboard on a smartphone or digital tablet. It can be a physical keyboard connected to a device or computer. They can have a profound impact on the typing of disable people. It is based on word prediction on Telugu sentence by using statistical approach, i.e. N-gram language model such as unigram, bigram, trigram, Language Models helps in predicting the next few words in sequence from the past history, for predicting the next word using probabilities which saves time and keystrokes of typing and also reduces misspelling. We use large data corpus of Telugu language of different word types to predict correct word with the accuracy as much as possible.

LIST OF FIGURES

1. INTRODUCTION

Automated word prediction helps to predict the next Telugu word using the sequence of previous n-words of Telugu language. It uses the probabilities and statistical language modeling. Machine learning is an application that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Its techniques are also used for prediction of accurate probabilities of the next possible words.

1.1 Objectives

The main goal of word prediction is for the users who tend to use keyboard or other similar devices. The Automated Word Prediction using Statistical Approach for predicting the next few words in sequence from the past history, for predicting the next word using probabilities which saves time and keystrokes of typing and also reduces misspelling. The assumption states that having the knowledge of the full history, one can make future predictions only in its present state. Using the dataset of Telugu language of different word types to predict the correct word with as much accuracy as possible.

1.2 Methodology

To predict the next possible word an enormous amount of dataset is required which is collected from a number of Wikipedia pages. The methodology used is discussed in detail in this section.

1.2.1 Dataset

Datasets hold a very prominent role in prediction .Proper and large dataset is required for all classification research during the training and the testing phase. The dataset for the experiment is downloaded from Telugu Wikipedia Pages. It is a collection of different domains like *districts, Poems, Poets, History, etc*. The Total of 5 lakhs sentences of wiki documents are extracted to form a Dataset. The extracted wiki documents data, 5,00,000 clean sentences (which mostly has Telugu words) are taken for model development and testing. The Dataset is Split into Train Data of 4,50,000 sentences and Test data of 50,000 sentences.

Word prediction can speed up the writing process and help with word recall and correct spelling. For people with dyslexia, especially students who are still working to improve their language skills, it can act as a bridge between remediation and accommodation.

Given a previous n-word sequence of Telugu language, predicts the next Telugu word. The statistical language modeling is done based on the probabilities and statistical approaches. It accepts previous n Telugu words as input and predicts best fit next Telugu word.

Input : Previous ‘n’ Telugu word sequence

Output : Next Telugu word

What you will learn:

- How to use Python
- Learning about probabilistic approach for text processing

The Dataset used for the word prediction has been extracted from Telugu Wikipedia page a total 5 lakhs wiki documents are taken. The dataset holds a collection of different streams of information like about various poets, about the legendries history, beautiful poems and much more. From extracted wiki documents data, 5,00,000 clean sentences (which mostly have Telugu words) are taken for model development and testing. The Total Number of words are 3,43,917 and unique words are 1,52,934. The Dataset is split into train and test data for getting accurate results. The Train Data consists of 4,50,000 sentences and the Test data consists of 50,000 sentences. The data extracted is inconsistent and contains errors or outliers, and often lacks specific attribute values/trends. This is where data preprocessing enters the scenario – it helps to clean, format, and organize the raw data, thereby making the dataset suitable for machine learning so that it yields accurate results.

Data preprocessing:

Data preprocessing is the first and most crucial step before implementation. It is the process of transforming raw data into appropriate understandable format. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Splitting dataset into training and test set

To achieve that in this project tokenization and cleaning is done.

Data Cleaning

Data cleaning or data cleansing is the process of detecting and correcting corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.
Cleaning is done because the raw data contains many unnecessary strings like
<u>Html Tags :</u>
Html tags that are present in an HTML code that defines every structure on an HTML page, including the placement of text and images and hypertext links. HTML tags begin with the less-than (<) character and end with greater-than (>). These symbols are also called "angle brackets".

English Words:

E*nglish words* as the data set mostly contain the words of only Telugu language but the raw data has some of the English alphabets embedded in these which cause havoc during the implementation so it is very necessary to remove them.

Punctuations:

Punctuations are generally used of spacing, conventional signs, and certain typographical devices as aids to the understanding and correct reading of written text, whether read silently or aloud. But these act as a barrier while implementing as the cleaned dataset should contain only Telugu words so in the process of cleaning these must be removed.

Tokenization:

Tokenization is one of the most common tasks when it comes to working with text data. Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units is called tokens. The tokens could be words, numbers or punctuation marks. In tokenization, smaller units are created by locating word boundaries. These are the ending points of a word and the beginning of the next word. Before processing a natural language, we need to identify the *words* that constitute a string of characters. Tokenization is the most basic step to proceed with NLP (text data). Tokenization can be done in various ways in python like

- Tokenization using Python split() Function
- Tokenization using Regular Expressions
- Tokenization using NLTK
- Tokenization using Spacy
- Tokenization using Keras
- Tokenization using Gensim

Dataset Sample:

తిక్కన మనుమసిద్ధి రాజ్యము అంతరించిన తరువాత కూడా చిరకాలము జీవించి, సర్వజనులచే గౌరవిమ్పబడేవాడైనా, మరణకాలమునకు విశేషవృత్తవంతుడిగా కనబడడు ,అందుచే అతని కుమారుడు కొమ్మ న పాటూరి కరినణమును సంపాదించవలసి వాడయ్యేన

తిక్కన (1205 - 1288) మహాభారతములో నన్న య్య రచించిన పర్వాలు కాకుండా మిగిలిన 15 పర్వాలను రచించాడు. ఆది కవి నన్న య ఆది పర్వము, సభాపర్వము, అరణ్యపర్వములో కొంతభాగము రచించి గతించిరి. అరణ్యపర్వములో మిగిలిన భాగమును ఎఱ్ఱన రచించాడు. తిక్కన అరణ్యపర్వమును వదలి, మిగిలిన పర్వములు రచించిరి. ముందుగా యజ్ఞము చేసి, సోమయాజియై, పిదప ఈ బృహత్తర కార్యక్రమాన్ని చేపట్టాడు. ఈయనకు "కవి బ్రహ్మ", "ఉభయ కవిమిత్రుడు" అనే బిరుదులు ఉన్నా యి.

క్రీస్తు శకం 1253 సంవత్సరంలో తిక్కన కోవూరు మండల పరిధిలోని పాటూరు గ్రామ సిద్దేశ్వరాలయంలో యజ్ఞం చేసినట్లు చరిత్ర చెబుతోంది. ఆశయసిద్ధి కోసం ఈశ్వరాలయంలో యజ్ఞం చేసినందువల్ల ఆ ఆలయాన్ని సిద్ధేశ్వరాలయంగా పిలిచారు. యజ్ఞం పూర్తి చేసిన తరువాత తిక్కన సోమయాజిగా మారి మహాభారత రచనకు ఉపక్రమించారు. అప్పటి యజ్ఞానికి సంబంధించిన అనేక అవశేషాలు నేడు శిధిలావస్థకు చేరుకొన్నా యి.

తిక్కన తిరుగాడిన జాడలేవీ?'వింటే భారతం వినాలి ... తింటే గారెలు తినాలి' అనే నానుడికి జీవం పోసింది తిక్కన. మహాభారత కథనాలకు అంతటి ఖ్యాతిని ఆర్జించిన కవిబ్రహ్మ తిక్కన మెచ్చిన ప్రదేశం, ఆయన పూజించిన ఆలయం నేడు దయనీయ స్థితికి చేరుకొన్నా యి.

మానవుడు పంజరంలోని చిలుకలాంటి వాడు' అనే ఉపమానం, నానుడి తిక్కన చాలా పర్యాయాలు ఉపయోగించారు. నిర్వచనోత్తర రామా యణంలో మొదటి మనుమసిద్ధిని వర్ణిస్తూ "కీర్తి జాలము త్రిలోకీ శారికకు అభిరామరాజిత పంజరంబుగజేసి" అని చెప్పారు. అలాంటి తిక్కనే పూజించి, యజ్ఞం చేసిన సిద్ధేశ్వరాలయం, రాతివిగ్రహాలు నేడు నిర్లక్ష్యమనే పంజరంలో చిక్కుకొని శిధిలావస్థలో కొట్టుమిట్టాడుతున్నా యి. ఆయన పూజలు చేసిన నందీశ్వరుడ్ని అపహరించారు. మహాభారతాన్ని రసరమ్యంగా వర్ణించేందుకు తిక్కనకు సహకరించింది కోవూరు ప్రాంతమే.

తిక్కన పూర్వీకులు 'కొట్టురువు' ఇంటి పేరుతో పాటూరు గ్రామాధిపతులుగా పనిచేసినట్లు చరిత్ర చెబుతోంది. మనుమసిద్ధి కాలంలో తిక్కన ఇంటిపేరు 'పాటూరుగా' మారినట్లు చరిత్రకారులు చెబుతున్నా రు. యజ్ఞయాగాదులు అంటే తిక్కనకు చాలా ఇష్టం. పదకొండు పర్యాయాలు ఆయన పాటూరులోని సిద్ధేశ్వరాలయంలో యజ్ఞం చేసినట్లుగా కేతన తన దశకుమార చరిత్రలో పేర్కొన్నా రు. వేప, రావి చెట్లు మొలచి ఆలయం ధ్వంసమవుతోంది. ఆలయ ప్రాంగణాన ఉన్న బావిలో తిక్కన నిత్యం స్నా నమాచరించి, సంధ్యావందనం చేసినట్లుగా తెలుస్తోంది. ఆ బావి వర అంతర్భా గంలో చెక్కిన చంద్రుడు, వినాయకుని శిల్పాలు సుందరంగా ఉండేవట కానీ, బావి పూర్తిగా ముళ్లపొదలతో నిండిపోవడం చేత ఆ శిల్పాల్ని ఇప్పుడు చూడలేము. మహాభారత రచనకు తిక్కన ఉపయోగించినట్లుగా చెప్పే 'ఘంటం' పాటూరుకు చెందిన తిక్కన వారసుల వద్ద ఉందని చెబుతారు. 'ఘంటం' ఉంచే ఒరకు ఒక వైపు సరస్వతీ దేవి, వినాయకుని ప్రతిమల్ని చెక్కారని, తాము చాలా సంవత్సరాల క్రిందట దానిని చూశామని పాటూరు గ్రామ వయోవృద్ధులు చెప్పారు.

నెల్లూరుకు చెందిన సాహిత్య సంస్థ 'వర్ధమానసమాజం' కొన్నే ళ్ల కిందట నిర్వహించిన 'తిక్కనతిరునాళ్ళ'లో దానిని ప్రదర్శించారు. ఆ తరువాత ఒర చిరునామా లేకుండా పోయింది.

తిక్కన రూపాన్ని దశకుమార చరిత్రలో కేతన వర్ణించారు. ఆయన వర్ణన ఆధారంగా 1924 సంవత్సరంలో గుర్రం మల్లయ్య అనే చిత్రకారుడు ఆంధ్రా యూనివర్సిటీ నిర్వహించిన చిత్రలేఖన పోటీల్లో తిక్కన రూపాన్ని చిత్రీకరించారు. ఆ చిత్రపటమే నేడు నెల్లూరు పురమందిరంలోని వర్ధమాన సమాజంలో పూజలందుకుంటోంది. 1986 సంవత్సరంలో అప్పటి రాష్ట్ర ప్రభుత్వం ఆలయ పునర్ని ర్మాణానికి రెండు లక్షల రూపాయల్ని మంజూరు చేసింది. అయితే - సిద్ధేశ్వరాలయం, తిక్కన పూజించిన శిలలు అన్నీ తమ సొంతమని, ప్రభుత్వానికీ దేవాదాయశాఖకూ సంబంధం లేదని పాటూరు వంశస్థుడు ఒకాయన ఆలయ పునర్ని ర్మాణాన్ని అడ్డుకొన్నా రట. పదేళ్ల కిం దట మాత్రం ఒక భక్తుడు శిథిల ఆలయానికి వెల్ల వేయించి తన భక్తిని చాటుకొన్నా రని చెబుతారు.

