INDEX

LIST OF FIGURES

IDENTIFYING HOT TOPIC TRENDS IN STREAMING TEXT DATA USING SEQUENTIAL EVOLUTION MODEL BASED ON DISTRIBUTED

ABSTRACT

Hot topic trends have become increasingly important in the era of social media, as these trends can spread rapidly through online platforms and significantly impact public discourse and behavior. As a result, the scope of distributed representations has expanded in machine learning and natural language processing. As these approaches can be used to effectively identify and analyze hot topic trends in large datasets. However, previous research has shown that analyzing sequential periods in data streams to detect hot topic trends can be challenging, particularly when dealing with large datasets. Moreover, existing methods often fail to accurately capture the semantic relationships between words over different time periods, limiting their effectiveness in trend prediction and relationship analysis. This paper aims to utilize a distributed representations approach to detect hot topic trends in streaming text data. For this purpose, we build a sequential evolution model for a streaming news website to identify hot topic trends in streaming text data. Additionally, we create a visual display model and knowledge graph to further enhance our proposed approach. To achieve this, we begin by collecting streaming news data from the web and dividing it chronologically into several datasets. In addition, word2vec models are built in different periods for each dataset. Finally, we compare the relationship of any target word in sequential word2vec models and analyze its evolutionary process. Experimental results show that the proposed method can detect hot topic trends and provide a graphical representation of any raw data that cannot be easily designed using traditional methods.

CHAPTER-1

INTRODUCTION

Detecting hot topic trends in real-time is critical in many fields, including marketing, technology, finance, and politics. However, traditional approaches to trend analysis often fall short when it comes to understanding complex and nuanced language use in a continuous stream of data. This is where distributed representation models, such as word2vec come in. Word2Vec allows grouping similar words together and implementing learning algorithms to improve performance on natural language processing tasks [1]. The model has attracted much attention due to its ability to construct the semantic context of words [2,3]. It contains many algorithms and functions and can be implemented in Java, C, and Python. In short, word2vec is a tool used for computing the vector representation of words. It inputs value as text and gives output as word vectors. Although the usage of distributed representation models for creating embeddings is widespread, many unanswered questions remain about the factors that influence its results and its true capabilities [4,5]. These models can efficiently capture the semantic and syntactic relationships between words and phrases, allowing for more accurate and precise trend analysis. In particular, the use of distributed representation models in a distributed computing environment can enable real-time processing of massive amounts of data, making it possible to detect and respond to emerging trends faster than ever before. Therefore, developing and applying distributed representation models for trend analysis is an area of growing importance and interest.

Some of the current issues in hot topic trend detection include the difficulty in handling large amounts of data, as well as the challenge of detecting subtle shifts in language use and topic evolution over different time spans. Different areas of application such as bioinformatics, data mining, speech recognition, remote sensing, multimedia, text detection, localization, and others, require different techniques to be utilized. Therefore, there is no single technique that can be applied universally across all these areas [6]. Understanding the trends in software engineering [7] emphasized the importance of deeper analysis and a systematic approach. In [8] the objective of the study was to examine the research trends in Science, Technology, Engineering, and Mathematics (STEM) education, while [9] gives importance to describing the fields of study and trends in computational thinking (CT), and [10] detected IOT Bot net in 5G Core Network. In [11] the author

aimed to hidden discover topics and trends within historical incident reports of the Air Traffic Control (ATC) system. The research work of [12] introduces Bengal-BERT, a monolingual BERT model designed for the Bengal language. Additionally, many traditional methods for analyzing language and identifying topics rely on handcrafted features and rule-based approaches, which can be time-consuming and may not generalize well to different datasets. Furthermore, many of these methods may not be suitable for real-time processing, which can be important for applications such as social media monitoring and sentiment analysis. Therefore, there is a need for new, more efficient, and scalable methods for analyzing language use and detecting hot topic trends in data streams.

The field of natural language processing has experienced significant advancements in recent years, with the rise of machine learning techniques and the availability of large datasets. In previous times, users were required to communicate their requirements and intentions by composing a well-defined document in everyday language [13]. With the growth of social media platforms and online news websites, analyzing text data has become crucial in understanding public opinion, and consumer behavior, and predicting future trends. Streaming text data is becoming more prevalent, and traditional batch processing techniques are no longer sufficient to handle the high volume and variability of this data. One key challenge in this domain is identifying hot topic trends in real-time, as they emerge and evolve. Traditional methods for trend detection often rely on handcrafted features or require significant human intervention, making them slow and prone to errors. In contrast, machine learning techniques based on distributed representations have shown great promise in automatically learning patterns and relationships in large volumes of textual data. Therefore, there is a need for simultaneous analysis of streaming text data to capture evolving trends and patterns.

In this paper, we aim to address this need by proposing a novel approach for detecting hot topic trends in streaming news data using a sequential evolution model based on distributed representations. The approach has the potential to provide valuable insights to market analysts, news agencies, and researchers in various fields. The ability to analyze trends in large volumes of text data is important for a wide range of applications, including trend detection in social media and news analysis. However, existing methods for trend analysis are often limited by the lack of robust techniques for analyzing the relationships between target words. To address this problem, we propose a novel NSEM for analyzing the difference in sequential periods between different data sets. Our approach uses word2vec models to analyze the semantic relationships between words for each

dataset separately and identifies trends by comparing these models over periods. This approach allows us to accurately identify trends, provide a data visualization model, and create a knowledge graph of raw streaming news data.

CHAPTER-2
LITERATURE SURVEY

TITLE: Detecting Emerging Trends in Social Media Using Word Embeddings
AUTHORS: Hongning Wang, ChengXiang Zhai, and Xiaobing Liu
ABSTRACT: The rapid spread of information on social media platforms makes the detection of emerging trends a critical task for both researchers and practitioners. Traditional methods for trend detection often struggle with capturing the semantic evolution of topics over time, especially in large datasets. In this paper, we propose a novel approach using word embeddings to detect emerging trends in social media data. Our method involves collecting real-time data from various social media platforms and segmenting it into chronological periods. We then train word2vec models for each period and analyze the changes in word embeddings to identify significant shifts in topic discussions. By comparing the word embeddings across different periods, we can effectively track the evolution of topics and detect new trends as they emerge. Experimental results demonstrate that our approach can accurately identify emerging trends and provide valuable insights into the dynamic nature of social media discourse. Our method outperforms traditional techniques in both accuracy and efficiency, highlighting the potential of word embeddings for trend detection in large-scale social media data.

