BANKING BOT USING NLP

A MINI PROJECT REPORT

Submitted by

ASHIKA S (720420104008)

SOPARNO S (720420104033)

SREYA PP (720420104034)

UMADEVI V (720420104322)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

CMS COLLEGE OF ENGINEERING AND TECHNOLOGY
COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"BANKING BOT USING NLP"** is the bonafide work of "**ASHIKA S, SOPARNO S, SREYA PP, UMADEVI V**" who carried out the projectwork under my supervision.

SIGNATURE	**SIGNATURE**
Dr.G.CHITRA GANAPATHY,	**Mr.M.JAYAPRAKASH,**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor ,	**Professor ,**
Department of Computer Science & Engineering,	Department of Electronics & Communication Engineering,
CMS College of Engineering and Technology,	CMS College of Engineering and Technology,
Coimbatore-641 032	Coimbatore-641 032

Submitted for the Anna University Project Viva voce held on……..

INTERNAL EXAMINER **EXTERNAL EXAMINAR**

ACKNOWLEDGEMENT

Any organized and systematic word calls for the corporation of team of people. Our project does not have any exception to this. Hence these pages find the space for thanking all those who have directly and indirectly contributed to completion of this work in a successful manner.

We record our indebtedness to our **Principal Dr.N.Sudha, M.E, Ph.D** for her support and sustained encouragementfor the successful completion of this project.

We express our heartiest thanks to **Dr.G.Chitra Ganapathy,M.E,Ph.D, Professor, Head Of The Department**, Department of Computer Science and Engineering ,CMS College of Engineering and Technology, for her encouragement and valuable guidance in carrying out our project work.

We express our heartfelt thanks to Project Guide **Mr.M.Jayaprakash, M.E., Ph.D., Professor**, Department of Electronics and Communication Engineering, CMS College of Engineering and Technology, Coimbatore, for his guidance and timely support for our project work.

We express our heartfelt thanks to Project Co-ordinator **Mr.S.Dinesh Kumar, M.E.**, **Assistant Professor**, Department of Computer Science and Engineering, CMS College of Engineering and Technology, Coimbatore, for his valuable and timely support for our project work.

We also express thanks to our parents, friends for their encouragement and best wishes in the successful completion of this dissertation.

ABSTRACT

A banking bot using Natural Language Processing (NLP) is an application that allows customers to interact with their bank using natural language queries. The primary objective of the bot is to enhance the customer experience by providing a convenient and efficient way to perform various banking tasks, such as getting information about products and services.

The bot uses NLP technology to interpret and understand the meaning behind customer queries, which enables it to provide helpful responses. By automating many routine customer service tasks, the banking bot can help banks reduce costs and improve operational efficiency. This, in turn, allows bank staff to focus on more complex issues that require human attention.

Overall, the objective of a banking bot using NLP is to provide customers with a personalized and seamless banking experience while reducing operational costs for banks. By leveraging the power of NLP, banks can enhance their customer service, improve efficiency, and stay ahead of the competition in the digital age.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGENO
	ACKNOWLEDGEMENT	iii
	ABSTRACT	iv
	LIST OF FIGURES	
1.	**INTRODUCTION**	**2**
2.	**LITERATURE SURVEY**	**3**
3.	**SYSTEM ANALYSIS**	**5**
	3.1 EXISTING SYSTEM	
	3.2 DRAWBACKS OF EXISTING SYSTEM	
	3.3 PROPOSED SYSTEM	
	3.4ADVANTAGES OF PROPOSED SYSTEM	
4.	**SYSTEM REQUIREMENTS**	**8**
	4.1 HARDWARE REQUIREMENTS	
	4.2 SOFTWARE REQUIREMENTS	
5.	**SOFTWARE DESCRIPTION**	**9**
	5.1 PYTHON	
	5.2 NATURAL LANGUAGE PROCESSING	
	5.3 SKLEARN TfidfVectorizer	

5.4 SKLEARN COSINE_SIMILARITY

6. MODULES 36

6.1 GATHER DATA AND USE CASES

6.2 PREPROCESS AND LABEL DATA

6.3 CHOOSE AND TRAIN NPL MODEL

6.4 IMPLEMENT THE BOT LOGIC

6.5 BUILD USER INTERFACE

7. SYSTEM ANALYSIS 39

7.1 FEASIBILITY STUDY

7.2 OPERATIONAL FEASIBILITY

7.3 ECONOMIC FEASIBILITY

8. SOFTWARE TESING 41

8.1 SYSTEM TESTING

8.2 UNIT TESTING

8.3 BLACK BOX TESTING

8.4 WHITE BOX TESTING

8.5 INTEGRATION TESTING

8.6 VALIDATION TESTING

9. CONCLUSION 44

9.1 CONCLUSION

9.2 FUTURE SCOPE

10. APPENDICES 46

10.1 SOURCE CODE

10.2 SCREENSHOTS

11. REFERENCE 56

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
Fig 6.1.1	Database Diagram	37
Fig 6.4.1	System Architecture	39

CHAPTER 1
INTRODUCTION

In recent years, banks have increasingly turned to technology to improve their customer service and operational efficiency. One area where technology has shown great promise is in the development of banking bots that use Natural Language Processing (NLP) to interact with customers.

A banking bot using NLP is an application that enables customers to communicate with their bank using natural language queries. The bot uses NLP technology to understand the meaning behind customer queries and provide helpful responses. This can range from checking account balances to transferring funds and paying bills, all without the need to navigate complex menus or search for information on the bank's website.

The primary objective of a banking bot using NLP is to enhance the customer experience by providing a fast, convenient, and personalized way to interact with the bank. By automating routine customer service tasks, the bot can also help banks reduce costs and improve operational efficiency.

This paper aims to explore the objective and benefits of a banking bot using NLP. We will delve into how NLP technology works and how it can be applied to improve the customer experience in banking. We will also examine the challenges of implementing a banking bot using NLP and potential solutions to overcome them.

Overall, a banking bot using NLP is a powerful tool for banks to enhance their customer service and operational efficiency, providing customers with a more convenient and personalized banking experience while also reducing costs for the bank.

CHAPTER 2
LITERATURE SURVEY

The banking Industry has been one of the earliest adopters of chatbots in its operations. Chatbots are sophisticated computer programs designed to interact with customers in a similar manner as humans. As a result they help banks to enhance customer satisfaction by answering a host of their queries within seconds. State Bank of India (SBI) is the largest public sector banking services provider in the country. To deliver effective banking services, the bank capitalizes on artificial intelligence.

SBI Intelligent Assistant (SIA), an AI-powered smart chat assistant, addresses customer enquiries instantly and helps them with everyday banking tasks like a human does. Developed by an AI banking platform Payjo, this smart chat assistant is equipped to handle nearly 10,000 enquiries per second or 864 million in a day, which is almost 25% of the queries are processed by Google each day.