పాటూరు గ్రామంలో తిక్కన విగ్రహాన్ని ప్రతిష్ఠించాలనే ఆలోచన ప్రభుత్వానికి ఇప్పటికీ లేకపోవడం విచారకరమని గ్రామస్థులు అన్నా రు. తిక్కన గురించి రాసిన వ్యాసాలు, గ్రంథాలతో ఒక గ్రం« థాలయం ఏర్పాటు చేయాల్సిందిగా ప్రజలు కోరుతున్నా రు. హైదరాబాదులోని టాంకుబండ్‌పై తిక్కన విగ్రహాన్ని ప్రతిష్ఠించిన రాష్ట్ర ప్రభుత్వం ఆయన నివసించిన పాటూరు గ్రామాన్ని మరచిపోవడం బాధాకరం. ఆయన పూజించి, యజ్ఞం చేసిన సిద్ధేశ్వరాలయాన్ని ప్రభుత్వం దర్శనీయ స్థలాల జాబితాలో చేర్చాలని జిల్లా వాసులు, సాహిత్యాభిలాషులు కోరుతున్నా రు. బ్రిటిషువారు నిర్మించిన కట్టడాల్ని సైతం చారిత్రక కట్టడాలుగా ప్రాధాన్యత కల్పించిన రాష్ట్ర ప్రభుత్వం తిక్కన తిరుగాడిన నేల స్మృతులు ... శిల్పా ల్ని, ఘంటాన్ని, ఒరను, నందీశ్వరుడ్ని పదిలపరచకపోవడం విచారకరం. తెలుగు జాతి గుండెల్లో తీయ తేనియ నుడుల్ని ఆచంద్రార్కం నిల్పిన తిక్కన జ్ఞాపకార్థం ఈ పని చేయాల్సిన అవసరం ఉంది.

- మడపర్తి రవీంద్ర, ఆన్‌లైన్, కోవూరు (Andhrajyothi sunday magazine-30/01/2011) మూలాలు

1.2.2 Text Processing

The term text processing refers to the automation of analyzing electronic text. This allows machine learning models to get structured information about the text to use for analysis, manipulation of the text, or to generate new text. Text processing is one of the most common tasks used in machine learning applications such as language translation, sentiment analysis, spam filtering, and many others.

Since we naturally communicate in words, not numbers, companies receive a lot of raw text data via emails, chat conversations, social media, and other channels. This unstructured data is filled with insights and opinions about different topics, products, and services, but companies' first need to organize, sort, and measure textual data to access this valuable information.

Text processing refers to only the analysis, manipulation, and generation of text, while natural language processing refers to the ability of a computer to understand human language in a valuable way. Basically, natural language processing is the next step after text processing.

Text processing is very important as it is one of the machine learning uses that average technology consumers don't even realize they're using, but most people use apps daily that are using text processing behind the scenes.

Since our interactions with brands have become increasingly online and text-based, text data is one of the most important ways for companies to derive business insights. Text data can show a business how their customers search, buy, and interact with their brand, products, and competitors online. Text processing with machine learning allows enterprises to handle these large amounts of text data.

1.2.3 Natural Language Processing

The study of natural language processing has been around for more than 50 years and grew out of the field of linguistics with the rise of computers. As a human, you may speak and write in English, Spanish or Chinese. But a computer’s native language – known as machine code or machine language – is largely incomprehensible to most people. At your device’s lowest levels, communication occurs not with words but through millions of zeros and ones that produce logical actions.

Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software.

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding.

It is very important as Natural language processing helps computers communicate with humans in their own language and scales other language-related tasks. For example, NLP makes it possible for computers to read text, hear speech, interpret it, measure sentiment and determine which parts are important.

Today’s machines can analyze more language-based data than humans, without fatigue and in a consistent, unbiased way. Considering the staggering amount of unstructured data that’s generated every day, from medical records to social media, automation will be critical to fully analyze text and speech data efficiently.

While supervised and unsupervised learning, and specifically deep learning, are now widely used for modeling human language, there’s also a need for syntactic and semantic understanding and domain expertise that are not necessarily present in these machine learning approaches. NLP is important because it helps resolve ambiguity in language and adds useful numeric structure to the data for many downstream applications, such as speech recognition or text analytics.

1.2.4 Language modeling

Language Models help in predicting the next few words in sequence from the past history. The prediction can be done mainly using Markov assumption. The assumption states that having the knowledge of the full history, one can make future predictions only on its present state. In case of N-gram models this assumption can be extended such that conditional probability may depend on a couple of previous words .For Example, In Bigram model the probability of any word only depends on previous word whereas in Trigram model probability of any word only depends on the previous two words.

Applications of Language Models:

1. Spelling Correction

Spelling correction is not a trivial task for a computer. Better and better models are invented to tackle problems such as spelling correction. Language models are the kind of models that are being used for this task.

2. Speech Recognition

Smart speakers, such as Alexa, use automatic speech recognition (ASR) mechanisms for translating the speech into text. It translates the spoken words into text and between this translation, the ASR mechanism analyzes the intent/sentiments of the user by differentiating between the words. For example, analyzing homophone phrases such as “Let her” or “Letter”, “But her” “Butter”.

3. Machine Translation

When translating a Chinese phrase “我在吃” into English, the translator can give several choices as output:
I eat lunch, I am eating, Me am eating, Eating am I
Here, the language model tells that the translation “I am eating” sounds natural and will suggest the same as output.

4. Handwriting Recognition:

Handwriting recognition (HWR), also known as Handwritten Text Recognition (HTR), is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The image of the written text may be sensed "off line" from a piece of paper by optical scanning (optical character recognition) or intelligent word recognition. Alternatively, the movements of the pen tip may be sensed "on line", for example by a pen-based computer screen surface, a generally easier task as there are more clues available. A handwriting recognition system handles formatting, performs correct segmentation into characters, and finds the most plausible words.

5. Information retrieval (IR)

Information retrieval (IR) is the process of obtaining information system resources that are relevant to an information need from a collection of those resources. Searches can be based on full-text or other content-based indexing. Information retrieval is the science of searching for information in a document, searching for documents themselves, and also searching for the metadata that describes data, and for databases of texts, images or sounds.

1.3 Organization of Project

The technique which is developed is taking input as a word and checks the modal dictionary for the most probable next word and displays the predicted word as a result. If it is Bi-Gram then the predicted word depends on the one word entered, and if it is Tri-gram then the predicted word depends on the two history words entered as input.

We have used.

- Bi-Gram
- Tri-Gram

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language	: Python 3.6
Dataset	: Telugu Wiki Data
Packages	: cltk, collections
Tool	: Jupyter Notebook

2.1.2 Hardware Requirements

Operating System	: Windows 10
Processor	: Intel Core i3-2348M
CPU Speed	: 2.30 GHz
Memory	: 8 GB (RAM)

2.2 Technologies Description

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readability and code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Telugu Wiki Dataset

The dataset for the experiment is downloaded from the various Telugu Wikipedia pages.It is a collection of different domains like *districts, Poems, Poets, History etc*. The Total of 5 lakhs sentences of wiki documents are extracted to form a Dataset. The xtracted wiki documents data, 5,00,000 clean sentences (which mostly has Telugu words) are taken for model development and testing.

Classical Language Toolkit

The Classical Language Toolkit (CLTK) is a Python library offering natural language processing (NLP) for the languages of pre–modern Eurasia. Pre-configured pipelines are available for 19 languages. CLTK is implemented from NLTK.

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Jupyter Notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

The main uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning.

3. DESIGN

3.1 Introduction

- Language Modelling is used to build a language model that provides context to distinguish between words and phrases that sound similar.
- A statistical language model provides likelihood of different string P(s) over given strings S.
- Language Model basically predicts the next word in a given sequence by assigning a probability to a sequence of words

$$P(wn)=P(wn|w1,w2,w3,w4,w5,...,wn-1)$$

$$P_{Laplace}\left(\frac{w_i}{w_i-1}\right)=\frac{\text{count(wi-1,wi)}}{\text{count(wi-1)}}$$

A language model is a simplified statistical model of text. It is a data driven approach instead of the many rule based approaches that exist. Rule based approaches are based on expert knowledge. The problem here is that languages (at least the human language) are approximately infinite. Every word has at least many variants and to make things worse, words can be combined to form new words. If a new brand pops up, then new words are introduced. So language is changing over time. That means that a static set of rules can never define all aspects of a language. Therefore, we need another approach to model language. This can be done by a statistical language model.

Statistical Language Modeling, or Language Modeling and LM for short, is the development of probabilistic models that are able to predict the next word in the sequence given the words that precede it. Language modeling is the task of assigning a probability to sentences in a language. Besides assigning a probability to each sequence of words, the language models also assigns a probability for the likelihood of a given word to follow a sequence of words

A language model learns the probability of word occurrence based on examples of text. Simpler models may look at a context of a short sequence of words, whereas larger models may work at the level of sentences or paragraphs. Most commonly, language models operate at the level of words.A language model can be developed and used standalone, such as to generate new sequences of text that appear to have come from the corpus.

Markov Assumption

- The **Markov assumption** states that only a limited number of previous words affect the probability of the next word.
- It makes predictions for the future of the process based solely on its present state.

$$P(wn|w1,w2,w3,w4,w5,...,wn-1)=P(wn|wn-1)$$

Example: Sentence : ‘weather is pleasant’

a). Probability of this sentence will be expressed as:

$P(\text{"weather is pleasant"})=P(weather)\times P(is|weather)\times P(pleasant|weatheris)$

b). Based on Markov’s assumption we can rewrite conditional probability:

$P(pleasant|weatheris)\approx P(pleasant|weather)$

N-Gram:

Natural Language Processing, or NLP, n-grams are used for a variety of things. Some examples include auto completion of sentences, such as the one we see in Gmail these days, auto spell check and even we can check for grammar in a given sentence.

An *n*-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a ($n-1$)–order Markov model. *n*-gram models are now widely used in probability, communication theory, computational linguistics mostly in statistical natural language processing, computational biology mostly in biological sequence analysis, and data compression. Two benefits of *n*-gram models are simplicity and scalability, with larger *n*, a model can store more context with a well-understood space, time tradeoff, enabling small experiments to scale up efficiently.

The n fields of computational linguistics and probability, an *n*-gram is a contiguous sequence of *n* items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The *n*-grams typically are collected from a text or speech corpus.

Using Latin numerical prefixes, an *n*-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" ; size 3 is a "trigram". English cardinal numbers are sometimes used, e.g., "four-gram", "five-gram", and so on. In computational biology, a polymer or oligomer of a known size is called a *k*-mer instead of an *n*-gram, with specific names using Greek numerical prefixes such as "monomer", "dimer", "trimer", "tetramer", "pentamer", etc., or English cardinal numbers, "one-mer", "two-mer", "three-mer", etc. Using these n-grams and the probabilities of the occurrences of certain words in certain sequences could improve the predictions of auto completion systems.

The N-gram model, like many statistical models, is significantly dependent on the training corpus. As a result, the probabilities often encode particular facts about a given training corpus. Besides, the performance of the N-gram model varies with the change in the value of N. Moreover, you may have a language task in which you know all the words that can occur, and hence we know the vocabulary size V in advance. The closed vocabulary assumption assumes there are no unknown words, which is unlikely in practical scenarios.

Bi-Gram:

A bigram or diagram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. A bigram is an *n*-gram for size n is 2. The frequency distribution of every bigram in a string is commonly used for simple statistical analysis of text in many applications, including in computational linguistics, cryptography, speech recognition, and so on.

Gappy bigrams or *skipping bigrams* are word pairs which allow gaps, perhaps avoiding connecting words, or allowing some simulation of dependencies, as in a dependency grammar.

Bigrams help provide the conditional probability of a token given the preceding token, when the relation of the conditional probability is applied:

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

That is, the probability

- P() of a token
- W_{n} given the preceding token
- W_{n-1} is equal to the probability of their bigram, or the co-occurrence of the two tokens
- P(W_{n-1},W_{n})}, divided by the probability of the preceding token.

Applications of Bigrams are used in most successful language models for speech recognition. They are a special case of N-gram.

Bigram frequency attacks can be used in cryptography to solve cryptograms. See frequency analysis.

Bigram frequency is one approach to statistical language identification.

Some activities in logology or recreational linguistics involve bigrams. These include attempts to find English words beginning with every possible bigram, or words containing a string of repeated bigrams, such as *logogogue*.

Tri-Gram:

Trigrams are a special case of the *n*-gram, where *n* is 3. They are often used in natural language processing for performing statistical analysis of texts and in cryptography for control and use of ciphers and codes.
Cryptography, or cryptology, is the practice and study of techniques for secure communication in the presence of third parties called adversaries. More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages; various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography. Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, electrical engineering, communication science, and physics. Applications of cryptography include electronic commerce, chip-based payment cards, digital currencies, computer passwords, and military

communications.

In cryptography, a cipher (or cypher) is an algorithm for performing encryption or decryption—a series of well-defined steps that can be followed as a procedure. An alternative, less common term is *encipherment*. To encipher or encode is to convert information into cipher or code. In common parlance, "cipher" is synonymous with "code", as they are both a set of steps that encrypt a message; however, the concepts are distinct in cryptography, especially classical cryptography.

Encryption is the process of encoding information. This process converts the original representation of the information, known as plaintext, into an alternative form known as ciphertext. Ideally, only authorized parties can decipher a ciphertext back to plaintext and access the original information. Encryption does not itself prevent interference but denies the intelligible content to a would-be interceptor.