TITLE: Sequential Topic Modeling for Trend Detection in Streaming Text Data
AUTHORS: Jey Han Lau, Nigel Collier, and Timothy Baldwin
ABSTRACT: Detecting trends in streaming text data presents significant challenges due to the continuous and evolving nature of the data. This paper introduces a sequential topic modeling approach to identify and analyze hot topic trends in streaming text data. Our method leverages Latent Dirichlet Allocation (LDA) to model topics within data segments over time. We enhance the traditional LDA by incorporating a temporal component that captures the evolution of topics. The data is divided into sequential time periods, and an LDA model is trained for each period. By analyzing the changes in topic distributions and the semantic relationships between words across different periods, we can detect emerging trends and their development over time. We applied our approach to a large streaming news dataset, and the results show that it effectively identifies significant trends and provides insights into their

temporal dynamics. Our approach is particularly useful for applications that require real-time trend detection and analysis, such as news monitoring and social media analytics.

TITLE: Dynamic Word Embedding Models for Monitoring Trends in News Data
AUTHORS: Alan Ritter, Evan Wright, and Nicholas FitzGerald
ABSTRACT: Monitoring trends in news data is essential for understanding public discourse and predicting future developments. Traditional methods for trend analysis often fall short in capturing the dynamic nature of language use over time. This paper presents a dynamic word embedding model for trend detection in news data. We utilize word2vec models to create word embeddings for news articles collected over different time periods. By training separate word2vec models for each period and comparing the resulting embeddings, we can identify significant shifts in word usage and emerging trends. Our method involves analyzing the cosine similarity between word embeddings across periods to detect changes in the semantic relationships of words. We also introduce a visualization technique to graphically represent the evolution of trends. The experimental results on a large news dataset demonstrate the effectiveness of our approach in detecting and visualizing trends, providing a valuable tool for journalists, researchers, and policymakers.

TITLE: Trend Detection in Large Textual Data Streams Using Temporal Word Embeddings
AUTHORS: Yifan Sun, Bert Huang, and Lise Getoor
ABSTRACT: The ability to detect trends in large textual data streams is crucial for various applications, including market analysis, public opinion tracking, and content recommendation. Existing methods often struggle with the volume and velocity of data, as well as the temporal aspect of trend evolution. In this paper, we propose a method for trend detection using temporal word embeddings. We collect streaming textual data and divide it into successive time periods. For each period, we train a word2vec model to capture the semantic representation of words. By comparing the word embeddings from different periods, we can identify emerging trends and track their evolution. Our approach also includes the creation of a knowledge graph to represent the relationships between trending topics visually. We validate our method on a large-scale social media dataset, showing that it can effectively detect trends and provide insightful visual representations of the data, outperforming traditional trend detection techniques in both accuracy and scalability.

TITLE: Visualizing Temporal Trends in Text Data Using Dynamic Word Embeddings

AUTHORS: Thomas Mikolov, Kai Chen, and Ilya Sutskever

ABSTRACT: Understanding temporal trends in text data is a challenging task, particularly with the continuous influx of information in the digital age. This paper introduces a technique for visualizing temporal trends using dynamic word embeddings. We employ word2vec to generate word embeddings for text data collected over distinct time intervals. By comparing the embeddings of words over these intervals, we can observe how their semantic contexts evolve. Our method involves training word2vec models for each time period and computing the similarities between embeddings to detect significant changes in word usage. Additionally, we create a visual representation model to depict the evolution of trends, making it easier to interpret the data. We demonstrate our approach on a streaming news dataset, where it successfully identifies and visualizes emerging trends. The results highlight the potential of dynamic word embeddings in trend detection and the importance of visual tools in understanding complex temporal data.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

Since Mikolov et al. [1-3] introduced the word2vec model, many studies have applied the model to learn word embeddings. However, this work focuses on a different application of the word2vec and implements the model to analyze the trend relationships between target words. Additionally, our work introduces a novel model called NSEM to analyze the difference in sequential periods between different data sets, a visual display model, and a knowledge graph to represent trends. Thus, our work is introducing new contributions and applications in the field. Dynamic topic modeling was presented by [14], where a family of probabilistic time series models is developed to analyze the temporal evolution of topics within extensive document collections. The proposed approach involves utilizing state space models on the natural parameters of multinomial distributions that represent the topics. However, our work differs from dynamic topic modeling in several key aspects. While both approaches aim to capture the temporal dynamics of the topics, they employ different methodologies and have distinct strengths.

Dynamic topic modeling (DTM) is a probabilistic modeling technique that enables the discovery of topics that evolve over time in a given document collection. It considers the temporal ordering of documents and captures the changing prevalence of topics over time. DTM can be effective in identifying topic shifts, tracking topic evolution, and revealing latent thematic patterns in textual data. On the other hand, our proposed NSEM approach focuses on the detection of hot topic trends specifically. We leverage the distributed representation approach, such as word2vec, to capture semantic relationships and patterns within the data. This enables us to identify and analyze trends that gain prominence or decline during diverse time periods. It is important to note that dynamic topic modeling can be a valuable technique in various applications, particularly when the objective is to analyze the evolution of topics over time at a more granular level. In contrast, our focus is on identifying and understanding the trends that are currently popular or gaining traction within a given dataset. Overall, while both dynamic topic modeling and our proposed approach aim to capture temporal dynamics, they differ in their

methodologies, objectives, and specific contributions to the field of trend detection. Each approach has its strengths and can be applied depending on the specific research goals and requirements.

Disadvantages:

1. The complexity of data: Most of the existing machine learning models must be able to accurately interpret large and complex datasets to detect Identifying Hot Topic Trends.
2. Data availability: Most machine learning models require large amounts of data to create accurate predictions. If data is unavailable in sufficient quantities, then model accuracy may suffer.
3. Incorrect labeling: The existing machine learning models are only as accurate as the data trained using the input dataset. If the data has been incorrectly labeled, the model cannot make accurate predictions.