S. Sarbabidya, T.Saha, (2020) explains that chatbot is an emerging, smart and immediately adoptable technology that shows it is very beneficial in rendering various customer services such as one-to-one conversations, responsive customer service through 24/7/365 availability, which is supported by M. Adam, M. Wessel, Benlian, (2020) says that chatbot is the system designed to communicate with human users by means of natural language often based on artificial intelligence. , Nguyen, Tung, (2019) had analysed an experiment at a case company for using chatbots for customer support which was utilized to evaluate the potential effects of chatbots on the operation of support for customers. Another author named E. Ojapuska,(2018) explains about the impact that chatbot has on customer engagement and also how chatbots differ from other communication channels that are also used to increase customer engagement.

Gupta. A, Sharma. D, (2018) explains all about the customer's attitude towards chatbots in the banking industry of india including TAM model. Motivations pertaining to entertainment, social and relational factors, L.N.Michanel, 2018 shares insights that are designed and implemented for a SMS Chatbot- based virtual assistant to hotel guests. Also, Petter Bae Brandtzaeg and Asbjorn Folstad, 2017 explains that why people use chatbots which had shown it has helped users to obtain timely and efficient assistance.

Through these research papers it can be concluded that customers need the full support in order to enjoy the services fully. It has been seen that it is not just respond but self- learn to improve itself thereby increasing not just the quality of customer service but also reducing the human load. Since, AI- based customer acquisition have increasingly popular in various settings and potentially offer a number of time and also the cost- saving opportunity. Thus a great idea to increase the use of chatbots. Also, it helps in improving consistency.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

These are third-party platforms that provide tools for building and deploying chatbots. Some popular chatbot platforms for banking include IBM Watson Assistant, Dialogflow, and Botpress. These platforms allow banks to create chatbots using pre-built templates, natural language processing (NLP), and machine learning (ML) capabilities.

Custom-built bots: Banks can also build their own custom chatbots using programming languages like Python, Java, or JavaScript. This approach requires more technical expertise, but it allows banks to create a chatbot tailored to their specific needs.

3.2 DISADVANTAGES OF EXISTING SYSTEM:

The existing banking system, while essential for financial services, is not without its disadvantages. Here are a few drawbacks associated with the traditional banking system:

1. Limited Accessibility: Physical bank branches have limited operating hours, and customers must visit them in person to access services. This can be inconvenient for individuals who work during banking hours or live in remote areas with limited branch coverage. Accessibility barriers can hinder certain individuals from accessing banking services.

2. Time-consuming Processes: Many banking processes in the traditional system can be time-consuming. Opening an account, applying for loans, or conducting complex transactions often involve extensive paperwork, manual verifications, and multiple visits to the bank. These processes can be inefficient and lead to delays and frustrations for customers.

3. Lack of Transparency: The traditional banking system can sometimes be opaque in terms of fees, interest rates, and terms and conditions. Customers may find it challenging to understand the true cost of services or accurately compare offerings from different banks. This lack of transparency can lead to confusion and dissatisfaction among customers.

3.3 PROPOSED SYSTEM:

User Interface: The user interface of the banking bot should be easy to use, intuitive, and user-friendly. Customers should be able to interact with the bot using natural language and receive responses that are accurate and relevant to their queries.

Natural Language Processing (NLP): The banking bot should have advanced NLP capabilities that allow it to understand and interpret the meaning behind customer queries. This would enable the bot to provide accurate and relevant responses, and even suggest additional services or products that the customer may be interested in.

Integration with banking systems: The banking bot should be integrated with the bank's core banking systems, such as customer databases, account information, and transaction history. This would allow the bot to provide personalized information to customers based on their individual account information.

3.4 ADVANTAGES OF PROPOSED SYSTEM:

Banking bots, also known as chatbots or virtual assistants in the banking industry, offer several advantages that enhance the customer experience and streamline banking operations. Here are some key advantages of banking bots:

1. 24/7 Availability: Banking bots are available round the clock, allowing customers to access banking services and support at any time. This eliminates the limitations of traditional banking hours and provides convenience to customers who may have queries or require assistance outside of regular business hours.

2. Instant Responses: Banking bots can provide instant responses to customer queries, reducing waiting times and improving response times compared to traditional customer service channels. Customers can receive immediate assistance for common banking tasks, such as balance inquiries, transaction history, or account-related information.

3. Cost Savings: Implementing banking bots can result in cost savings for banks. Automated bots can handle a high volume of customer inquiries simultaneously, reducing the need for a large customer support team. This can lead to significant cost reductions associated with staffing, training, and operational expenses.

It's important to note that while banking bots offer numerous advantages, they are not intended to replace human agents entirely. A well-designed bot-human hybrid model can leverage the strengths of both automated systems and human assistance to deliver the best possible customer experience.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS:

Processor : Pentium IV(minimum)

Hard Disk : 40GB

RAM : 256MB (minimum)

4.2 SOFTWARE REQUIREMENTS:

Operating System : Windows or Linux

Technology : PYTHON, NLP

CHAPTER 5
SOFTWARE DESCRIPTION

5.1 PYTHON

Dating from 1991, Python is a relatively new programming language. From the start, Python was considered a gap-filler, a way to write scripts that "automate the boring stuff" (as one popular book on learning Python put it) or to rapidly prototype applications that will be implemented in one or more other languages. However, over the past few years, Python has emerged as a first-class citizen in modern software development, infrastructure management, and data analysis. It is no longer a backroom utility language, but a major force in web application development and systems management and a key driver behind the explosion in big data analytics and machine Perfect for IT, Python simplifies many kinds of work, from system automation to working in cutting-edge fields like machine learning. Python is easy to learn. The number of features in the language itself is modest, requiring relatively little investment of time or effort to produce one's first programs. Python syntax is designed to be readable and straightforward. This simplicity makes Python an ideal teaching language, and allows newcomers to pick it up quickly. Developers spend more time thinking about the problem they're trying to solve, and less time thinking about language complexities or deciphering code left by others. Python is broadly used and supported. Python is both popular and widely used, as the high rankings in surveys like the Tiobe Index and the large number of GitHub projects using Python attest. Python runs on every major operating system and platform, and most minor ones too. Many major libraries and API-powered services have Python bindings or wrappers, allowing Python to interface freely with those services or make direct use of those libraries. Python may not be the fastest language, but what it lacks in speed, it