Smoothing Techniques:

Smoothing techniques in NLP are used to address scenarios related to determining probability / likelihood estimate of a sequence of words (say, a sentence) occurring together when one or more words individually (unigram) or N-grams such as bigram(wi/wi−1) or trigram (wi/wi−1wi−2) in the given set have never occured in the past.

The following is the list of some of the smoothing techniques:

- Laplace smoothing
- Additive smoothing
- Good-turing smoothing
- Kneser-Ney smoothing
- Katz smoothing
- Church and Gale Smoothing

Based on the training data set, what is the probability of "cats sleep" assuming bigram technique is used? Based on bigram technique, the probability of the sequence of words "cats sleep" can be calculated as the product of following:

$$P(catssleep) = P(\frac{cats}{<s>}) \times P(\frac{sleep}{cats}) \times P(\frac{</s>}{sleep})$$

You will notice that $P(\frac{sleep}{cats}) = 0.$ Thus, the overall probability of occurrence of "cats sleep" would result in zero (0) value. However, the probability of occurrence of a sequence of words should not be zero at all.

This is where various different smoothing techniques come into the picture.

Laplace or Add-one Smoothing:

In Laplace smoothing, 1 (one) is added to all the counts and thereafter, the probability is calculated. This is one of the most trivial smoothing techniques out of all the techniques.

Maximum likelihood estimate (MLE) of a word wi occuring in a corpus can be calculated as the following. N is total number of words, and count(wi) is count of words for whose probability is required to be calculated

$$\textbf{MLE: } P(w_i) = \frac{count(w_i)}{N}$$

After applying Laplace smoothing, the following happens. Adding 1 leads to extra V observations.

$$P_{Laplace}(w_i) = \frac{count(w_i)+1}{N+V}$$

Similarly, for N-grams (say, Bigram), MLE is calculated as the following:

$$P(\frac{w_i}{w_{i-1}}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

After applying Laplace smoothing, the following happens for N-grams (Bigram). Adding 1 leads to extra V observations.

$$P_{Laplace}(\frac{w_i}{w_{i-1}}) = \frac{count(w_{i-1}, w_i)+1}{count(w_{i-1})+V}$$

Add- Alpha:

Add-one smoothing gives undue credence to add-one smoothing counts that we do not observe. We can remedy this by add-a smoothing, adding a smaller number add-a smoothing a < 1 instead of one.

$$P_{Laplace}\left(\frac{w_i}{w_i-1}\right)=\frac{\text{count}(wi-1,wi)+\alpha}{\text{count}(wi-1)+\alpha|V|}$$

3.2 Architecture Diagram

An architectural diagram is a diagram of a system that is used to abstract the overall outline of the software system and the relationships, constraints, and boundaries between components. It is an important tool as it provides an overall view of the physical deployment of the software system and its evolution roadmap.

The main purpose of architectural diagrams should be to facilitate collaboration, to increase communication, and to provide vision and guidance. Paint one or two high-level diagrams on the wall and use them during meetings (stand-ups, etc).

Architecture, the art and technique of designing and building, as distinguished from the skills associated with construction. The practice of architecture is employed to fulfill both practical and expressive requirements, and thus it serves both utilitarian and aesthetic ends.

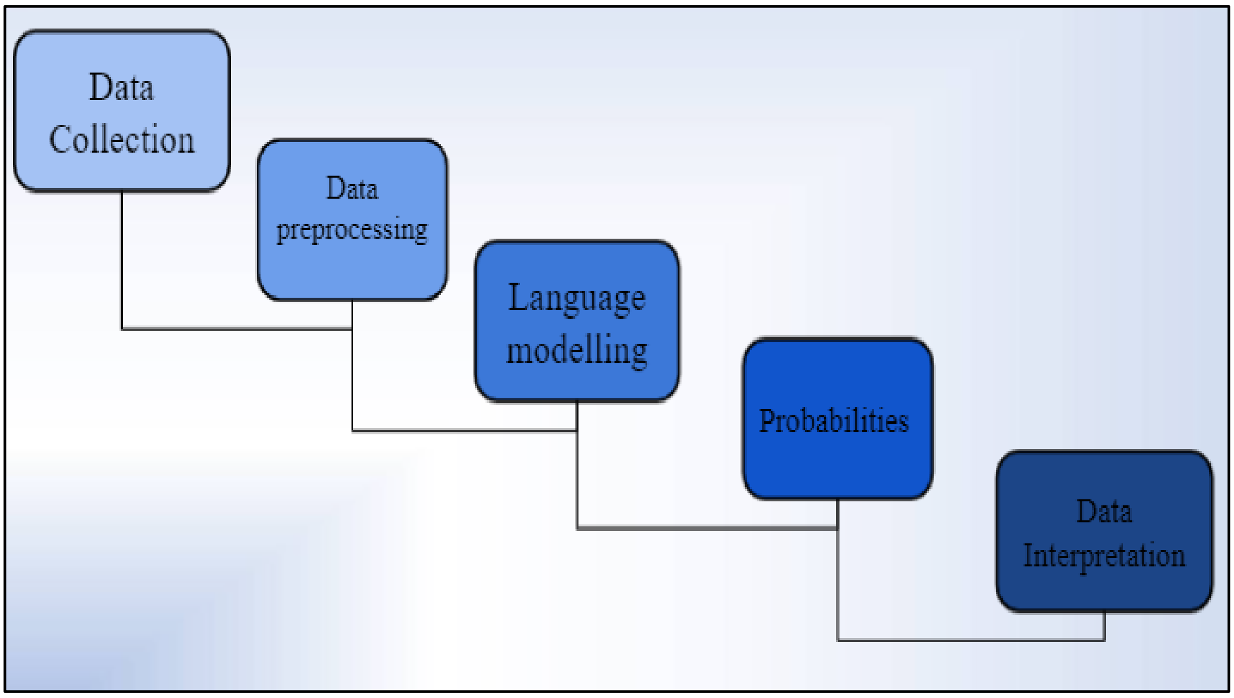


Fig 3.2: Architecture Diagram

3.3 UML Diagrams

3.3.1 Use Case Diagram

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagrams are one of them. Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consist of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

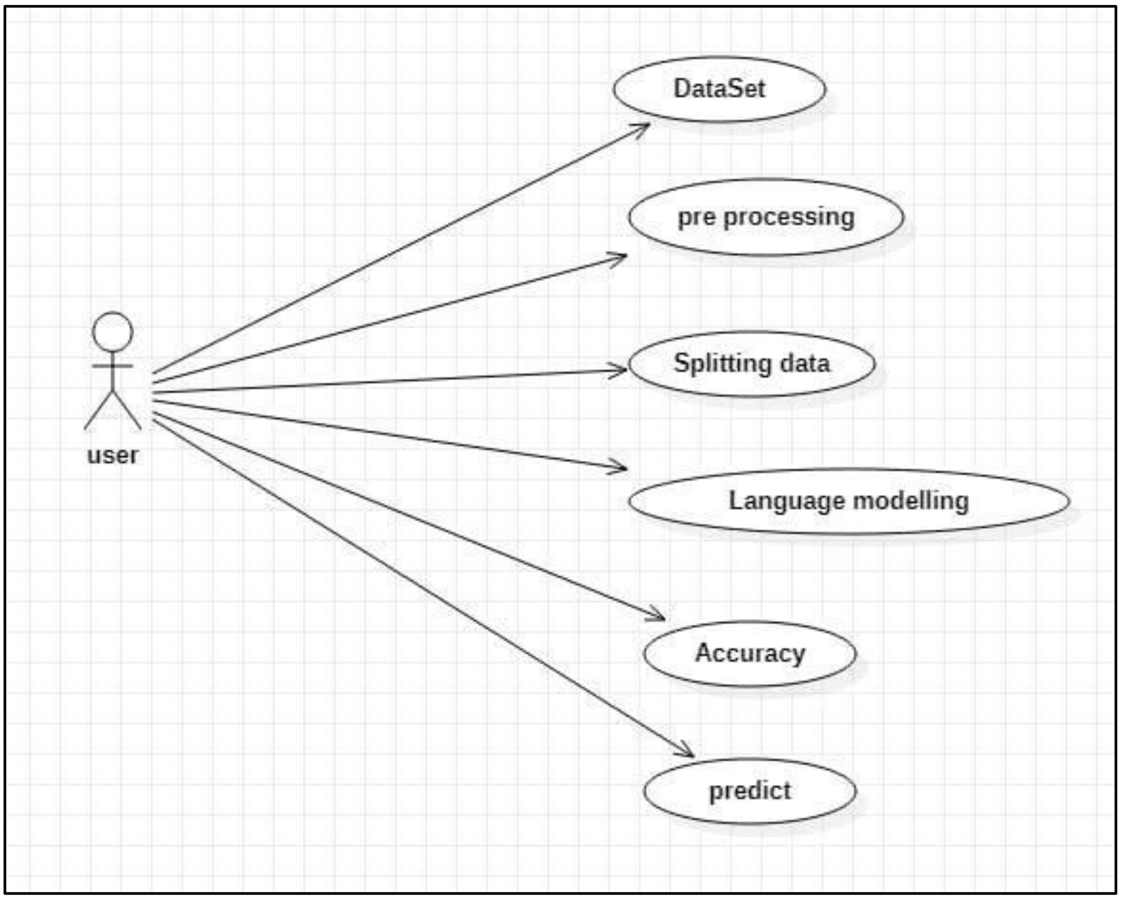


Fig 3.2.1: Use Case Diagram

3.3.2 Sequence Diagram

Sequence Diagrams Represent the objects participating in the interaction horizontally and time vertically. A Use Case is a kind of behavioral classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling.

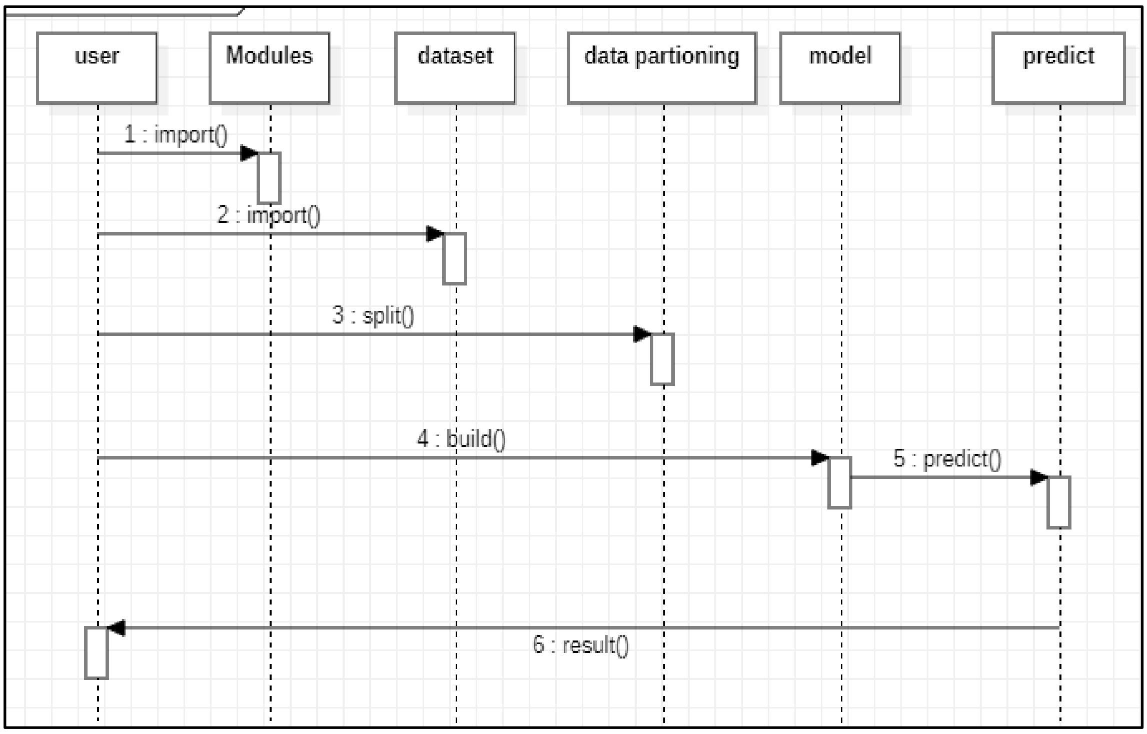


Fig 3.3.2: Sequence Diagram

3.3.3 Activity Diagram

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

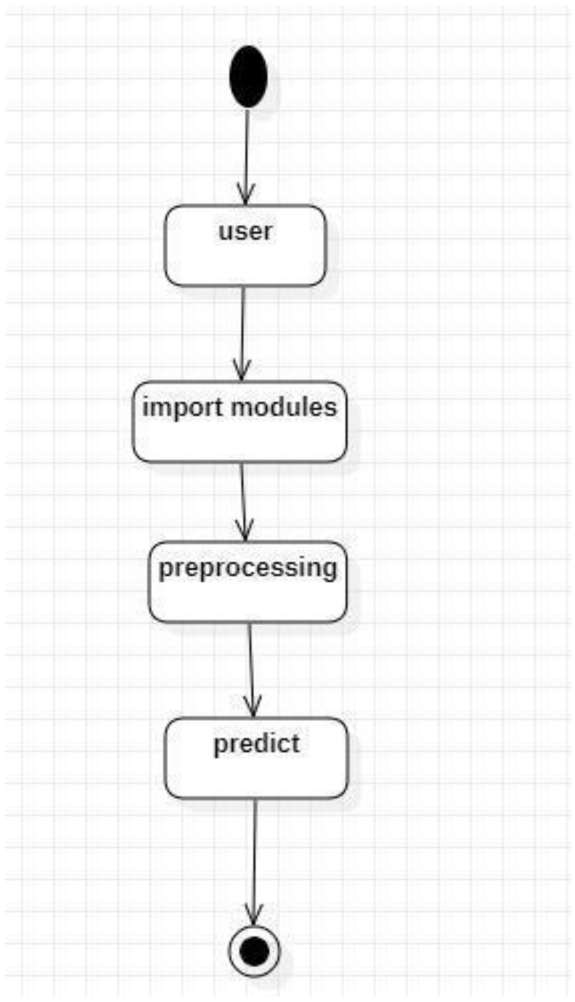


Fig 3.2.3: Activity Diagram

3.3.4 Class Diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the system of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