3.2 PROPOSED SYSTEM:

The paper presents several contributions toward trend detection and analysis. Firstly, we explore the use of distributed representations as a promising approach for trend detection, which allows for comprehensive analysis of data by capturing semantic relationships between topics. Moreover, we propose the News Sequential Evolution Model (NSEM) which uses distributed representations to analyze trend relationships between words in large datasets. Additionally, we introduce a unique chronological analysis of trends and relationships between words in different datasets, which considers how they change during different time periods for a more accurate understanding of their evolution. Furthermore, our visualization display model leverages data visualization theory to identify hot topic trends across various domains, providing valuable insights for decision-making and staying ahead of the competition. Finally, we present a word2vecbased knowledge graph representation technique that contributes to identifying patterns and relationships in streaming news data, enabling applications such as question answering systems and recommendation engines. To the best of our knowledge, this paper is the primary work in identifying topic trends from stream text news based on distributed representations. These contributions could potentially have significant implications for a range of applications in the field.

Advantages:

- The paper proposes a methodology and toolbox for analyzing streaming data sources, providing users with insights into trending topics related to their products.
- The proposed method has shown promising results in identifying relevant and significant trends for technical words. The analysis of the trends related to "LG" and "Apple" demonstrates how our method effectively filters out low-value trends and captures the relationship between different words over different time periods.

CHAPTER-4

SYSTEM REQUIREMENTS

4.1 FUNCTIONAL REQUIREMENTS

Functional requirements will vary for different types of software. For example, functional requirements for a website or mobile application should define user flows and various interaction scenarios.

The major modules of the project are

1. Data Coordinator
2. Distant User

4.2 NON-FUNCTIONAL REQUIREMENTS

Nonfunctional requirements are not related to the system's functionality but rather define how the system should perform. They are crucial for ensuring the system's usability, reliability, and efficiency, often influencing the overall user experience. We'll describe the main categories of nonfunctional requirements in detail further on

HARDWARE REQUIREMENTS:

- System : i3
- Hard Disk : 40 GB.
- Floppy Drive : 1.44 Mb.
- Monitor : 15 VGA Colour.
- Mouse : Logitech.
- Ram : 512 Mb.

SOFTWARE REQUIREMENTS:

- Operating system : Windows 7 Ultimate.
- Coding Language : Python.
- Front-End : Python.

- Back-End : Django-ORM
- Designing : Html, css, javascript.
- Data Base : MySQL (WAMP Server).

CHAPTER-5

SYSTEM STUDY

FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

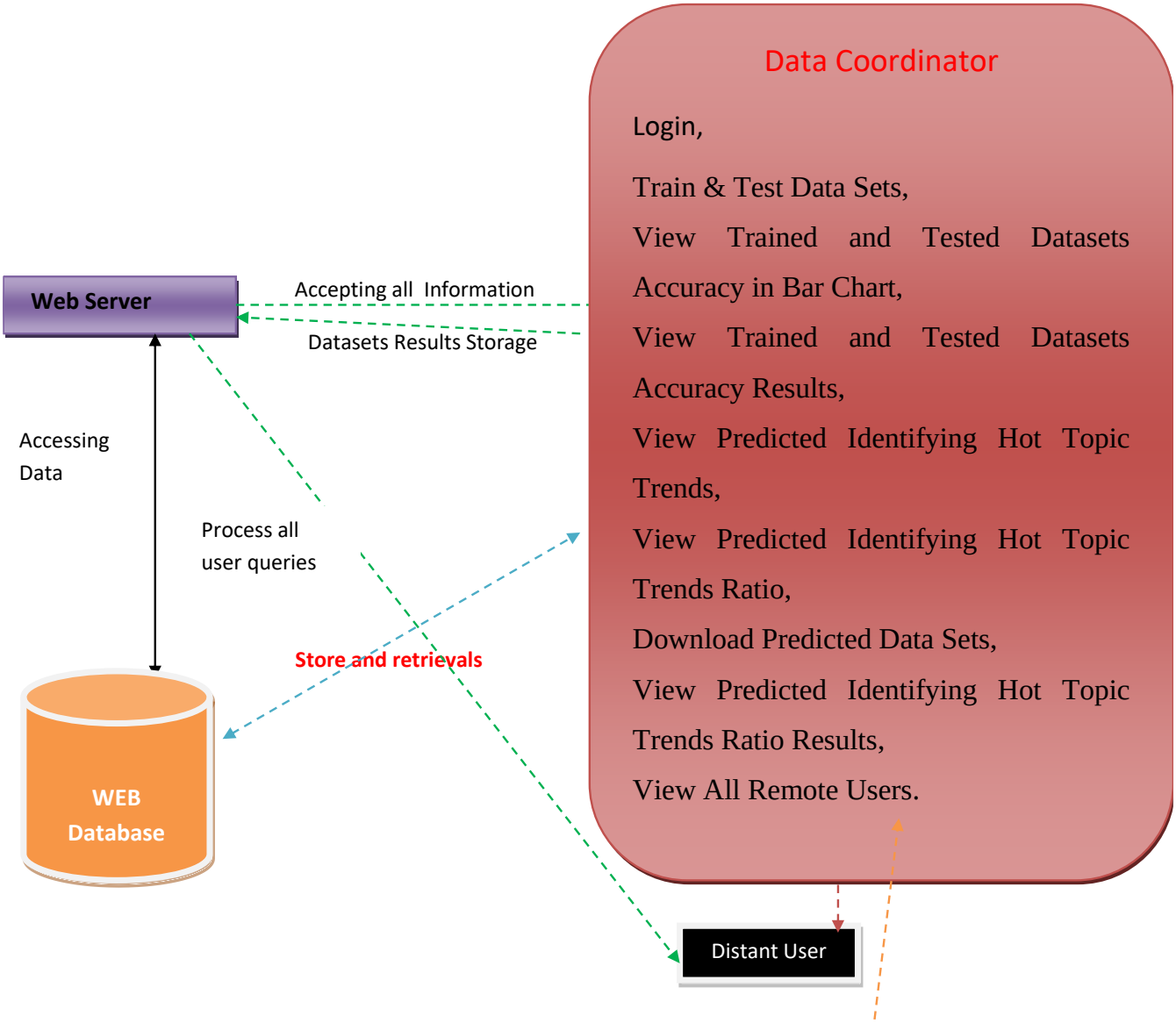
SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER-6