makes up for in versatility. Python is not a “toy” language. Even though scripting and automation cover a large chunk of Python’s use cases (more on that below), Python is also used to build robust, professional- quality software, both as standalone applications and as web services. What is Python used for? The most basic use case for Python is as a scripting and automation language. Python isn’t just a replacement for shell scripts or batch files, but is also used to automate interactions with web browsers or application GUIs or system provisioning and configuration in tools such as Ansible and Salt. But scripting and automation represent only the tip of the iceberg with Python. Python is used for general application programming. Both CLI and cross-platform GUI applications can be created with Python and deployed as self-contained executables. Python doesn’t have the native ability to generate a standalone binary from a script, but third-party packages like cx_Freeze or PyInstaller can be used to accomplish that. Python is used for data science and machine learning. Sophisticated data analysis has become one of fastest moving areas of IT and one of Python’s star use cases. The vast majority of the libraries used for data science or machine learning have Python interfaces, making the language the most popular high-level command interface to for machine learning libraries and other numerical algorithms. Python is used for web services and RESTful APIs. Python’s native libraries and third-party web frameworks provide fast and convenient ways to create everything from simple REST APIs in a few lines of code, to full-blown, data-driven sites. Python’s latest versions have powerful support for asynchronous operations, allowing sites to handle up to tens of thousands of requests per second with the right libraries. Python is used for metaprogramming. In Python, everything in the language is an object, including Python modules and libraries themselves. This allows Python to work as a highly efficient code generator, making it possible to write applications that manipulate their own functions and have the kind of extensibility that would be difficult or impossible to pull off in other languages. Python is used for glue code. Python is often described as a “glue language,”

meaning it can allow disparate code (typically libraries with C language interfaces) to interoperate. Its use in data science and machine learning is in this vein, but that's just one incarnation of the general idea. Also worth noting are the sorts of tasks Python is not well-suited for. Python is a high-level language, so it's not suitable for system-level programming—device drivers or OS kernels are straight out. It's also not ideal for situations that call for cross-platform standalone binaries. You could build a standalone Python app for Windows, Mac, and Linux, but not elegantly or simply. Finally, Python is not the best choice when speed is an absolute priority in every aspect of the application. For that you're better off with C/C++ or another language of that caliber. The Python language's pros and cons: Python syntax is meant to be readable and clean, with little pretense. A standard "hello world" in Python 3.x is nothing more than:

```
print("Hello world!")
```

Python provides many syntactical elements that make it possible to concisely express any common program flows. Consider a sample program for reading lines from a text file into a list object, stripping each line of its terminating newline character along the way:

```
with open('myfile.txt') as my_file:
file_lines = [x.strip('\n') for x in my_file]
```

The with/as construction is a "context manager," which provides an efficient way to instantiate a given object for a block of code and then dispose of it outside of that block. In this case, the object in question is my_file, instantiated with the open() function. This takes the place of several lines of boilerplate to open the file, read individual lines from it, then close it up. The [x … for x in my_file] construction is another Python idiosyncrasy, the "list comprehension." It allows a given item that contains other items (here, my_file and the lines it contains) to be iterated through, and to allow each iterated element (that is, each x) to be

processed and automatically appended into a list. You could write such a thing as a formal for… loop in Python, much as you would in another language. The point is that Python has a way to economically express things like loops that iterate over multiple objects and perform some simple operation on each element in the loop, or work with things that require explicit instantiation and disposal. Constructions like this allow Python developers to balance terseness and readability.

Python's other language features are meant to complement common use cases. Most modern object types—Unicode strings, for instance—are built directly into the language. Data structures—like lists, dictionaries (i.e., hashmaps), tuples (for storing immutable collections of objects), and sets (for storing collections of unique objects)—are available as standard-issue items. Like C#, Java, and Go, Python has garbage-collected memory management, meaning the programmer doesn't have to implement code to track and release objects. Normally garbage collection happens automatically in the background, but if that poses a performance problem, it can be triggered manually or disabled entirely. An important aspect of Python is its dynamism. Everything in the language, including functions and modules themselves, are handled as objects. This comes at the expenseof speed (more on that below), but makes it far easier to write high-level code. Developers can perform complex object manipulations with only a few instructions, and even treat parts of an application as abstractions that can be altered if needed.

Python's use of significant whitespace has been cited as both one of Python's best and worst attributes. The indentation on the second line shown above isn't just for readability; it is part of Python's syntax. Python interpreters will reject programs that don't use proper indentation to indicate control flow. Syntactical white space might cause noses to wrinkle, and some people do reject Python out of hand for this reason. But strict indentation rules are far less

obtrusive in practice than they might seem in theory, even with the most minimal of code editors, and the end result is code that is cleaner and more readable. Python 2 versus Python 3 Python is available in two versions, which are different enough to trip up many new users. Python 2.x, the older “legacy” branch, will continue to be supported (i.e. receive official updates) through 2020, and it might even persist unofficially after that. Python 3.x, the current and future incarnation of the language, has many useful andimportant features not found in 2.x, such as better concurrency controls and a more efficient interpreter.

5.2 NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is the study of letting computers understand human languages[3]. Without NLP, human language sentences are just a series of meaningless symbols to computers. Computers don’t recognize the words and don’t understand the grammars. NLP can be regard as a “ translator”, who will translate human languages to computer understandable information. Traditionally, users need to follow well-defined procedures accurately, in order to interact with computers. For example, in Linux systems, all commands must be precise. A single replace of one character or even a space can have significant difference. However, the emergence of NLP is changing the way of interacting. Apple Siri and Microsoft Cortana have made it possible to give command in everyday languages and is changing the way of interacting.\

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.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

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."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been been updated for Python 3 and NLTK 3.

Installing NLTK Data

NLTK comes with many corpora, toy grammars, trained models, etc. A complete list is posted at: https://www.nltk.org/nltk_data/

To install the data, first install NLTK (see https://www.nltk.org/install.html), then use NLTK's data downloader as described below.

Apart from individual data packages, you can download the entire collection (using "all"), or just the data required for the examples and exercises in the book (using "book"), or just the corpora and no grammars or trained models (using "all-corpora").

Interactive installer

For central installation on a multi-user machine, do the following from an administrator account.

Run the Python interpreter and type the commands:

```
>>> import nltk
>>> nltk.download()
```

A new window should open, showing the NLTK Downloader. Click on the File menu and select Change Download Directory. For central installation, set this to C:\nltk_data (Windows), /usr/local/share/nltk_data (Mac), or /usr/share/nltk_data (Unix). Next, select the packages or collections you want to download.

If you did not install the data to one of the above central locations, you will need to set the NLTK_DATA environment variable to specify the location of the data. (On a Windows machine, right click on “My Computer” then select Properties > Advanced > Environment Variables > User Variables > New...)

Test that the data has been installed as follows. (This assumes you downloaded the Brown Corpus):

```
>>> from nltk.corpus import brown
>>> brown.words()
```

['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

Installing via a proxy web server

If your web connection uses a proxy server, you should specify the proxy address as follows. In the case of an authenticating proxy, specify a username and password. If the proxy is set to None then this function will attempt to detect the system proxy.

```
>>> nltk.set_proxy('http://proxy.example.com:3128', ('USERNAME', 'PASSWORD'))
>>> nltk.download()
```

Command line installation

The downloader will search for an existing nltk_data directory to install NLTK data. If one does not exist it will attempt to create one in a central location (when using an administrator account) or otherwise in the user's filespace. If necessary, run the download command from an administrator account, or using sudo. The recommended system location is C:\nltk_data (Windows); /usr/local/share/nltk_data (Mac); and /usr/share/nltk_data (Unix). You can use the -d flag to specify a different location (but if you do this, be sure to set the NLTK_DATA environment variable accordingly).