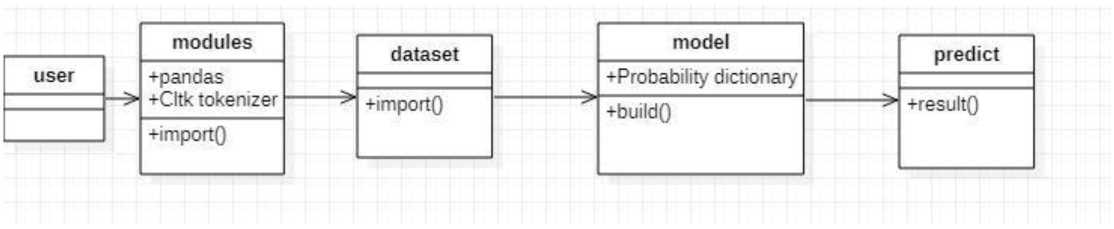


Fig 3.2.5: Class Diagram

4. IMPLEMENTATION

4.1 Code

4.1.1 Bi-Gram

```
#importing  packages
import re
import string
import collections
import math as m

#installing cltk
!pip install cltk==v0.1.99
from cltk.tokenize.sentence import TokenizeSentence
tokenizer = TokenizeSentence('telugu')

#loading and converting the train data into a list of sentences
#loading and converting the test data into a list of sentences
path='/home/karthik/Desktop/keeru/datasets/tewiki_all.txt'
with open(path) as myfile:
lines = [line.split('. ') for line in myfile.readlines()]
testlines = lines[450000:500000]
lines = lines[:450000]

#function removing the html tags,punctuations,numbers,english words
def preprocessing_data(str):
u = re.sub('<[^<]+?>','',str)
o = re.sub('\u200c',' ',u)
table = str.maketrans({key: None for key in string.punctuation})
translated = o.translate(table)
w = re.sub('[a-zA-Z]*','',translated)
d = re.sub(' +',' ',w)
y = re.sub("\n",'',d)
return y
```

```
#getting a list of sentences that are clean
def get_preprocessed_sentences(sentences):
tokenized_sentences=[]
for i in range(0,len(sentences)):
temp = []
for j in range(0,len(sentences[i])):
final = preprocessing_data(sentences[i][j])
if len(final) != 0:
temp.append(final)
tokenized_sentences.append(temp)
return tokenized_sentences

train_preprocessed_sentences = get_preprocessed_sentences(lines)
test_preprocessed_sentences = get_preprocessed_sentences(testlines)

#function to tokenize the sentences
def getting_sentences(line):
sentences = []
for i in range(0,len(line)):
for j in range(0,len(line[i])):
sentence = line[i][j]
s = tokenizer.tokenize(sentence)
sentences.append(s)
return sentences
train_sentences = getting_sentences(train_preprocessed_sentences)
test_sentences = getting_sentences(test_preprocessed_sentences)




# adding start and end delimiter, then splitting
def delimiting_spliting(tokenized_sentences):
tokenized_sentences = [x for x in tokenized_sentences if x != ['\n'] and x != [' \n']]
```

```
sentences_with_delimiter = []
for i in range(0,len(tokenized_sentences)):
s = tokenized_sentences[i]
if len(s) == 0:
continue
temp = ['<s> ']+s+[' </s>']                 #adding start and end delimiter
sentences_with_delimiter.append(temp)
print(sentences_with_delimiter[:10])
return sentences_with_delimiter
split_train_data = delimiting_spliting(train_sentences)
split_test_data = delimiting_spliting(test_sentences)

#getting a single list of words
def getting_words_list(split_data):
list = [item for sublist in split_data for item in sublist]
return list
words_train = getting_words_list(split_train_data)   #train data words
words_test = getting_words_list(split_test_data)       #test data words

# getting number of unique words in train data
unique_words = len(set(words_train))

#getting the dictionary(model) from the train data, where keys are the history words and values are
#another dictionary which contains keys as the next word and its values corresponding to the number
#of times that words has occurred



dic = {}
for i in range(0,len(words_train)-1):
t = words_train[i]
s = dic.get(t)
next = i+1
```

```
if s == None:
dic[t] = [words_train[next]]
else:
s.append(words_train[next])
temp = {t:s}
dic.update(temp)

#test_dic is a dictionary where keys are the history words of the test data and values is the list of next words
test_dic = {}
for i in range(0,len(words_test)-1):
t = words_test[i]
s = test_dic.get(t)
next = i+1
if s == None:
test_dic[t] = [words_test[next]]
else:
s.append(words_test[next])
temp = {t:s}
test_dic.update(temp)

# getting the count of each element in the list
def frequency_count(list):
c = collections.Counter(list)
count_dic={}
for i in range(0,len(list)):
count_dic[list[i]] = c[list[i]]
return count_dic
# updating the dic with the word frequency
for word in dic:
s = dic.get(word)
value = frequency_count(s)
temporary = {word:value}
```

```
dic.update(temporary)

# getting the probability of each predicted word
def get_probability(s1,s2):
firstWord = s1
nextWord = s2
inner_dic = dic.get(firstWord)
if inner_dic == None:
return 0
else:
numerator = inner_dic.get(nextWord)
if numerator == None:
return 0
else:
denominator = sum(inner_dic.values())
return numerator/denominator

# getting the add-one probability of each predicted word
def get_addOne_probability(s1,s2):
firstWord = s1
nextWord = s2
inner_dic = dic.get(firstWord)
if inner_dic == None:
return 0
else:
numerator = inner_dic.get(nextWord)
if numerator == None:
numerator = 1
else:
numerator = numerator + 1
denominator = sum(inner_dic.values()) + unique_words
return numerator/denominator
```

```
# getting the add-alpha probability of each predicted word
def get_addAlpha_probability(s1,s2):
firstWord = s1
nextWord = s2
alpha = 0.1
inner_dic = dic.get(firstWord)
if inner_dic == None:
return 0
else:
numerator = inner_dic.get(nextWord)
if numerator == None:
numerator = alpha
else:
numerator = numerator + alpha
denominator = sum(inner_dic.values()) + (unique_words*alpha)
return numerator/denominator

#preparing probability dictionaries with corresponding probabilities
prob_dic = {}
addOneProb_dic = {}
addAlphaProb_dic = {}
for i in range(0,len(words_test)-2):
bigram = words_test[i]+words_test[i+1]
prob = get_probability(words_test[i],words_test[i+1])
addOneProb = get_addOne_probability(words_test[i],words_test[i+1])
prob_dic[bigram]=prob
addOneProb_dic[bigram] = addOneProb
addAlphaProb = get_addAlpha_probability(words_test[i],words_test[i+1])
addAlphaProb_dic[bigram] = addAlphaProb

# computing perplexity Score
def perplexityScore(prob_dic):
logProb = 0
```

```
num = 0
for word in prob_dic:
p = prob_dic.get(word)
if p == 0:
logProb = logProb
else:
w = m.log(p**-1,10)
logProb = logProb + w
num = num + 1
return (2**(logProb/num))

# perplexity Scores for different cases
ans = perplexityScore(prob_dic)
print(ans)
ans1 = perplexityScore(addOneProb_dic)
print(ans1)
ans2 = perplexityScore(addAlphaProb_dic)
print(ans2)

#input interface
s1 = input("Enter the 1st word " )
input_word = s1
print(input_word)
def getting_2nd(input_word):
s = dic.get(input_word)
if s == None:
print("not found")
return
else:
maximum = max(s, key=s.get)
return maximum
print(getting_2nd(input_word))
```

4.1.2 Tri-Gram

```
#importing packages
import re
import string
import collections
import math as m

#intalling cltk
!pip install cltk
from cltk.tokenize.sentence import TokenizeSentence
tokenizer = TokenizeSentence('telugu')

path='/home/karthik/Desktop/keeru/datasets/tewiki_all.txt'
with open(path) as myfile:
lines = [line.split('. ') for line in myfile.readlines()]
testlines = lines[450000:500000]
lines = lines[:45000]

#function removing the html tags,punctuations,numbers,english words
def preprocessing_data(str):
u = re.sub('<[^<]+?>','',str)
o = re.sub('\u200c',' ',u)
table = str.maketrans({key: None for key in string.punctuation})
translated = o.translate(table)
w = re.sub('[a-zA-Z]*','',translated)
d = re.sub(' +',' ',w)
y = re.sub("\n",'',d)
return y

#getting a list of sentences that are clean
def get_preprocessed_sentences(sentences):
tokenized_sentences=[]
```

```
for i in range(0,len(sentences)):
temp = []
for j in range(0,len(sentences[i])):
final = preprocessing_data(sentences[i][j])
if len(final) != 0:
temp.append(final)
tokenized_sentences.append(temp)
return tokenized_sentences

train_preprocessed_sentences = get_preprocessed_sentences(lines)
test_preprocessed_sentences = get_preprocessed_sentences(testlines)

#function to tokenize the sentences
def getting_sentences(line):
sentences = []
for i in range(0,len(line)):
for j in range(0,len(line[i])):
sentence = line[i][j]
s = tokenizer.tokenize(sentence)
sentences.append(s)
return sentences
train_sentences = getting_sentences(train_preprocessed_sentences)
test_sentences = getting_sentences(test_preprocessed_sentences)

# adding start and end delimiter, then splitting
def delimiting_spliting(tokenized_sentences):
tokenized_sentences = [x for x in tokenized_sentences if x != ['\n'] and x != [' \n']]
sentences_with_delimiter = []
for i in range(0,len(tokenized_sentences)):
s = tokenized_sentences[i]
if len(s) == 0:
continue
temp = ['<s> ']+s+[' </s>']                    #adding start and end delimiter
```

```
sentences_with_delimiter.append(temp)
return sentences_with_delimiter
split_train_data = delimiting_spliting(train_sentences)
split_test_data = delimiting_spliting(test_sentences)

#getting a single list of words
def getting_words_list(split_data):
list = [item for sublist in split_data for item in sublist]
return list

words_train = getting_words_list(split_train_data)   #train data words
words_test = getting_words_list(split_test_data)      #test data words

# getting number of unique words in train data
unique_words = len(set(words_train))


#getting the dictionary(model) from the train data, where keys are the history words and values are
#another dictionary which contains keys as the next word and its values corresponding to the number
#of times that words has occurred
dic = {}
for i in range(0,len(words_train)-2):
t = words_train[i] +'_'+ words_train[i+1]
s = dic.get(t)
next = i+2
if s == None:
dic[t] = [words_train[next]]
else:
 s.append(words_train[next])
temp = {t:s}
 dic.update(temp)
```

```
#test_dic is a dictionary where keys are the history words of the test data and values is
the list of next words

test_dic = {}
for i in range(0,len(words_test)-2):
t = words_test[i] +'_'+ words_test[i+1]
s = test_dic.get(t)
next = i+2
if s == None:
test_dic[t] = [words_test[next]]
else:
s.append(words_test[next])
temp = {t:s}
test_dic.update(temp)

# getting the count of each element in the list
def frequency_count(list):
c = collections.Counter(list)
count_dic={}
for i in range(0,len(list)):
count_dic[list[i]] = c[list[i]]
return count_dic

# updating the dic with the word frequency
for word in dic:
s = dic.get(word)
value = frequency_count(s)
temporary = {word:value}
dic.update(temporary)

# getting the probability of each predictable word
def get_probability(s1,s2,s3):
twoWords = s1+'_'+s2
```

```
nextWord = s3
inner_dic = dic.get(twoWords)
if inner_dic == None:
return 0
else:
numerator = inner_dic.get(nextWord)
if numerator == None:
return 0
else:
denominator = sum(inner_dic.values())
return numerator/denominator

# getting the add-one probability of each predictable word
def get_addOne_probability(s1,s2,s3):
twoWords = s1+'_'+s2
nextWord = s3
inner_dic = dic.get(twoWords)
if inner_dic == None:
return 0
else:
numerator = inner_dic.get(nextWord)
if numerator == None:
numerator = 1
else:
numerator = numerator + 1
denominator = sum(inner_dic.values()) + unique_words
return numerator/denominator

# getting the add-alpha probability of each predictable word
def get_addAlpha_probability(s1,s2,s3):
twoWords = s1+'_'+s2
nextWord = s3
alpha = 0.1
```

```
inner_dic = dic.get(twoWords)
if inner_dic == None:
return 0 #alpha/(alpha*unique_words)
else:
numerator = inner_dic.get(nextWord)
if numerator == None:
numerator = alpha
else:
numerator = numerator + alpha
denominator = sum(inner_dic.values()) + (unique_words*alpha)
return numerator/denominator

#preparing probability dictionaries with corresponding probabilities
prob_dic = {}
addOneProb_dic = {}
addAlphaProb_dic = {}
for i in range(0,len(words_test)-3):
trigram = words_test[i]+'_'+words_test[i+1]+'_'+words_test[i+2]
prob = get_probability(words_test[i],words_test[i+1],words_test[i+2])
prob_dic[trigram]=prob
addOneProb=get_addOne_probability(words_test[i],words_test[i+1],words_test[i+2])
addOneProb_dic[trigram] = addOneProb
addAlphaProb=get_addAlpha_probability(words_test[i],words_test[i+1],words_test[i
+2])
addAlphaProb_dic[trigram] = addAlphaProb

# computing perplexity Score
def perplexityScore(prob_dic):
logProb = 0
num = 0
for word in prob_dic:
p = prob_dic.get(word)
if p == 0:
```

```
logProb = logProb
else:
w = m.log(p**-1,10)
logProb = logProb + w
num = num + 1
return (2**(logProb/num))

# perplexity Scores for different cases
ans = perplexityScore(prob_dic)
print(ans)
ans1 = perplexityScore(addOneProb_dic)
print(ans1)
ans2 = perplexityScore(addAlphaProb_dic)
print(ans2)

#input interface
s1 = input("Enter the 1st word " )
s2 = input("Enter the 2nd word " )
input_word = s1 +'_'+ s2
def getting_3rd(input_word):
s = dic.get(input_word)
if s == None:
print("not found")
return
else:
maximum = max(s, key=s.get)
return maximum
getting_3rd(input_word)
```