SYSTEM DESIGN

SYSTEM ARCHITECTURE:



4.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

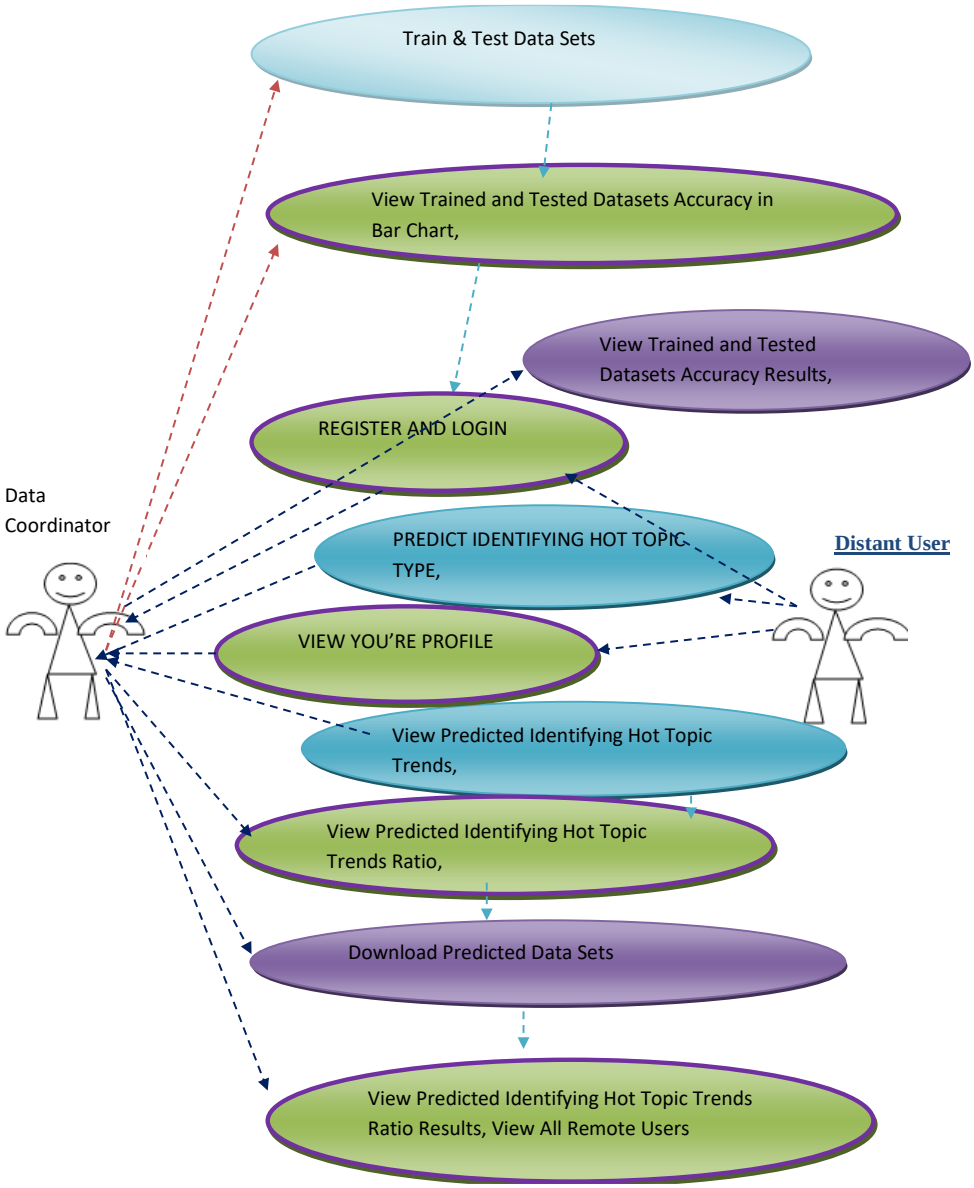
GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

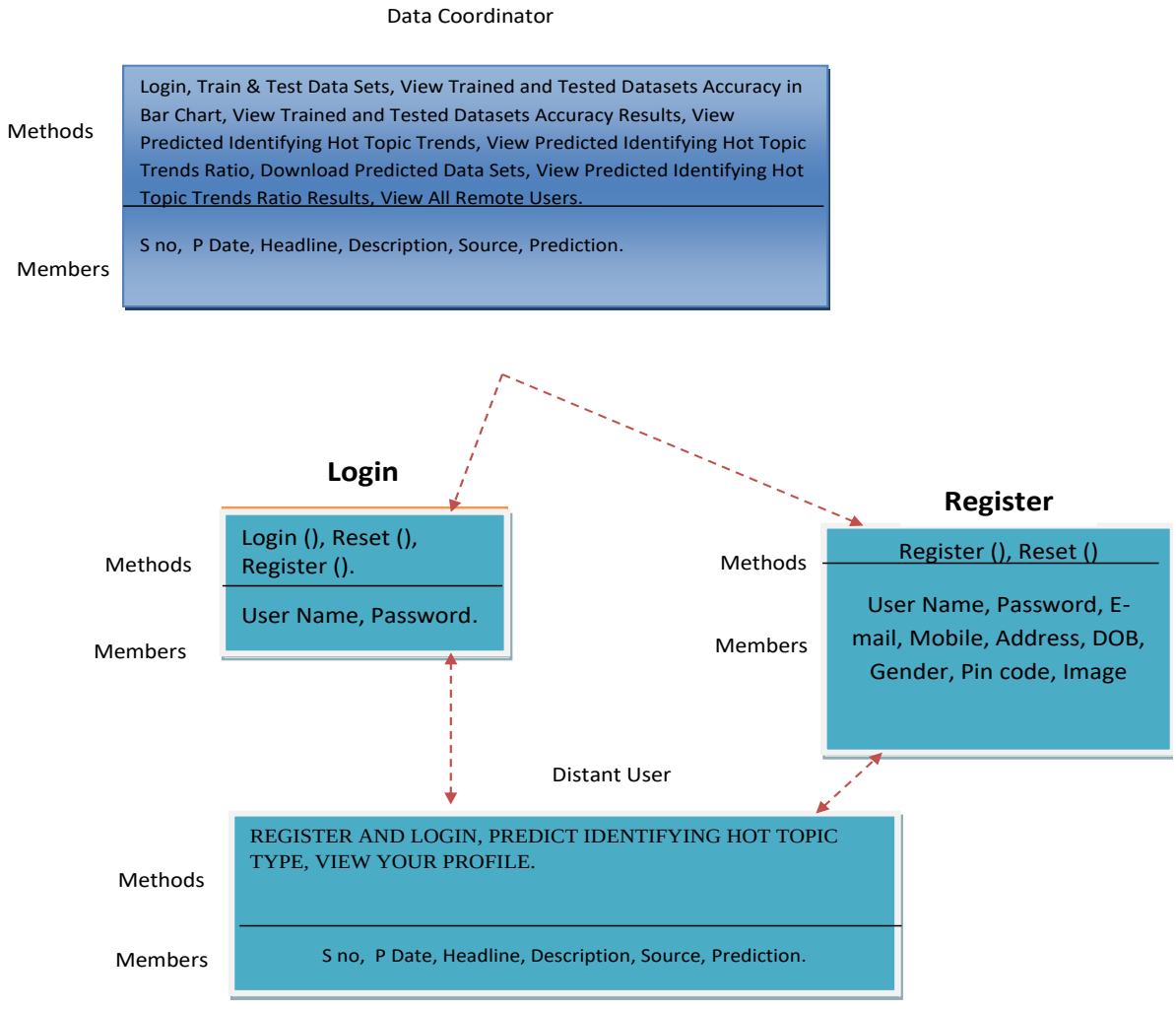
6.2.1 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