Run the command python -m nltk.downloader all. To ensure central installation, run the command sudo python -m nltk.downloader -d /usr/local/share/nltk_data all.

Windows: Use the “Run…” option on the Start menu. Windows Vista users need to first turn on this option, using Start -> Properties -> Customize to check the box to activate the “Run…” option.

Test the installation: Check that the user environment and privileges are set correctly by logging in to a user account, starting the Python interpreter, and accessing the Brown Corpus (see the previous section).

5.3 SKLEARN TfidfVectorizer

Parameters:

input{‘filename’, ‘file’, ‘content’}, default=’content’

If 'filename', the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.

If 'file', the sequence items must have a ‘read’ method (file-like object) that is called to fetch the bytes in memory.

If 'content', the input is expected to be a sequence of items that can be of type string or byte.

encodingstr, default=’utf-8’

If bytes or files are given to analyze, this encoding is used to decode.

decode_error{‘strict’, ‘ignore’, ‘replace’}, default=’strict’

Instruction on what to do if a byte sequence is given to analyze that contains characters not of the given encoding. By default, it is ‘strict’, meaning that a UnicodeDecodeError will be raised. Other values are ‘ignore’ and ‘replace’.

strip_accents{‘ascii’, ‘unicode’} or callable, default=None

Remove accents and perform other character normalization during the preprocessing step. ‘ascii’ is a fast method that only works on characters that have a direct ASCII mapping. ‘unicode’ is a slightly slower method that works on any characters. None (default) does nothing.

Both ‘ascii’ and ‘unicode’ use NFKD normalization from unicodedata.normalize.

lowercasebool, default=True

Convert all characters to lowercase before tokenizing.

preprocessorcallable, default=None

Override the preprocessing (string transformation) stage while preserving the tokenizing and n-grams generation steps. Only applies if analyzer is not callable.

tokenizercallable, default=None

Override the string tokenization step while preserving the preprocessing and n-grams generation steps. Only applies if analyzer == 'word'.

analyzer{‘word’, ‘char’, ‘char_wb’} or callable, default=’word’

Whether the feature should be made of word or character n-grams. Option ‘char_wb’ creates character n-grams only from text inside word boundaries; n-grams at the edges of words are padded with space.

If a callable is passed it is used to extract the sequence of features out of the raw, unprocessed input.

Changed in version 0.21: Since v0.21, if input is 'filename' or 'file', the data is first read from the file and then passed to the given callable analyzer.

stop_words{‘english’}, list, default=None

If a string, it is passed to _check_stop_list and the appropriate stop list is returned. ‘english’ is currently the only supported string value. There are several known issues with ‘english’ and you should consider an alternative (see Using stop words).

If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens. Only applies if analyzer == 'word'.

If None, no stop words will be used. In this case, setting max_df to a higher value, such as in the range (0.7, 1.0), can automatically detect and filter stop words based on intra corpus document frequency of terms.

token_patternstr, default=r”(?u)\b\w\w+\b”

Regular expression denoting what constitutes a “token”, only used if analyzer == 'word'. The default regexp selects tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator).

If there is a capturing group in token_pattern then the captured group content, not the entire match, becomes the token. At most one capturing group is permitted.

ngram_rangetuple (min_n, max_n), default=(1, 1)

The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used. For example an ngram_range of (1, 1) means only unigrams, (1, 2) means unigrams and bigrams, and (2, 2) means only bigrams. Only applies if analyzer is not callable.

max_dffloat or int, default=1.0

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float in range [0.0, 1.0], the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

min_dffloat or int, default=1

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float in range of [0.0, 1.0], the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

max_featuresint, default=None

If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus. Otherwise, all features are used.

This parameter is ignored if vocabulary is not None.

vocabularyMapping or iterable, default=None

Either a Mapping (e.g., a dict) where keys are terms and values are indices in the feature matrix, or an iterable over terms. If not given, a vocabulary is determined from the input documents.

binarybool, default=False

If True, all non-zero term counts are set to 1. This does not mean outputs will have only 0/1 values, only that the tf term in tf-idf is binary. (Set idf and normalization to False to get 0/1 outputs).

dtypedtype, default=float64

Type of the matrix returned by fit_transform() or transform().

norm{‘l1’, ‘l2’} or None, default=’l2’

Each output row will have unit norm, either:

‘l2’: Sum of squares of vector elements is 1. The cosine similarity between two vectors is their dot product when l2 norm has been applied.

‘l1’: Sum of absolute values of vector elements is 1. See preprocessing.normalize.

None: No normalization.

use_idfbool, default=True

Enable inverse-document-frequency reweighting. If False, idf(t) = 1.

smooth_idfbool, default=True

Smooth idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. Prevents zero divisions.

sublinear_tfbool, default=False

Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).

Attributes:

vocabulary_dict

A mapping of terms to feature indices.

fixed_vocabulary_bool

True if a fixed vocabulary of term to indices mapping is provided by the user.

idf_array of shape (n_features,)

Inverse document frequency vector, only defined if use_idf=True.

stop_words_set

Terms that were ignored because they either:

occurred in too many documents (max_df)

occurred in too few documents (min_df)

were cut off by feature selection (max_features).

This is only available if no vocabulary was given.

See also

CountVectorizer

Transforms text into a sparse matrix of n-gram counts.

TfidfTransformer

Performs the TF-IDF transformation from a provided matrix of counts.

Notes

The stop_words_ attribute can get large and increase the model size when pickling. This attribute is provided only for introspection and can be safely removed using delattr or set to None before pickling.

Examples

```
>>>
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
```

```
>>> vectorizer = TfidfVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> vectorizer.get_feature_names_out()
array(['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',
       'this'], ...)
>>> print(X.shape)
(4, 9)
```

Methods

build_analyzer()

Return a callable to process input data.

build_preprocessor()

Return a function to preprocess the text before tokenization.

build_tokenizer()

Return a function that splits a string into a sequence of tokens.

decode(doc)

Decode the input into a string of unicode symbols.

fit(raw_documents[, y])

Learn vocabulary and idf from training set.

fit_transform(raw_documents[, y])

Learn vocabulary and idf, return document-term matrix.

get_feature_names_out([input_features])

Get output feature names for transformation.

get_params([deep])

Get parameters for this estimator.

get_stop_words()

Build or fetch the effective stop words list.

inverse_transform(X)

Return terms per document with nonzero entries in X.

set_params(**params)

Set the parameters of this estimator.

transform(raw_documents)

Transform documents to document-term matrix.

build_analyzer()[source]

Return a callable to process input data.

The callable handles preprocessing, tokenization, and n-grams generation.

Returns:

analyzer: callable

A function to handle preprocessing, tokenization and n-grams generation.

build_preprocessor()[source]

Return a function to preprocess the text before tokenization.