4.2 Evaluation Metric

4.2.1 Perplexity

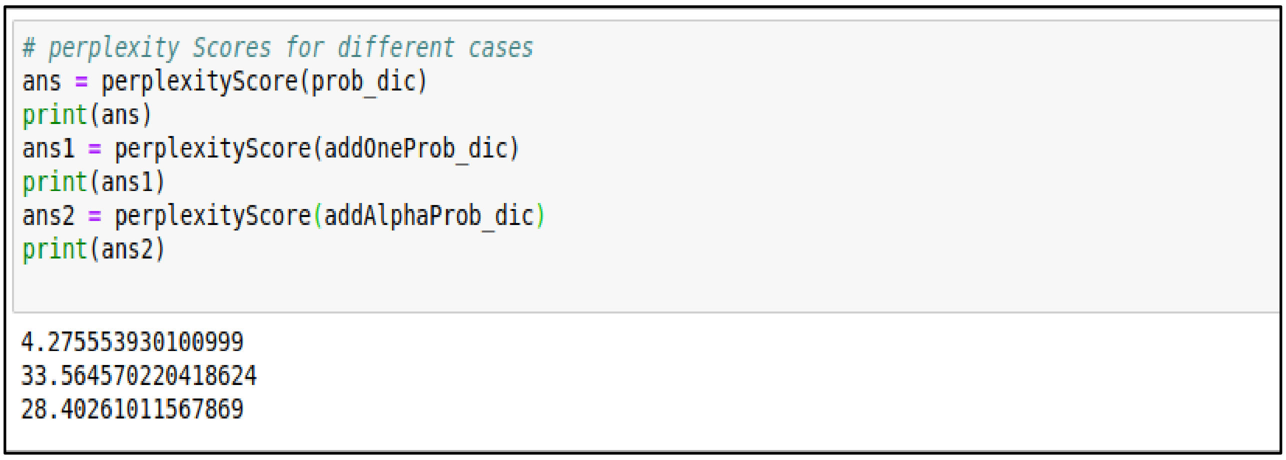
Perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample. It is an Evaluation Metric for N-Gram Models. In natural language processing, perplexity is a way of evaluating language models. A language model is a probability distribution over entire sentences or texts.Perplexity is the inverse probability of the test set normalized by number of words. It is the weighted average number of choices a random variable can make i.e the number of possible next words that can be followed by a given word.

$$PP(p) := 2^{H(p)} = 2^{-\sum_x p(x)\log_2 p(x)} = \prod_x p(x)^{-p(x)}$$

The Entropy may be calculated as cross entropy

$$H(\tilde{p}, q) = -\sum_x \tilde{p}(x)\log_2 q(x)$$

4.2.2 Bi-Gram Perplexity scores

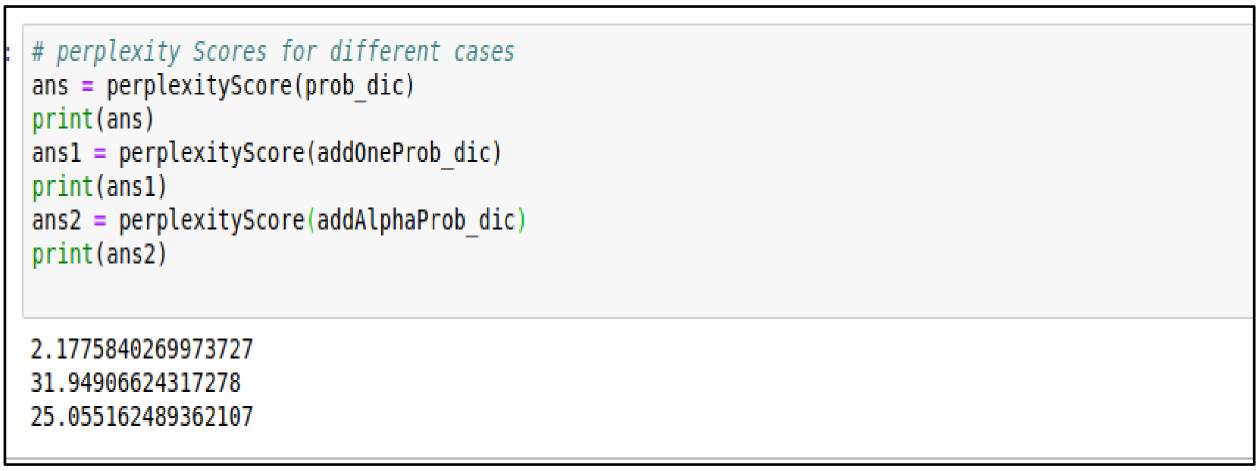


```
# perplexity Scores for different cases
ans = perplexityScore(prob_dic)
print(ans)
ans1 = perplexityScore(addOneProb_dic)
print(ans1)
ans2 = perplexityScore(addAlphaProb_dic)
print(ans2)
```

```
4.275553930100999
33.564570220418624
28.40261011567869
```

Fig 4.2.2

4.2.3 Tri-Gram Perplexity scores



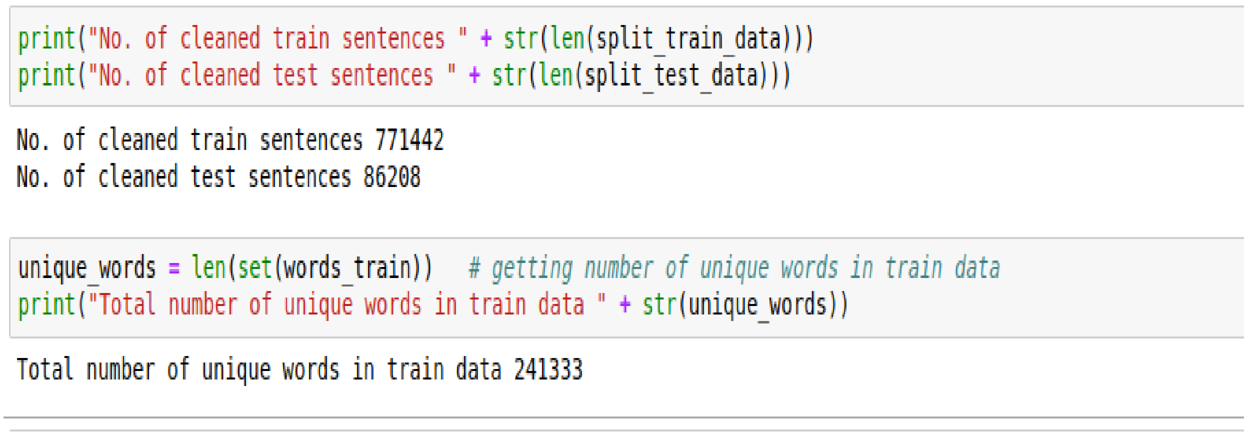
```
# perplexity Scores for different cases
ans = perplexityScore(prob_dic)
print(ans)
ans1 = perplexityScore(addOneProb_dic)
print(ans1)
ans2 = perplexityScore(addAlphaProb_dic)
print(ans2)
```

```
2.1775840269973727
31.94906624317278
25.055162489362107
```

Fig 4.2.3

4.3 Dataset Analyzing

4.3.1 No. of Tests and Training Sentences.



```
print("No. of cleaned train sentences " + str(len(split_train_data)))
print("No. of cleaned test sentences " + str(len(split_test_data)))
```

```
No. of cleaned train sentences 771442
No. of cleaned test sentences 86208
```

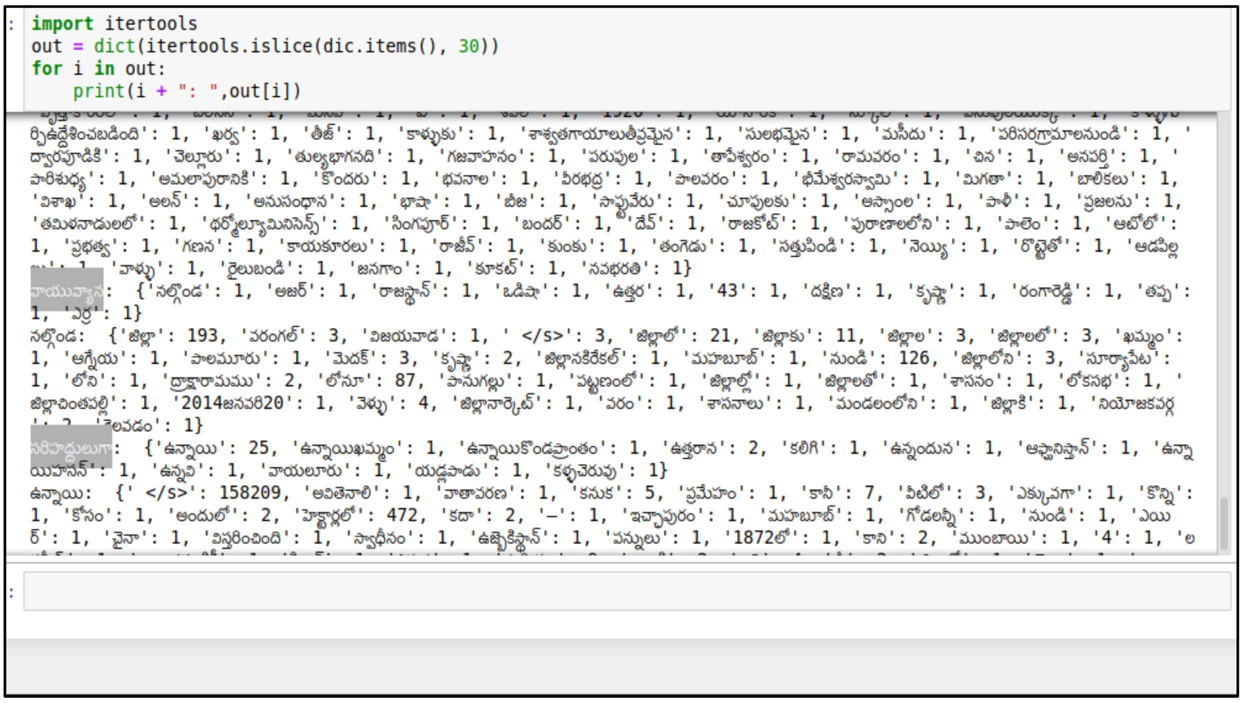
```
unique_words = len(set(words_train))   # getting number of unique words in train data
print("Total number of unique words in train data " + str(unique_words))
```

```
Total number of unique words in train data 241333
```