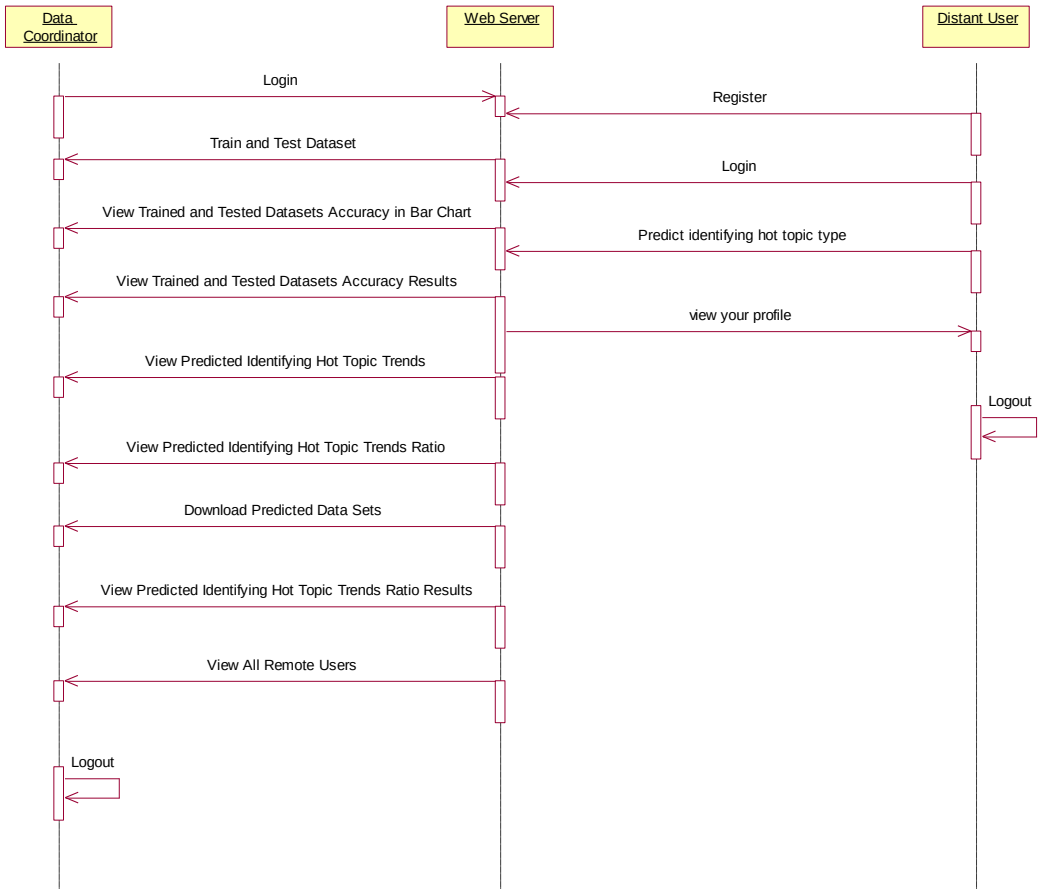
6.2.2 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which Class contains information.