Returns:

preprocessor: callable

A function to preprocess the text before tokenization.

build_tokenizer()[source]

Return a function that splits a string into a sequence of tokens.

Returns:

tokenizer: callable

A function to split a string into a sequence of tokens.

decode(doc)[source]

Decode the input into a string of unicode symbols.

The decoding strategy depends on the vectorizer parameters.

Parameters:

docbytes or str

The string to decode.

Returns:

doc: str

A string of unicode symbols.

fit(raw_documents, y=None)[source]

Learn vocabulary and idf from training set.

Parameters:

raw_documentsiterable

An iterable which generates either str, unicode or file objects.

yNone

This parameter is not needed to compute tfidf.

Returns:

selfobject

Fitted vectorizer.

fit_transform(raw_documents, y=None)[source]

Learn vocabulary and idf, return document-term matrix.

This is equivalent to fit followed by transform, but more efficiently implemented.

Parameters:

raw_documentsiterable

An iterable which generates either str, unicode or file objects.

yNone

This parameter is ignored.

Returns:

Xsparse matrix of (n_samples, n_features)

Tf-idf-weighted document-term matrix.

get_feature_names_out(input_features=None)[source]

Get output feature names for transformation.

Parameters:

input_featuresarray-like of str or None, default=None

Not used, present here for API consistency by convention.

Returns:

feature_names_outndarray of str objects

Transformed feature names.

get_params(deep=True)[source]

Get parameters for this estimator.

Parameters:

deepbool, default=True

If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns:

paramsdict

Parameter names mapped to their values.

get_stop_words()[source]

Build or fetch the effective stop words list.

Returns:

stop_words: list or None

A list of stop words.

property idf_

Inverse document frequency vector, only defined if use_idf=True.

Returns:

ndarray of shape (n_features,)

inverse_transform(X)[source]

Return terms per document with nonzero entries in X.

Parameters:

X{array-like, sparse matrix} of shape (n_samples, n_features)

Document-term matrix.

Returns:

X_invlist of arrays of shape (n_samples,)

List of arrays of terms.

set_params(**params)[source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as Pipeline). The latter have parameters of the form <component>_<parameter> so that it's possible to update each component of a nested object.

Parameters:

**paramsdict

Estimator parameters.

Returns:

selfestimator instance

Estimator instance.

transform(raw_documents)[source]

Transform documents to document-term matrix.

Uses the vocabulary and document frequencies (df) learned by fit (or fit_transform).

Parameters:

raw_documentsiterable

An iterable which generates either str, unicode or file objects.

Returns:

Xsparse matrix of (n_samples, n_features)

Tf-idf-weighted document-term matrix.

5.4 SKLEARN COSINE_SIMILARITY

COMPUTE COSINE SIMILARITY BETWEEN SAMPLES IN X AND Y.

COSINE SIMILARITY, OR THE COSINE KERNEL, COMPUTES SIMILARITY AS THE NORMALIZED DOT PRODUCT OF X AND Y:

K(X, Y) = <X, Y> / (||X||*||Y||)

ON L2-NORMALIZED DATA, THIS FUNCTION IS EQUIVALENT TO LINEAR_KERNEL.

READ MORE IN THE USER GUIDE.

PARAMETERS:

X{NDARRAY, SPARSE MATRIX} OF SHAPE (N_SAMPLES_X, N_FEATURES)

INPUT DATA.

Y{NDARRAY, SPARSE MATRIX} OF SHAPE (N_SAMPLES_Y, N_FEATURES), DEFAULT=NONE

INPUT DATA. IF NONE, THE OUTPUT WILL BE THE PAIRWISE SIMILARITIES BETWEEN ALL SAMPLES IN X.

DENSE_OUTPUTBOOL, DEFAULT=TRUE

WHETHER TO RETURN DENSE OUTPUT EVEN WHEN THE INPUT IS SPARSE. IF FALSE, THE OUTPUT IS SPARSE IF BOTH INPUT ARRAYS ARE SPARSE.

NEW IN VERSION 0.17: PARAMETER DENSE_OUTPUT FOR DENSE OUTPUT.

RETURNS:

KERNEL MATRIXNDARRAY OF SHAPE (N_SAMPLES_X, N_SAMPLES_Y)

RETURNS THE COSINE SIMILARITY BETWEEN SAMPLES IN X AND Y.

CHAPTER 6
MODULES

6.1 GATHER DATA AND USE CASES

- Determine the specific use cases or tasks you want the banking bot to handle. For example, ask questions about banking system etc.
- Collect a dataset of banking-related text examples to train and test your NLP model. This dataset should include user queries or statements and corresponding intents or actions.

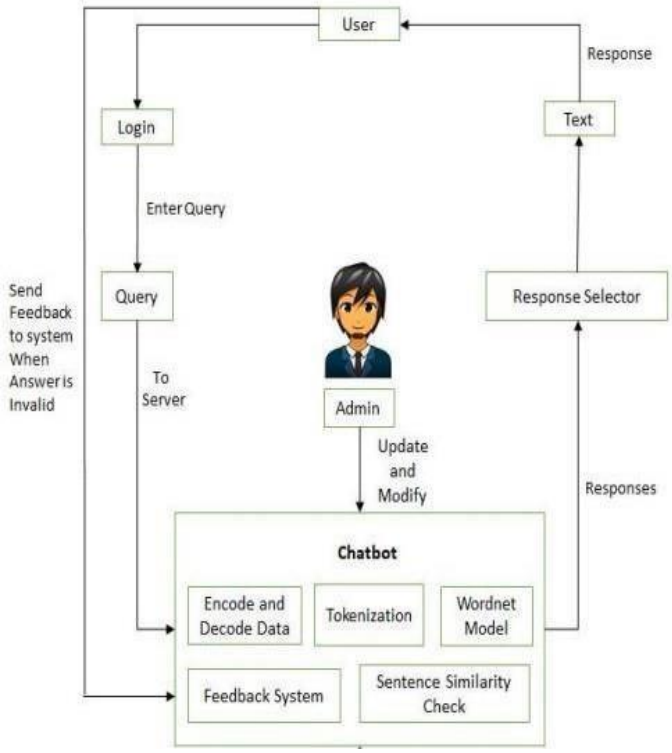


Fig 6.1.1. Database Diagram

6.2 PREPROCESS AND LABEL DATA

- Clean and preprocess the collected data. This step may involve removing noise, normalizing text, and tokenizing sentences.
- Assign labels or intents to each user query or statement in your dataset. This step is crucial for training a supervised learning model.

6.3 CHOOSE AND TRAIN NPL MODEL

- Select an NLP framework or library that suits your requirements. Popular choices include NLTK, spaCy, or TensorFlow's Keras API.
- Build and train an NLP model using the labeled dataset. This typically involves training a classifier or a sequence-to-sequence model depending on the complexity of your bot.

6.4 IMPLEMENT THE BOT LOGIC

- Evaluate the performance of your NLP model using metrics such as accuracy, precision, recall, and F1 score. Fine-tune the model if necessary.
- Develop the logic for your banking bot. This includes handling user queries, understanding intents, and generating appropriate responses. Use the trained NLP model to classify user queries and determine the actions to take.