Fig 4.3.1

4.4 Model Dictionary Screenshots

4.4.1 Bi-Gram

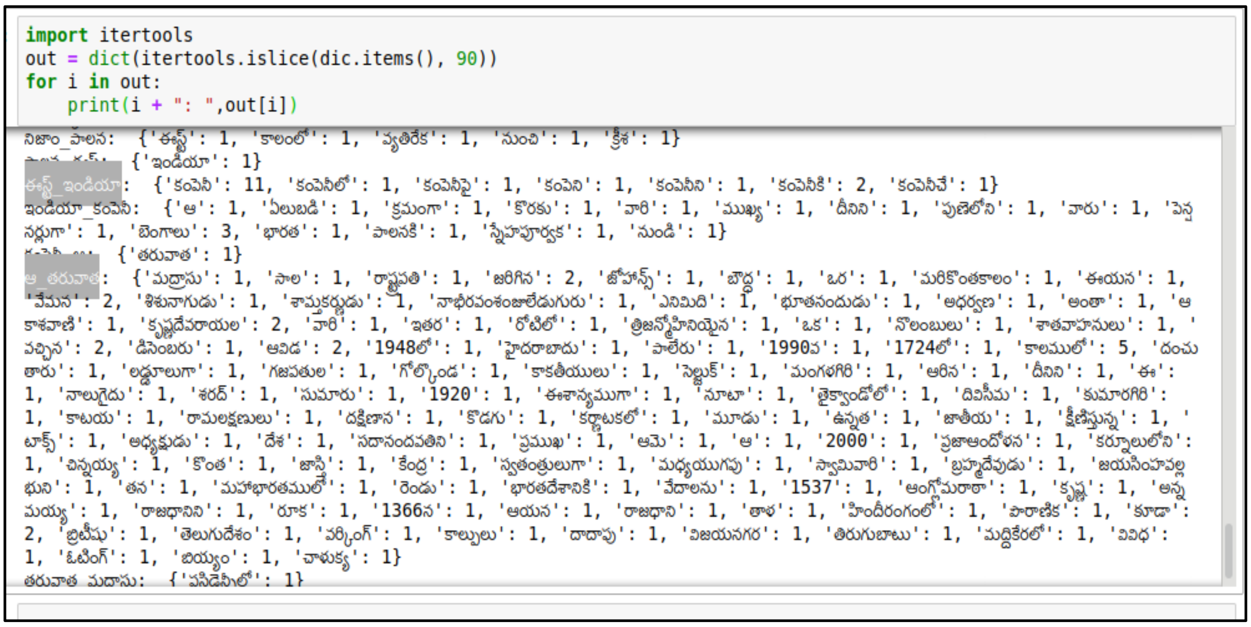


```
import itertools
out = dict(itertools.islice(dic.items(), 30))
for i in out:
    print(i + ": ",out[i])
```

```
ర్పుఉద్దేశించబడింది': 1, 'ఖర్వ': 1, 'తీజ్': 1, 'కాళ్ళకు': 1, 'శాశ్వతగాయాలుతీవ్రమైన': 1, 'సులభమైన': 1, 'మసీదు': 1, 'పరిసరగ్రామాలనుండి': 1, '
ద్వారపూడికి': 1, 'చెల్లూరు': 1, 'తుల్యభాగనది': 1, 'గజవాహనం': 1, 'పరుపుల': 1, 'తాపేశ్వరం': 1, 'రామవరం': 1, 'చిన': 1, 'అనపర్తి': 1, '
పారిశుధ్య': 1, 'అమలాపురానికి': 1, 'కొందరు': 1, 'భవనాల': 1, 'వీరభద్ర': 1, 'పాలవరం': 1, 'భీమేశ్వరస్వామి': 1, 'మిగతా': 1, 'బాలికలు': 1,
'విశాఖ': 1, 'అలన్': 1, 'అనుసంధాన': 1, 'భాషా': 1, 'బీజ': 1, 'సాఫ్ట్వేరు': 1, 'చూపులకు': 1, 'అస్పాంల': 1, 'పాళీ': 1, 'ప్రజలను': 1,
'తమిళనాడులలో': 1, 'థర్మోల్యూమినిసెన్స్': 1, 'సింగపూర్': 1, 'బందర్': 1, 'దేవ్': 1, 'రాజకోట్': 1, 'పురాణాలలోని': 1, 'పాలెం': 1, 'ఆటోలో':
1, 'ప్రభత్వ': 1, 'గణన': 1, 'కాయకూరలు': 1, 'రాజీవ్': 1, 'కుంకు': 1, 'తంగెడు': 1, 'సత్తుపిండి': 1, 'నెయ్యి': 1, 'రొట్టెతో': 1, 'ఆడపిల్ల
      'వాళ్ళు': 1, 'రైలుబండి': 1, 'జనగాం': 1, 'కూకట్': 1, 'నవభరతి': 1}
వాయువ్యాన:  {'నల్గొండ': 1, 'అజర్': 1, 'రాజస్థాన్': 1, 'ఒడిషా': 1, 'ఉత్తర': 1, '43': 1, 'దక్షిణ': 1, 'కృష్ణా': 1, 'రంగారెడ్డి': 1, 'తప్ప':
1, 'ఎర్ర': 1}
నల్గొండ:  {'జిల్లా': 193, 'వరంగల్': 3, 'విజయవాడ': 1, ' </s>': 3, 'జిల్లాలో': 21, 'జిల్లాకు': 11, 'జిల్లాల': 3, 'జిల్లాలలో': 3, 'ఖమ్మం':
1, 'ఆగ్నేయ': 1, 'పాలమూరు': 1, 'మెదక్': 3, 'కృష్ణా': 2, 'జిల్లానకిరేకల్': 1, 'మహబూబ్': 1, 'నుండి': 126, 'జిల్లాలోని': 3, 'సూర్యాపేట':
1, 'లోని': 1, 'ద్రాక్షారామము': 2, 'లోనూ': 87, 'పానుగల్లు': 1, 'పట్టణంలో': 1, 'జిల్లాల్లో': 1, 'జిల్లాలతో': 1, 'శాసనం': 1, 'లోకసభ': 1, '
జిల్లాచింతపల్లి': 1, '2014జనవరి20': 1, 'వెళ్ళు': 4, 'జిల్లానార్కెట్': 1, 'వరం': 1, 'శాసనాలు': 1, 'మండలంలోని': 1, 'జిల్లాకి': 1, 'నియోజకవర్గ
        'ెలవడం': 1}
సరిహద్దులుగా:  {'ఉన్నాయి': 25, 'ఉన్నాయిఖమ్మం': 1, 'ఉన్నాయికొండప్రాంతం': 1, 'ఉత్తరాన': 2, 'కలిగి': 1, 'ఉన్నందున': 1, 'ఆఫ్ఘనిస్తాన్': 1, 'ఉన్నా
యిహసన్': 1, 'ఉన్నవి': 1, 'వాయలూరు': 1, 'యడ్లపాడు': 1, 'కళ్ళచెరువు': 1}
ఉన్నాయి:  {' </s>': 158209, 'అవితెనాలి': 1, 'వాతావరణ': 1, 'కనుక': 5, 'ప్రమేహం': 1, 'కానీ': 7, 'వీటిలో': 3, 'ఎక్కువగా': 1, 'కొన్ని':
1, 'కోసం': 1, 'అందులో': 2, 'హెక్టార్లలో': 472, 'కదా': 2, '–': 1, 'ఇచ్ఛాపురం': 1, 'మహబూబ్': 1, 'గోడలన్నీ': 1, 'నుండి': 1, 'ఎయి
ర్': 1, 'చైనా': 1, 'విస్తరించింది': 1, 'స్వాధీనం': 1, 'ఉజ్బెకిస్థాన్': 1, 'పన్నులు': 1, '1872లో': 1, 'కాని': 2, 'ముంబాయి': 1, '4': 1, 'ల
```

Fig 4.4.1

4.4.2 Tri-Gram



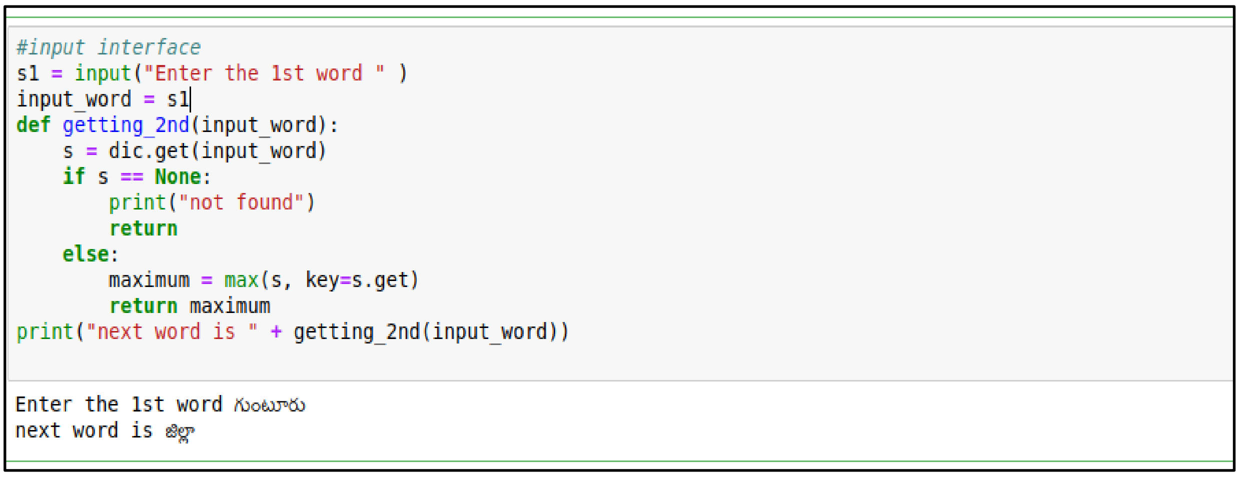
```
import itertools
out = dict(itertools.islice(dic.items(), 90))
for i in out:
    print(i + ": ",out[i])
```

```
నిజాం_పాలన:  {'ఈస్ట్': 1, 'కాలంలో': 1, 'వ్యతిరేక': 1, 'నుంచి': 1, 'క్రీశ': 1}
           {'ఇండియా': 1}
ఈస్ట్_ఇండియా:  {'కంపెనీ': 11, 'కంపెనీలో': 1, 'కంపెనీపై': 1, 'కంపెని': 1, 'కంపెనీని': 1, 'కంపెనీకి': 2, 'కంపెనీచే': 1}
ఇండియా_కంపెనీ:  {'ఆ': 1, 'ఏలుబడి': 1, 'క్రమంగా': 1, 'కొరకు': 1, 'వారి': 1, 'ముఖ్య': 1, 'దీనిని': 1, 'పుణెలోని': 1, 'వారు': 1, 'పెన్న
నర్లుగా': 1, 'బెంగాలు': 3, 'భారత': 1, 'పాలనకి': 1, 'స్నేహపూర్వక': 1, 'నుండి': 1}
          {'తరువాత': 1}
ఆ_తరువాత:  {'మద్రాసు': 1, 'సాల': 1, 'రాష్ట్రపతి': 1, 'జరిగిన': 2, 'జోహాన్స్': 1, 'బౌద్ధ': 1, 'ఒర': 1, 'మరికొంతకాలం': 1, 'ఈయన': 1,
'వేమన': 2, 'శిశునాగుడు': 1, 'శాత్తకర్ణుడు': 1, 'నాభీరవంశంజాలేడుగురు': 1, 'ఎనిమిది': 1, 'భూతనందుడు': 1, 'అధర్వణ': 1, 'అంతా': 1, 'ఆ
కాశవాణి': 1, 'కృష్ణదేవరాయల': 2, 'వారి': 1, 'ఇతర': 1, 'రోటిలో': 1, 'త్రిజన్మోహినియైన': 1, 'ఒక': 1, 'నొలంబులు': 1, 'శాతవాహనులు': 1, '
వచ్చిన': 2, 'డిసెంబరు': 1, 'ఆవిడ': 2, '1948లో': 1, 'హైదరాబాదు': 1, 'పాలేరు': 1, '1990వ': 1, '1724లో': 1, 'కాలములో': 5, 'దంచు
తారు': 1, 'లడ్డూలుగా': 1, 'గజపతుల': 1, 'గోల్కొండ': 1, 'కాకతీయులు': 1, 'సెల్జుక్': 1, 'మంగళగిరి': 1, 'ఆరిన': 1, 'దీనిని': 1, 'ఈ':
1, 'నాలుగైదు': 1, 'శరద్': 1, 'సుమారు': 1, '1920': 1, 'ఈశాన్యముగా': 1, 'నూటా': 1, 'తైక్వాండోలో': 1, 'దివిసీమ': 1, 'కుమారగిరి':
1, 'కాటయ': 1, 'రామలక్షణులు': 1, 'దక్షిణాన': 1, 'కొడగు': 1, 'కర్ణాటకలో': 1, 'మూడు': 1, 'ఉన్నత': 1, 'జాతీయ': 1, 'క్షీణిస్తున్న': 1, '
టాక్స్': 1, 'అధ్యక్షుడు': 1, 'దేశ': 1, 'సదానందవతిని': 1, 'ప్రముఖ': 1, 'ఆమె': 1, 'ఆ': 1, '2000': 1, 'ప్రజాఆందోళన': 1, 'కర్నూలులోని':
1, 'చిన్నయ్య': 1, 'కొంత': 1, 'జాస్తి': 1, 'కేంద్ర': 1, 'స్వతంత్రులుగా': 1, 'మధ్యయుగపు': 1, 'స్వామివారి': 1, 'బ్రహ్మదేవుడు': 1, 'జయసింహవల్ల
భుని': 1, 'తన': 1, 'మహాభారతములో': 1, 'రెండు': 1, 'భారతదేశానికి': 1, 'వేదాలను': 1, '1537': 1, 'ఆంగ్లోమరాఠా': 1, 'కృష్ణ': 1, 'అన్న
మయ్య': 1, 'రాజధానిని': 1, 'రూక': 1, '1366న': 1, 'ఆయన': 1, 'రాజధాని': 1, 'తాళ': 1, 'హిందీరంగంలో': 1, 'పౌరాణిక': 1, 'కూడా':
2, 'బ్రిటీషు': 1, 'తెలుగుదేశం': 1, 'వర్కింగ్': 1, 'కాల్వులు': 1, 'దాదాపు': 1, 'విజయనగర': 1, 'తిరుగుబాటు': 1, 'మద్దికేరలో': 1, 'వివిధ':
1, 'ఓటింగ్': 1, 'బియ్యం': 1, 'చాళుక్య': 1}
తరువాత_మద్రాసు:  {'పసిడెసీలో': 1}
```