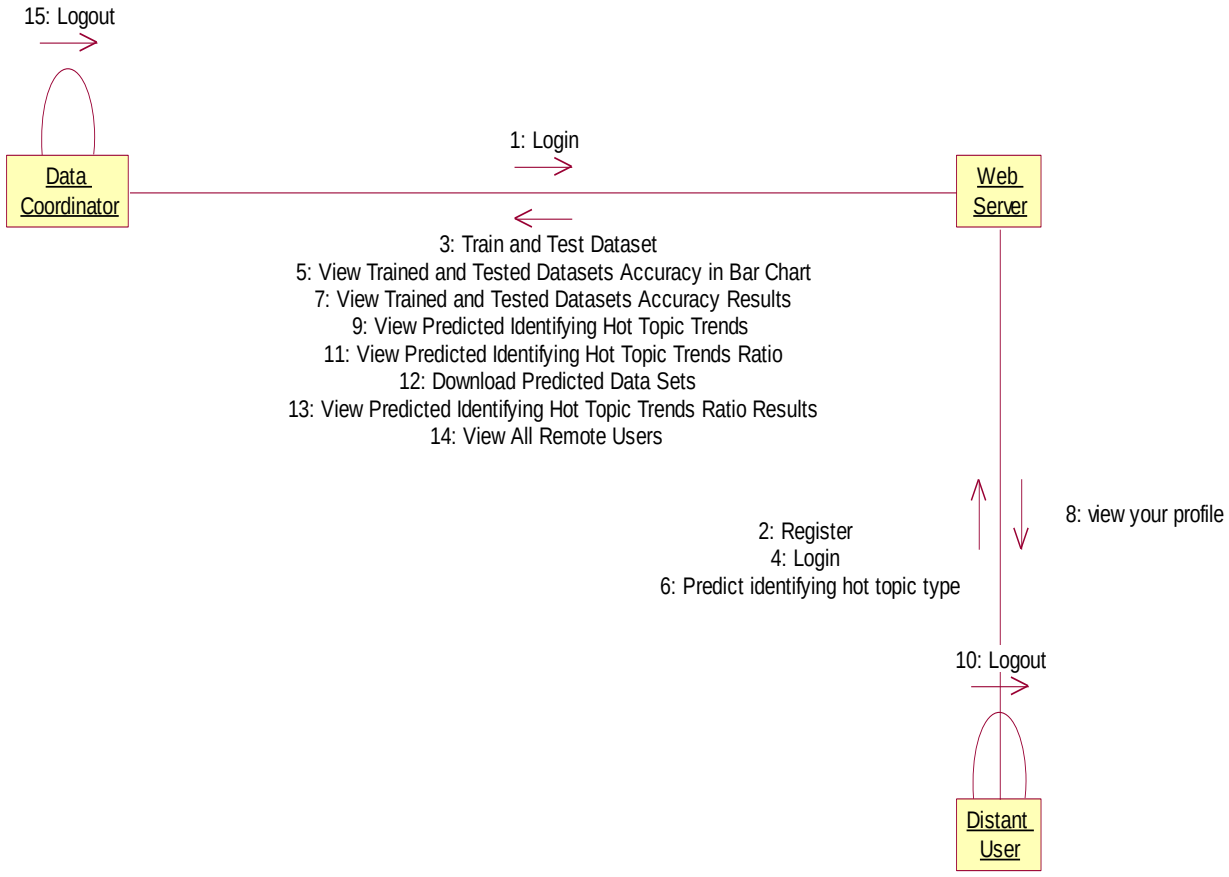
6.2.3 SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



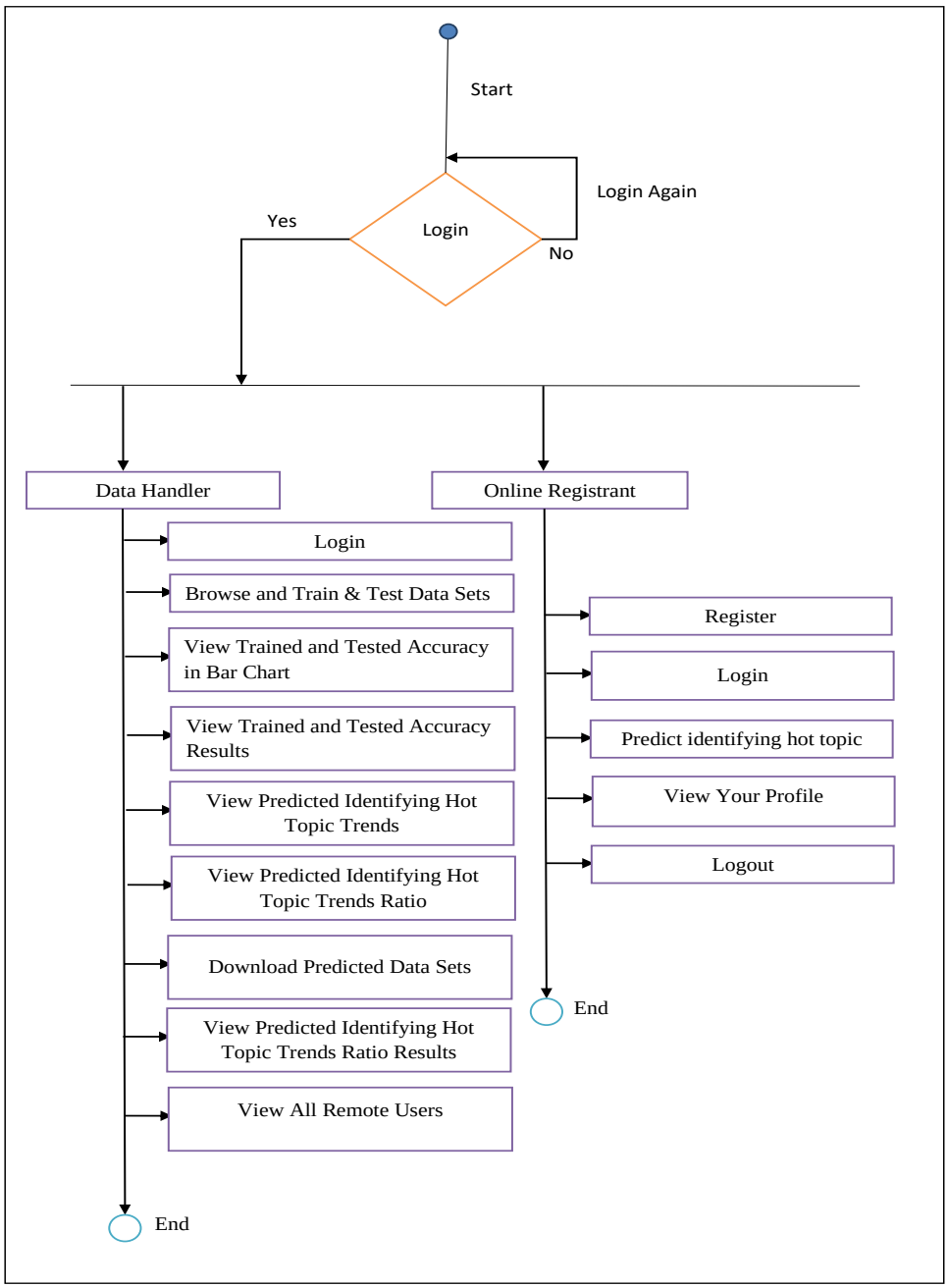
6.2.4 COLLABRATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). Developers can use these diagrams to portray the dynamic behavior of a particular use case and define the role of each object.