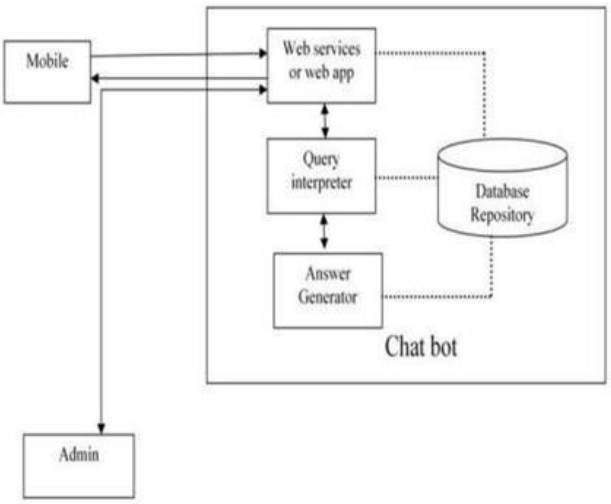


Fig 6.4.1 System Architecture

6.5 BUILD USER INTERFACE

- Create a user interface to interact with the banking bot. This can be a command-line interface, a web application, or even integration with existing chat platforms.
- Connect your banking bot with the backend systems such as databases, APIs, or other banking services to perform actual transactions or retrieve account information.

CHAPTER 7

SYSTEM ANALYSIS

7.1 FEASIBILITY STUDY

A feasibility analysis is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. Feasibility study lets the developer to foresee the project and the usefulness of the system proposal as per its workability. It impacts the organization, ability to meet the user needs and effective use of resource. Thus, when a new application is proposed it normally goes through a feasibility study before it is approved for development.

Three key consideration involved in the feasibility analysis are,

- Technical feasibility
- Operational feasibility
- Economic feasibility
- Technical feasibility

This phase focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the ideas can be converted into working system model. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of the proposed system.

7.2 OPERATIONAL FEASIBILITY

This phase involves undertaking a study to analyse and determine how well the organization's needs can be met by completing the project. Operational feasibility

study also examines how a project plan satisfies the requirements that are needed for the phase of system development.

7.3 ECONOMIC FEASIBILITY

This phase typically involves a cost benefits analysis of the project and help the organization to determine the viability, cost-benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility. It helps the decision-makers to determine the positive economic benefits of the organization that the proposed project will provide.

CHAPTER 8
SOFTWARE TESTING

8.1 SYSTEM TESTING

System testing is the stage of implementation that is aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes logical assumption that if all the parts of the system are correct, then the goal will be successfully achieved. System testing involves user training system testing and successful running of the developed proposed system. The user tests the developed system and changes are made per their needs. The testing phase involves the testing of developed system using various kinds of data. While testing, errors are noted and the corrections are made. The corrections are also noted for the future use.

8.2 UNIT TESTING:

Unit testing focuses verification effort on the smallest unit of software design, software component or module. Using the component level design description as a control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors those uncover is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This is normally considered as an adjunct to the coding step. The design of unit tests can be performed before coding begins.

8.3 BLACK BOX TESTING:

Black box testing also called behavioural testing, focuses on the functional requirement of the software. This testing enables to derive set of input conditions of all functional requirements for a program. This technique focuses on the information domain of the software, deriving test cases by partitioning the input and output of a program.

8.4 WHITE BOX TESTING:

White box testing also called as glass box testing, is a test case design that uses the control structures described as part of component level design to derive test cases. This test case is derived to ensure all statements in the program have been executed at least once during the testing and that all logical conditions have been exercised.

8.5 INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the software architecture to conduct errors associated with interfacing. Top-down integration testing is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module. Bottom-up integration testing begins the construction and testing with atomic modules. Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available.

8.6 VALIDATION TESTING

Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package. The testing focuses on user visible actions and user recognizable output from the system. The testing has been conducted on possible condition such as the function characteristic conforms the specification and a deviation or error is uncovered. The alpha test and beta test is conducted at the developer site by end-users.

CHAPTER 9
CONCLUSION

9.1 CONCLUSION

In banking bot using Natural Language Processing (NLP) has the potential to revolutionize the way customers interact with their bank. By using NLP technology, the bot can understand the meaning behind customer queries and provide helpful responses, allowing customers to perform various banking tasks more efficiently and conveniently.

The primary objective of a banking bot using NLP is to enhance the customer experience by providing a fast, convenient, and personalized way to interact with the bank. This can help build customer loyalty and increase customer satisfaction.

In addition to improving the customer experience, a banking bot using NLP can also help banks reduce costs by automating many routine customer service tasks. This can free up bank staff to focus on more complex issues that require human attention, ultimately improving operational efficiency and reducing costs for the bank.

However, there are still challenges to be addressed in implementing a banking bot using NLP. These include ensuring data privacy and security, addressing potential bias in NLP algorithms, and ensuring the bot's responses are accurate and helpful.

Despite these challenges, a banking bot using NLP is a promising technology that can greatly benefit both customers and banks. As technology continues to evolve, we can expect to see more advanced banking bots that use NLP to provide even more personalized and efficient service.

9.2 FUTURE SCOPE

There are limitations to what has been currently achieved with chatbots. The limitations of data processing and retrieval are hindering chatbots to reach their full potential. It is not that we lack the computational processing power to do so. However, there is a limitation on “How” we do it. One of the biggest examples is the retail customer market. Retail customers are primarily interested in interacting with humans because of nature of their needs. They don’t want bots to process their needs and respond accordingly.

CHAPTER 10

APPENDICES

10.1 SOURCE CODE

```
from chatterbot import ChatBot

from chatterbot.trainers import ListTrainer

import pandas as pd


import os

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split as tts

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.preprocessing import LabelEncoder as LE

from sklearn.metrics.pairwise import cosine_similarity


import nltk

from nltk.stem.lancaster import LancasterStemmer

from nltk.corpus import stopwords

import datetime


stop_words = set(stopwords.words('english'))
```

```
#print(stop_words)

def cleanup(sentence):

    word_tok = nltk.word_tokenize(sentence)
    stemmed_words = [w for w in word_tok if not w in stop_words]
    #print(stemmed_words)

    #stemmed_words = [stemmer.stem(w) for w in word_tok]
    return ' '.join(stemmed_words)

le = LE()
tfv = TfidfVectorizer(min_df=1, stop_words='english')

data = pd.read_csv("C:\\Users\\Rohit\\Desktop\\BankFAQs.csv")

questions = data['Question'].values

X = []
for question in questions:
```

```
    X.append(cleanup(question))
tfv.fit(X)
le.fit(data['Class'])

X = tfv.transform(X)
y = le.transform(data['Class'])

trainx, testx, trainy, testy = tts(X, y, test_size=.3, random_state=42)
model = SVC(kernel='linear')
model.fit(trainx, trainy)
class_=le.inverse_transform(model.predict(X))

#print("SVC:", model.score(testx, testy))

def get_max5(arr):
    ixarr = []
    for ix, el in enumerate(arr):
        ixarr.append((el, ix))
    ixarr.sort()
    ixs = []
```

```
    for i in ixarr[-5:]:

        ixs.append(i[1])

    return ixs[::-1]
def get_response(usrText):
    while True:
        if usrText.lower() == "bye":
            return "Bye"

        GREETING_INPUTS = ["hello", "hi", "greetings", "sup", "what's up",
"hey","hiii","hii","yo"]

        a = [x.lower() for x in GREETING_INPUTS]

        sd=["Thanks","Welcome"]

        d = [x.lower() for x in sd]
        am=["OK"]

        c = [x.lower() for x in am]
```