Fig 4.4.2

4.5 INPUT-OUTPUT SCREENSHOT

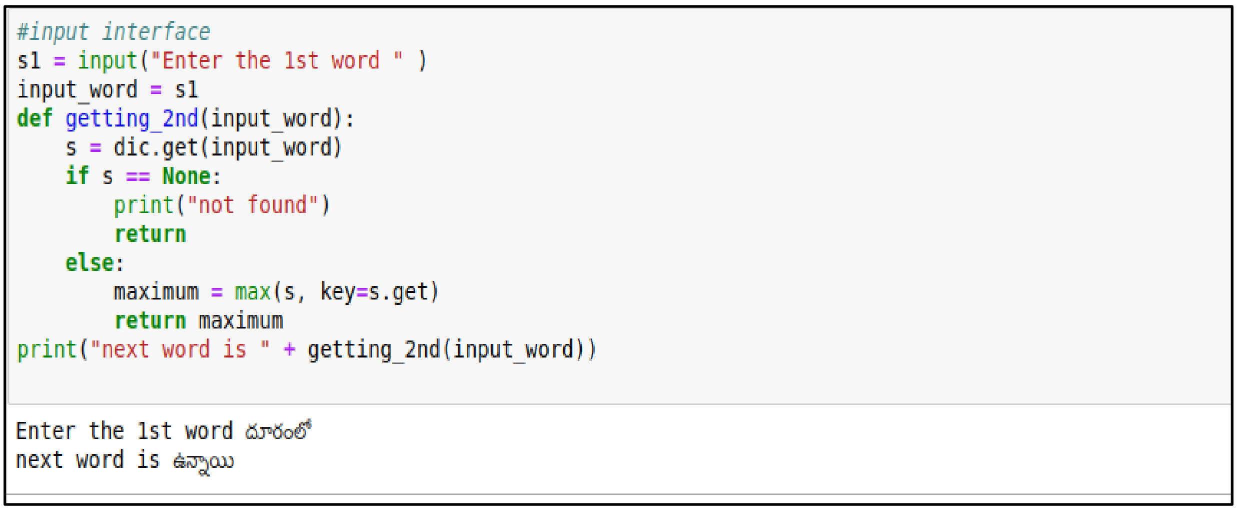
4.5.1 Bi-Gram



```
#input interface
s1 = input("Enter the 1st word " )
input_word = s1
def getting_2nd(input_word):
    s = dic.get(input_word)
    if s == None:
        print("not found")
        return
    else:
        maximum = max(s, key=s.get)
        return maximum
print("next word is " + getting_2nd(input_word))
```

```
Enter the 1st word గుంటూరు
next word is జిల్లా
```

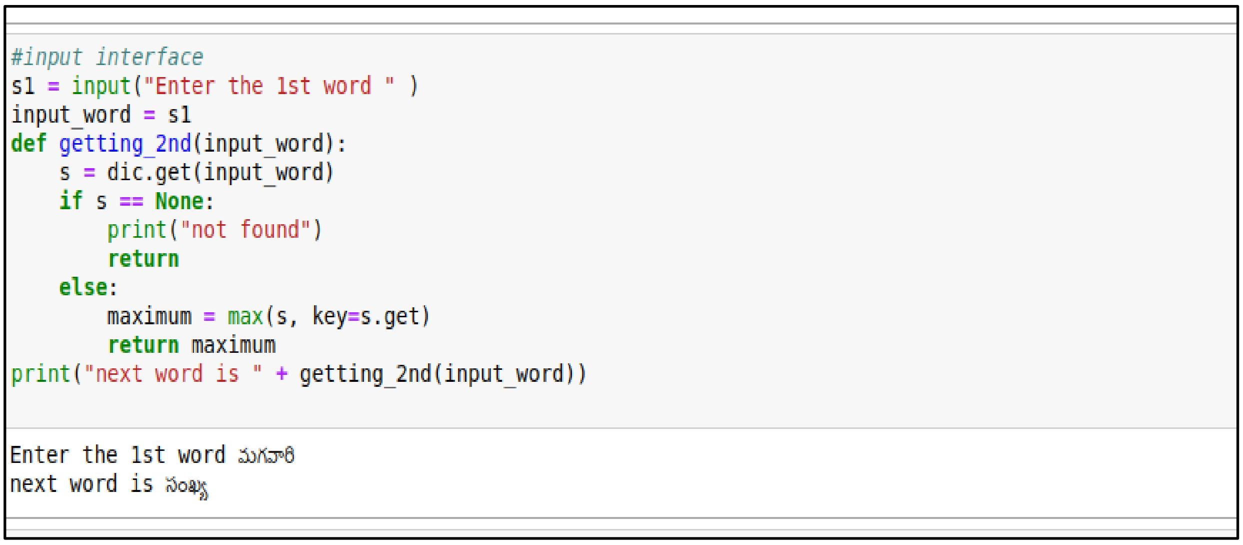
Fig 4.5.1.1



```
#input interface
s1 = input("Enter the 1st word " )
input_word = s1
def getting_2nd(input_word):
    s = dic.get(input_word)
    if s == None:
        print("not found")
        return
    else:
        maximum = max(s, key=s.get)
        return maximum
print("next word is " + getting_2nd(input_word))
```

```
Enter the 1st word దూరంలో
next word is ఉన్నాయి
```

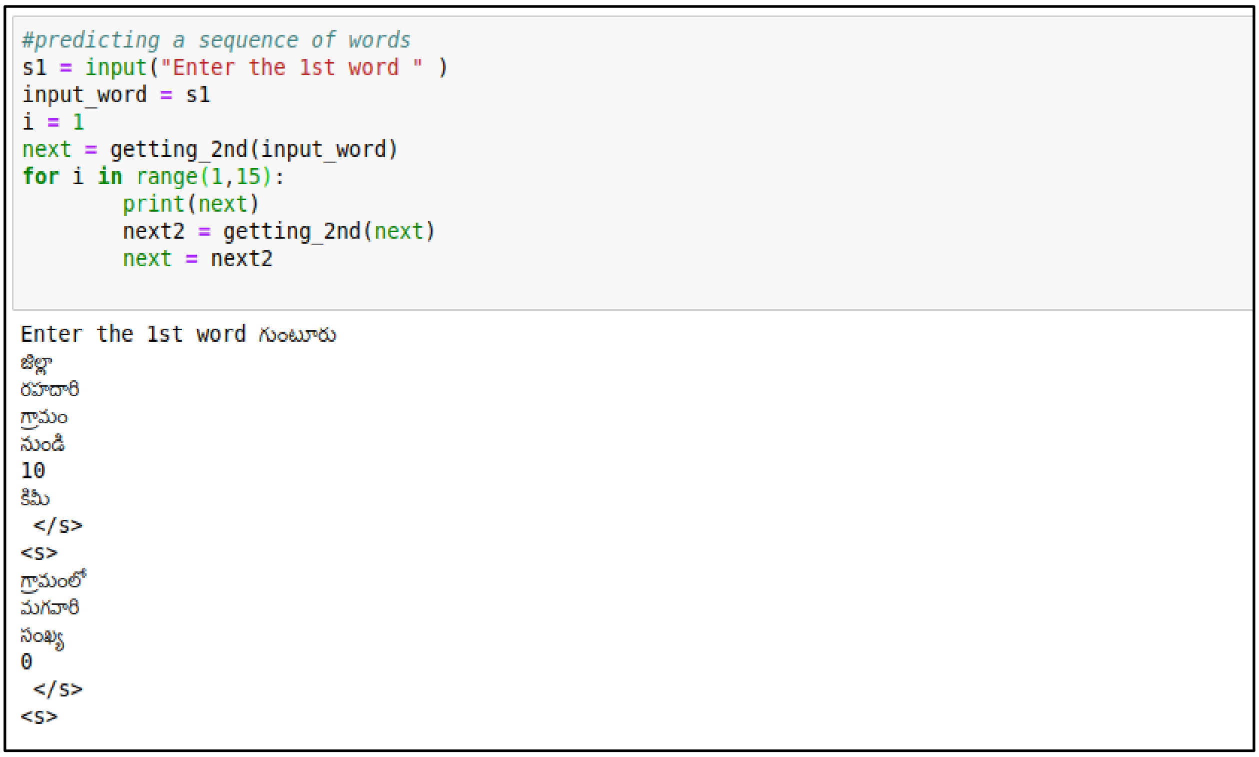
Fig 4.5.1.2



```
#input interface
s1 = input("Enter the 1st word " )
input_word = s1
def getting_2nd(input_word):
    s = dic.get(input_word)
    if s == None:
        print("not found")
        return
    else:
        maximum = max(s, key=s.get)
        return maximum
print("next word is " + getting_2nd(input_word))
```

```
Enter the 1st word మగవారి
next word is సంఖ్య
```

Fig 4.5.1.3

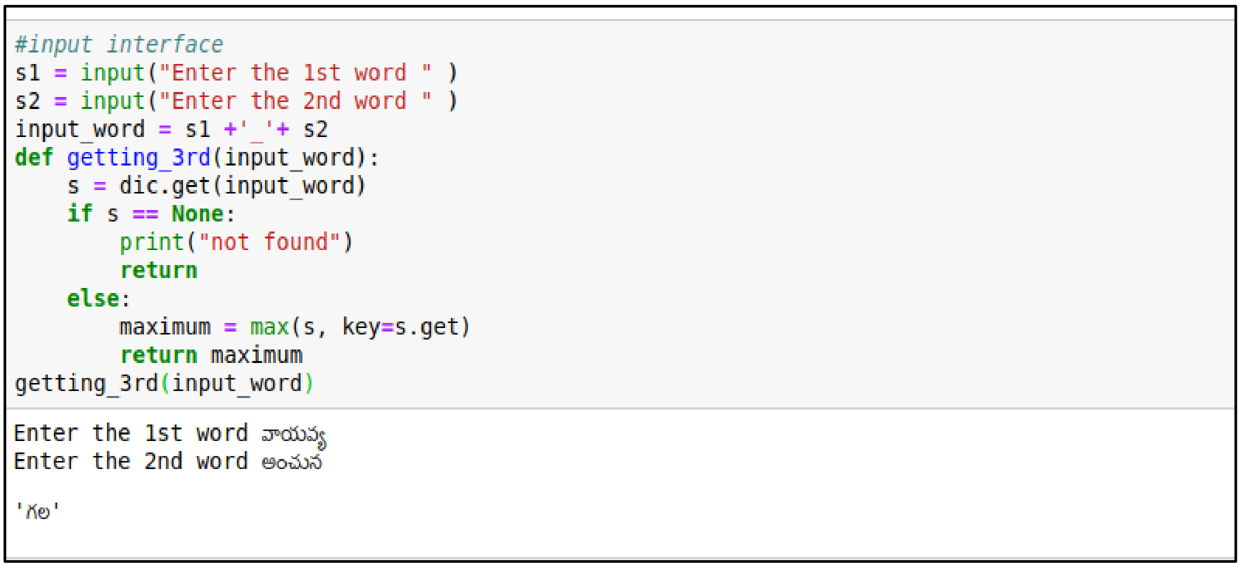


```
#predicting a sequence of words
s1 = input("Enter the 1st word " )
input_word = s1
i = 1
next = getting_2nd(input_word)
for i in range(1,15):
        print(next)
        next2 = getting_2nd(next)
        next = next2
```

```
Enter the 1st word గుంటూరు
జిల్లా
రహదారి
గ్రామం
నుండి
10
కిమీ
 </s>
<s>
గ్రామంలో
మగవారి
సంఖ్య
0
 </s>
<s>
```