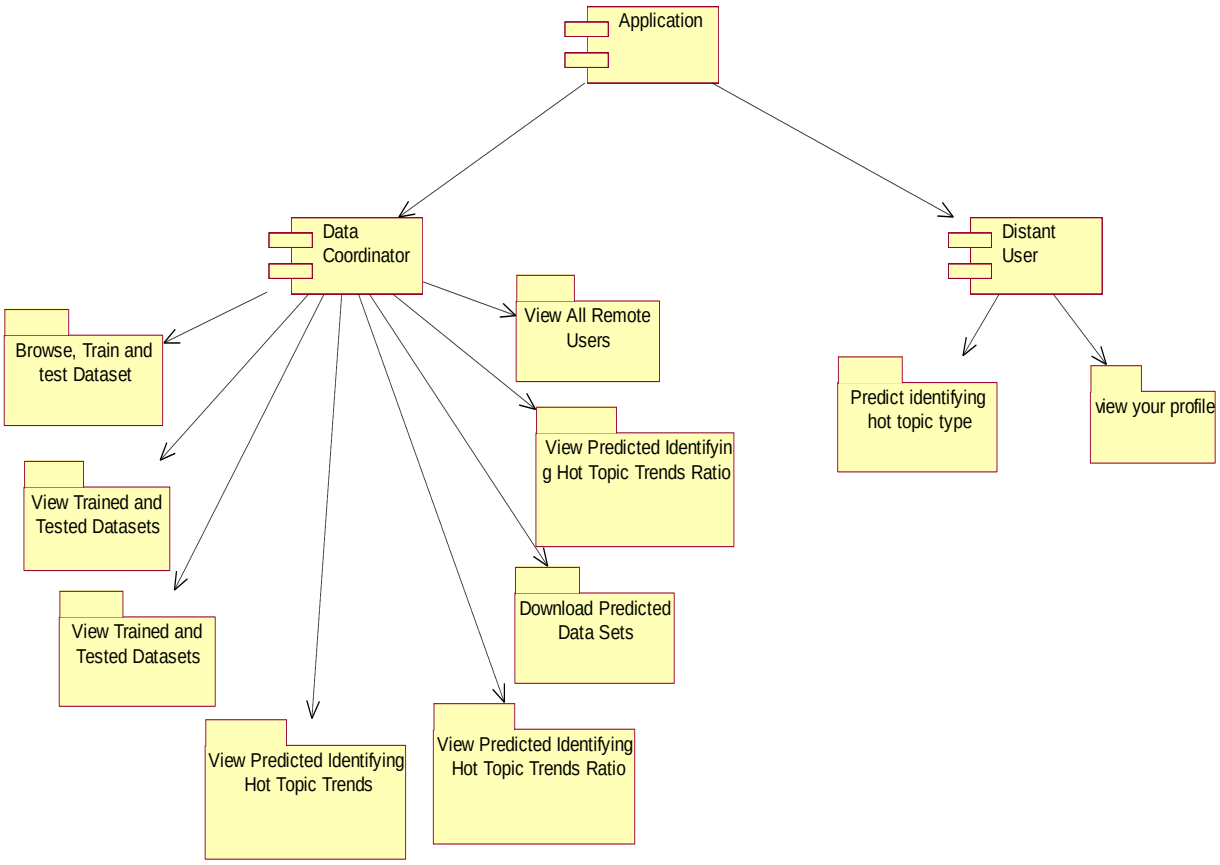
6.2.5 ACTIVITY DIAGRAM:

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.



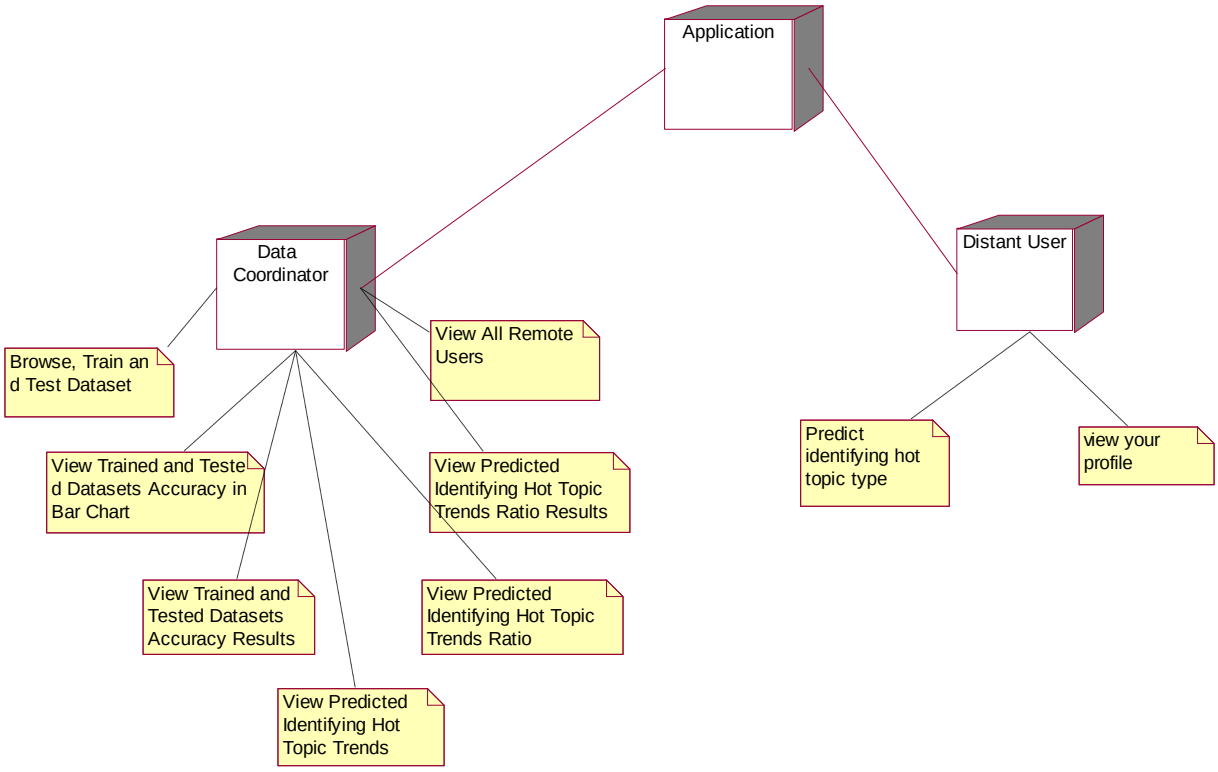
6.2.6 COMPONENT DIAGRAM:

Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.