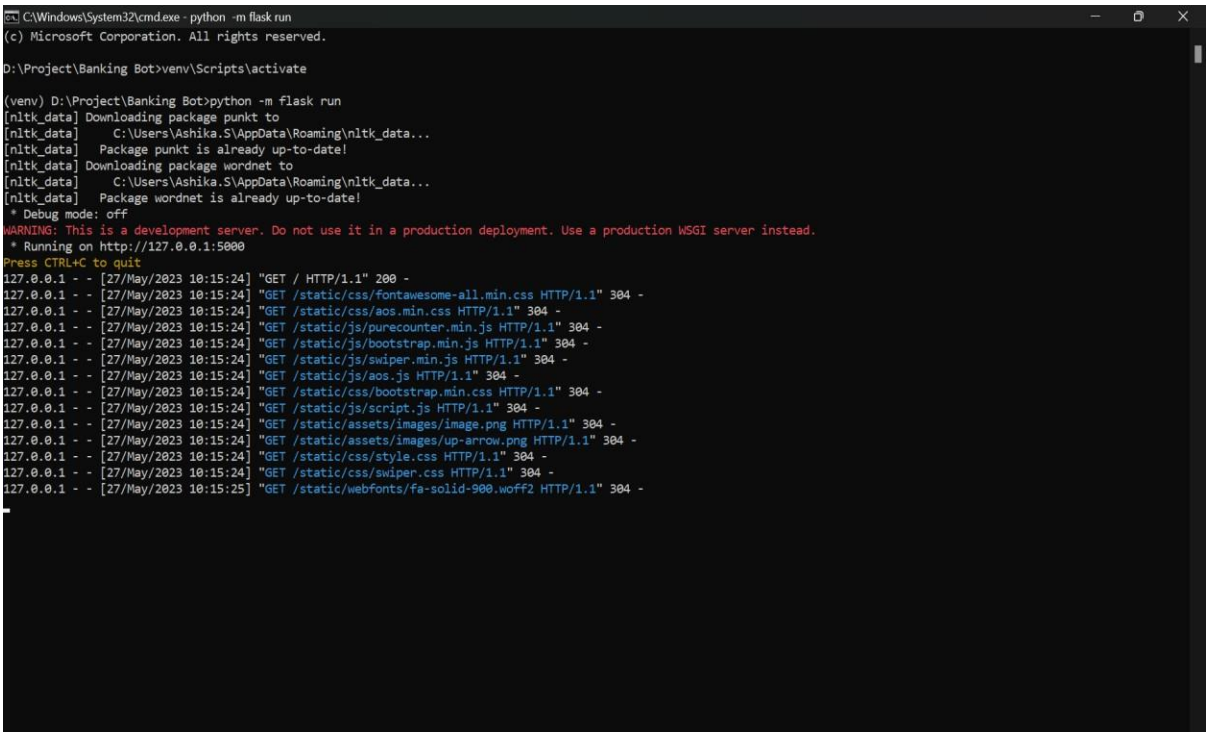
```
# ty = ["getting"]

# r = [x.lower() for x in ty]


 t_usr = tfv.transform([cleanup(usrText.strip().lower())])

 class_ = le.inverse_transform(model.predict(t_usr))


 questionset = data[data['Class'].values == class_]

 cos_sims = []

 for question in questionset['Question']:

     sims = cosine_similarity(tfv.transform([question]), t_usr)

     cos_sims.append(sims)


 ind = cos_sims.index(max(cos_sims))

 b = [questionset.index[ind]]


 if usrText.lower() in a:

     return ("Hi, I'm Emily!\U0001F60A")


 if usrText.lower() in c:
```

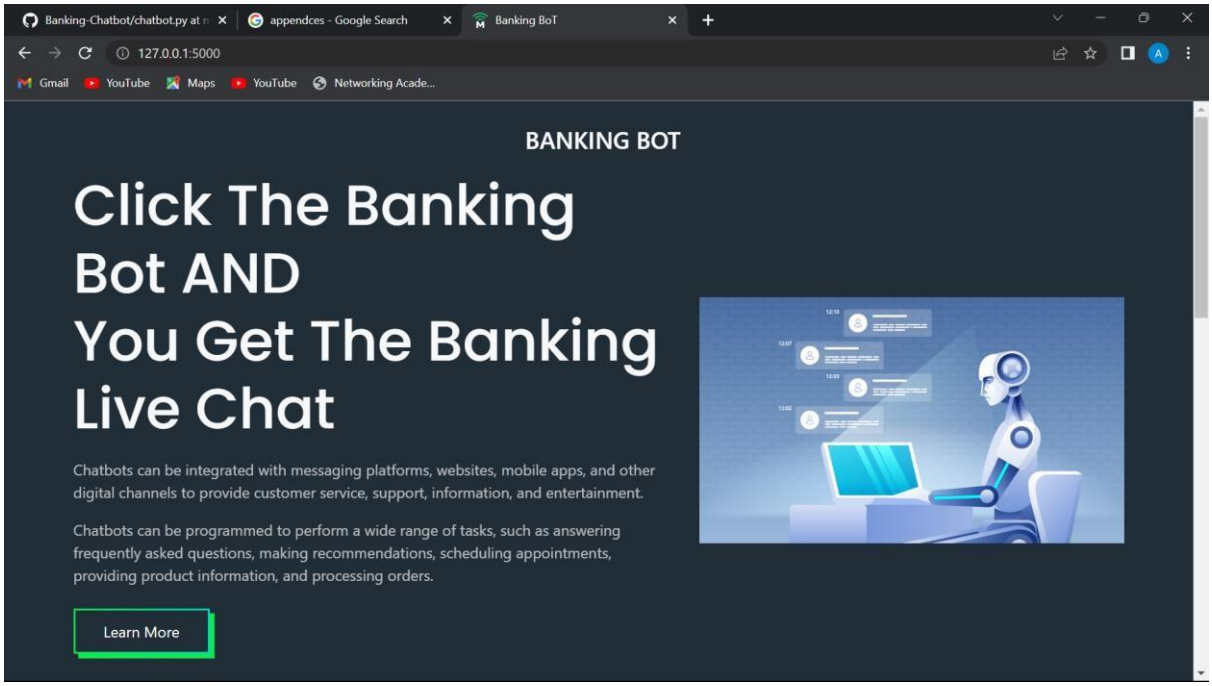
```
            return "Ok...Alright!\U0001F64C"


        if usrText.lower() in d:
            return ("My pleasure! \U0001F607")
        if max(cos_sims) > [[0.]]:
            a = data['Answer'][questionset.index[ind]]+"    "
            return a


        elif max(cos_sims)==[[0.]]:
           return "sorry! \U0001F605"


def get_response2(usr):
    if usr.lower() == "bye":
        return "Thanks for having conversation! \U0001F60E"


    GREETING_INPUTS = ["hello", "hi", "greetings", "sup", "what's up",
"hey","hii","hiii","hiiiii","yo","Hey there"]
    a = [x.lower() for x in GREETING_INPUTS]
    sd = ["Thanks", "Welcome"]
    d = [x.lower() for x in sd]
    am = ["OK"]
```