Fig 4.5.1.4

4.5.2 Tri-Gram

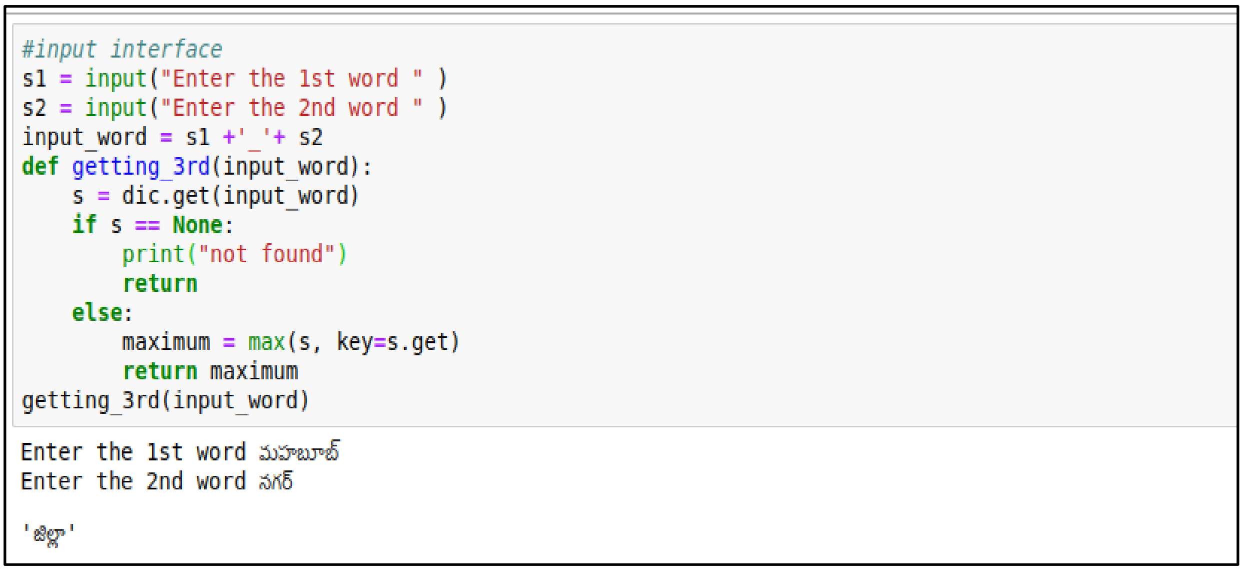


```
#input interface
s1 = input("Enter the 1st word " )
s2 = input("Enter the 2nd word " )
input_word = s1 +'_'+ s2
def getting_3rd(input_word):
    s = dic.get(input_word)
    if s == None:
        print("not found")
        return
    else:
        maximum = max(s, key=s.get)
        return maximum
getting_3rd(input_word)
```

```
Enter the 1st word వాయువ్య
Enter the 2nd word అంచున

'గల'
```

Fig 4.5.2.1

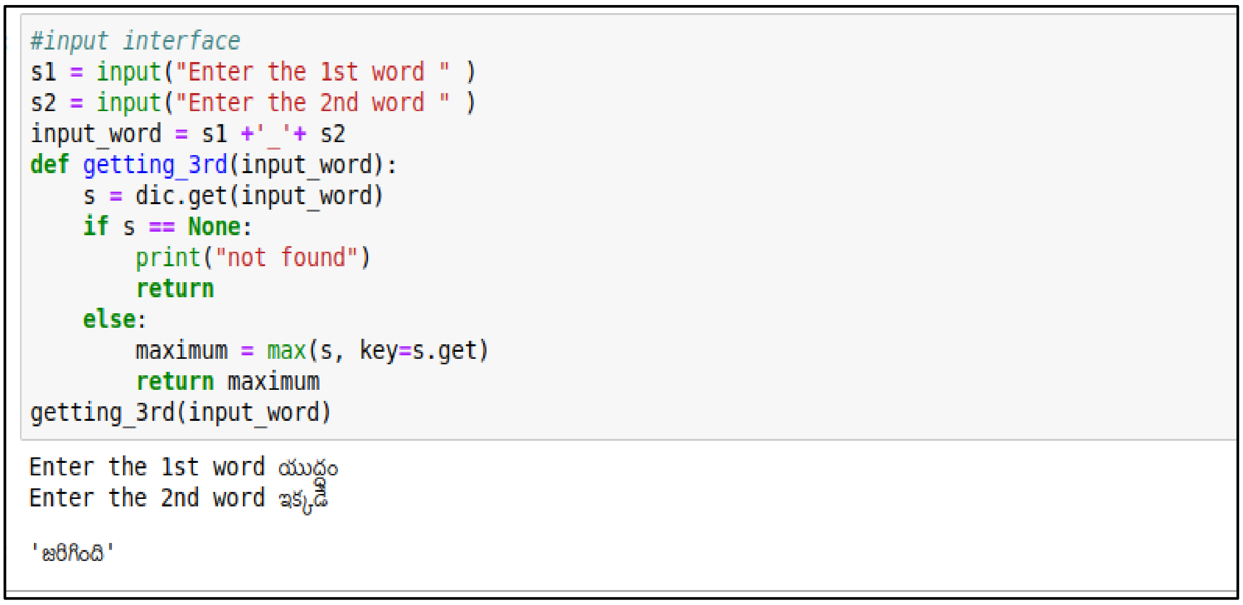


```
#input interface
s1 = input("Enter the 1st word " )
s2 = input("Enter the 2nd word " )
input_word = s1 +'_'+ s2
def getting_3rd(input_word):
    s = dic.get(input_word)
    if s == None:
        print("not found")
        return
    else:
        maximum = max(s, key=s.get)
        return maximum
getting_3rd(input_word)
```

```
Enter the 1st word మహబూబ్
Enter the 2nd word నగర్

'జిల్లా'
```

Fig 4.5.2.2

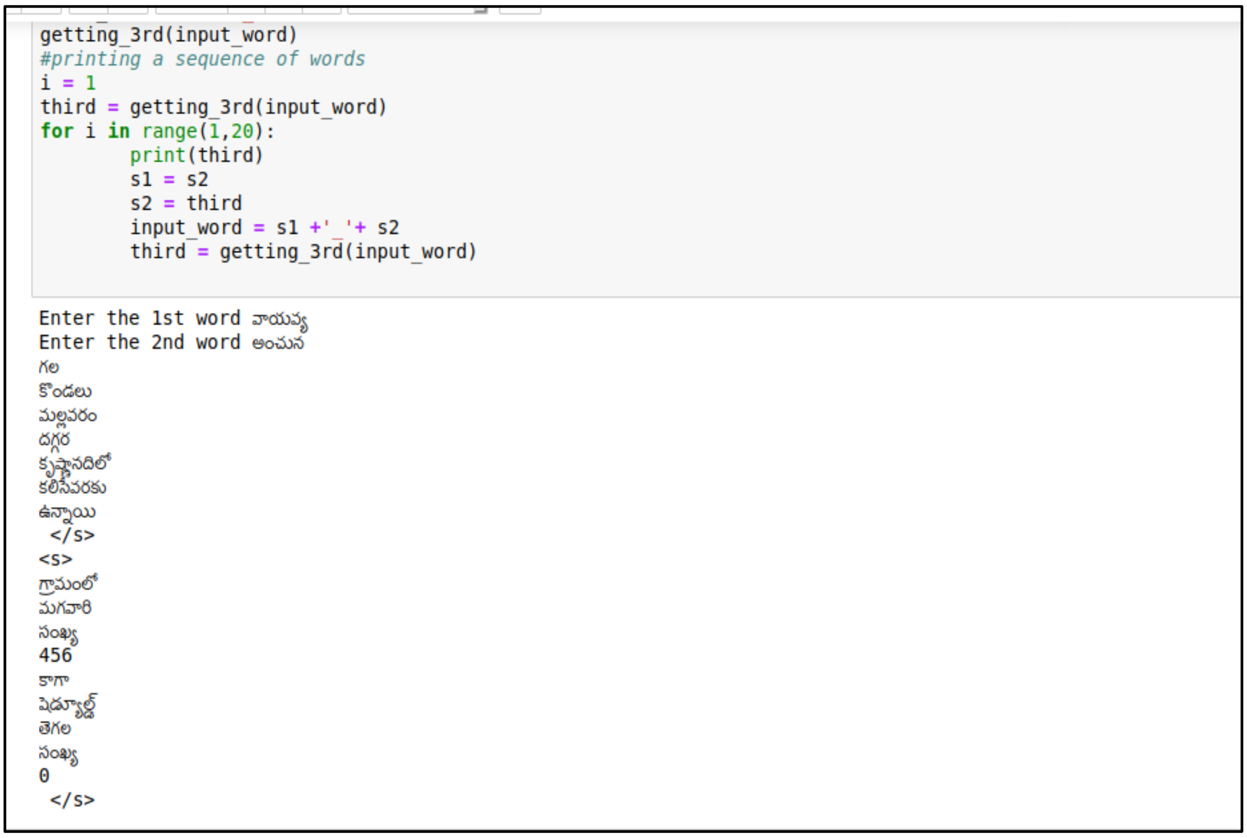


```
#input interface
s1 = input("Enter the 1st word " )
s2 = input("Enter the 2nd word " )
input_word = s1 +'_'+ s2
def getting_3rd(input_word):
    s = dic.get(input_word)
    if s == None:
        print("not found")
        return
    else:
        maximum = max(s, key=s.get)
        return maximum
getting_3rd(input_word)
```

```
Enter the 1st word యుద్ధం
Enter the 2nd word ఇక్కడే

'జరిగింది'
```

Fig 4.5.2.3



```
getting_3rd(input_word)
#printing a sequence of words
i = 1
third = getting_3rd(input_word)
for i in range(1,20):
        print(third)
        s1 = s2
        s2 = third
        input_word = s1 +'_'+ s2
        third = getting_3rd(input_word)
```

```
Enter the 1st word వాయువ్య
Enter the 2nd word అంచున
గల
కొండలు
మల్లవరం
దగ్గర
కృష్ణానదిలో
కలిసేవరకు
ఉన్నాయి
 </s>
<s>
గ్రామంలో
మగవారి
సంఖ్య
456
కాగా
షెడ్యూల్డ్
తెగల
సంఖ్య
0
 </s>
```

Fig 4.5.2.4

5. CONCLUSION

We made an attempt to develop Statistical Telugu Language Mode. It accepts previous n Telugu words as input and predicts best fit next Telugu word. Top of that to handle unknown words in test data add-one and add-alpha smoothing techniques are applied. We reported results for various n-gram models. Results on such small data are reasonable. Results can be further improved if the size of the corpus is larger and used other good smoothing techniques.

6. FUTURE SCOPE

The future scope is add Neural Telugu Language Model, Bidirectional LSTM Language Model and use Beam Search instead of greedy search.

7. REFERENCES

1. Clarkson, Philip, and Ronald Rosenfeld. "Statistical language modeling using the CMU-Cambridge toolkit." *Fifth European Conference on Speech Communication and Technology*. 1997.

2. Brants, Thorsten, et al. "Large language models in machine translation." *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2007.

3. Rani, B. Padmaja, et al. "Analysis of N-gram model on Telugu document classification." *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008.

4. RamaSree, R. J., and P. Kusuma Kumari. "Combining pos taggers for improved accuracy to create telugu annotated texts for information retrieval." *Dept. of Telugu Studies, Tirupathi, India* (2007).