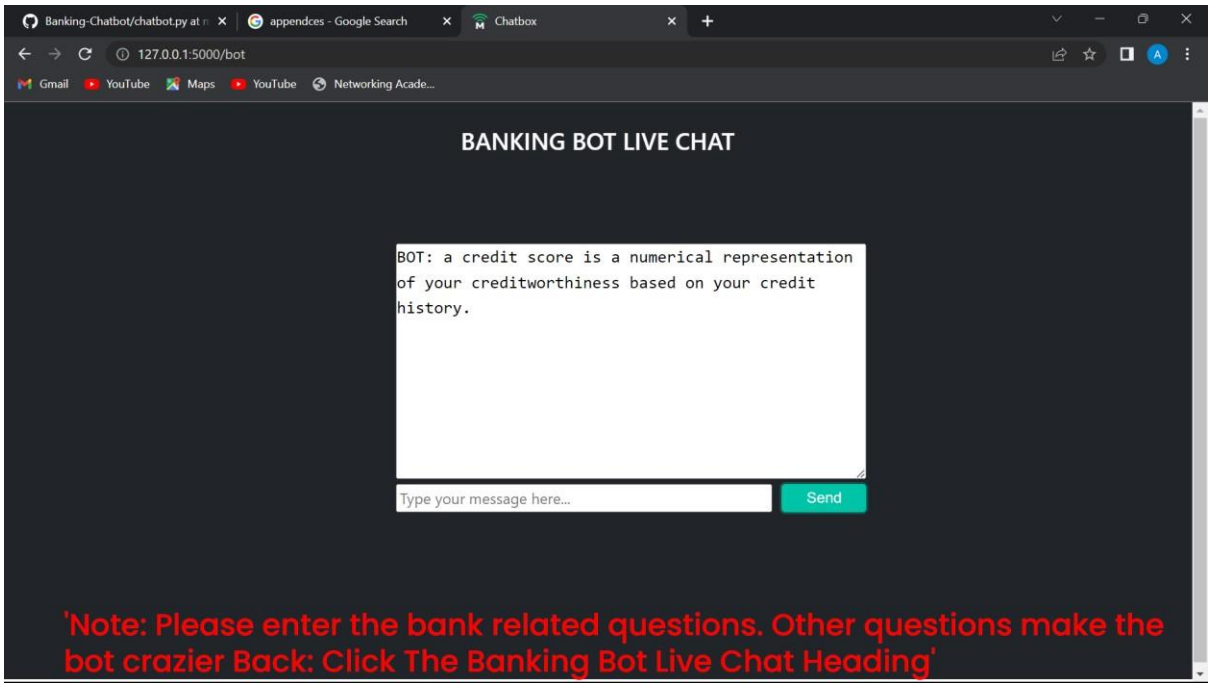
```
    c = [x.lower() for x in am]
    t_usr = tfv.transform([cleanup(usr.strip().lower())])
    class_ = le.inverse_transform(model.predict(t_usr))

    questionset = data[data['Class'].values == class_]

    cos_sims = []
    for question in questionset['Question']:
        sims = cosine_similarity(tfv.transform([question]), t_usr)
        cos_sims.append(sims)
    ind = cos_sims.index(max(cos_sims))
    b = [questionset.index[ind]]

    if usr.lower() in a:
        return ("you can ask me questions related to: Accounts, Investments,
Funds, etc.")
    if usr.lower() in c:
        return " Cool! \U0001f604"
    if usr.lower() in d:
        return ("\U0001F44D")
    if max(cos_sims) == [[0.]]:
```

```
        return "I'm not able to solve this question at this moment. You can call to customer support 1860 999 9999 \U0001F615"
    if max(cos_sims) > [[0.]]:
        inds = get_max5(cos_sims)
        print(inds)
        b = "(1)" + data['Question'][questionset.index[0]]
        c = "(2)" + data['Question'][questionset.index[1]]
        d = "(3)" + data['Question'][questionset.index[2]]
        e = "(4)" + data['Question'][questionset.index[3]]
        f = "(5)" + data['Question'][questionset.index[4]]

        return "Following are the Recommended Questions----->" + b + c + d + e + f
```

10.2 SCREENSHOTS






Banking-Chatbot/chatbot.py at
appendces - Google Search
Chatbox
127.0.0.1:5000/bot
Gmail
YouTube
Maps
YouTube
Networking Acade...
BANKING BOT LIVE CHAT
BOT: a credit score is a numerical representation of your creditworthiness based on your credit history.
Type your message here...
Send
'Note: Please enter the bank related questions. Other questions make the bot crazier Back: Click The Banking Bot Live Chat Heading'

CHAPTER 11

REFERENCE

- Tiong LK, “Risks and guarantees in BOT tender,” Journal of Construction Engineeringand Management,pp.183-187,Jun 1995.
- Tiong LK, “Final negotiation in competitive BOT tender,” Journal of Construction Engineering and Management,pp.6-10,Jan1997.
- Zayed TM and Chang LM, “Prototype model for build-operate-transfer risk assessment, ”Journal of Management in Engineering, pp.7-16, Jan 2002.
- David AK, “Risk modeling in energy contracts between host utilities and BOT plant investors,” IEEE Transactions on Energy Conversion, pp. 359-366,Jun l996.
- Bell DE, “Risk, return, and utility,” Management Science,pp.23-30,Jan 1995.
- Feng CM and Kang CC, “Risk identication and measurement of BOT projects, ”Journal of the Eastern Asia Society for TransportationStudies ,pp.331-350,Apr 1999.
- De Silva, Garza AG and Maher ML, “An evolutionary approach to case adaptation, case-based reasoning research and applications,” Proceedings of the Third International Conference on Case-BasedReasoning, ICCBR-99, Munich,Jul 1999.
- J.Toussaint and K.Cheng, “Web-basedCBR(case-based reasoning) as a tool with the application to tooling selection,” Springer- Verlag London Limited 2005 , pp. 24- 34, Jul 2005.