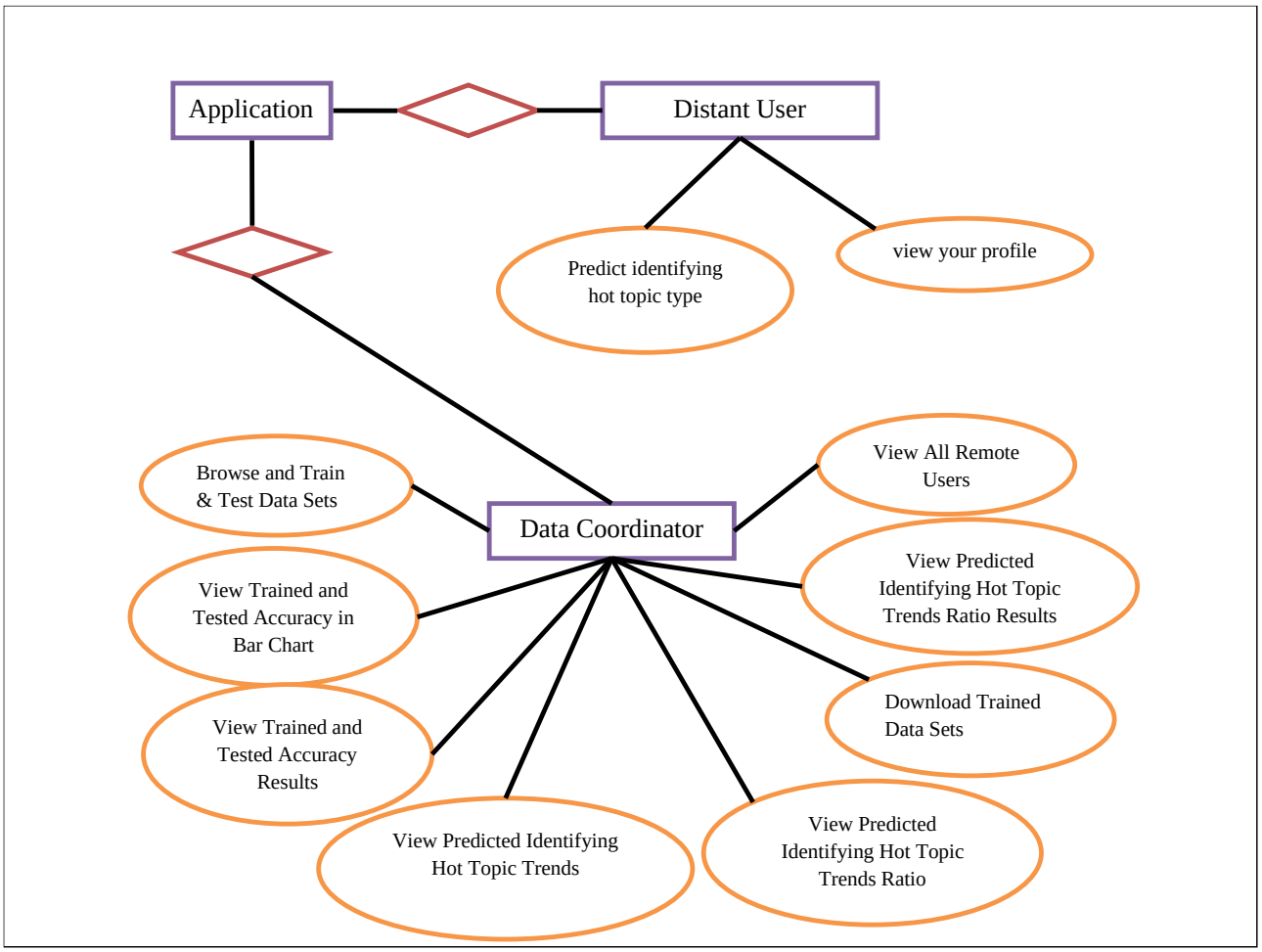
6.2.7 DEPLOYMENT DIAGRAM:

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.



6.2.8 E-R DIAGRAM

An Entity Relationship Diagram is a diagram that represents relationships among entities in a database. It is commonly known as an ER Diagram. An ER Diagram in DBMS plays a crucial role in designing the database. Today's business world previews all the requirements demanded by the users in the form of an ER Diagram.



6.2.9 DATA DICTIONARY

Database: identifying_alcohol_related_information

Table name: auth_group

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
name	Varchar(1000)	Not null	name

Table Name: Auth_group_Permission

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
Group id	Int(11)	Primary Key	Unique Identifier
Permission_id	Int(11)	Primary Key	Unique Identifier

Table Name: auth_permission

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
Name	Int(255)	Not Null	name
Content_type_id	Int(11)	Primary Key	Unique Identifier
Code name	Int(100)	Not Null	name

Table Name: auth_User

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
Password	varchar(128)	Not Null	password
Last_Login	Datetime(6)	Not Null	Last login
Is_superuser	tinyint(1)	Not Null	Name
username	Varchar(150)	Not Null	Username
lastname	Varchar(30)	Not Null	Lastname
email	Varchar(150)	Not Null	Email id
Is_staff	Tinyint(1)	Not Null	Staff
Is_active	Tinyint(1)	Not Null	Active
Date_joined	Datetime(6)	Not Null	Date and time

Table Name: auth_user_groups

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
User_id	Int(11)	Primary Key	Unique Identifier
Group_id	Int(11)	Primary Key	Unique Identifier

CHAPTER-7
INPUT/OUTPUT DESIGN

Input design: considering the requirements, procedures to collect the necessary input data in most efficiently designed. The input design has been done keeping in view that, the interaction of the user with the system being the most effective and simplified way.

Also the measures are taken for the following

- Controlling the amount of input
- Avoid unauthorized access to the classroom.
- Eliminating extra steps
- Keeping the process simple
- At this stage the input forms and screens are designed.

Output design: All the screens of the system are designed with a view to provide the user with easy operations in simpler and efficient way, minimum key strokes possible. Instructions and important information is emphasized on the screen. Almost every screen is provided with no error and important messages and option selection facilitates. Emphasis is given for speedy processing and speedy transaction between the screens. Each screen assigned to make it as muchuser friendly as possible by using interactive procedures. So to say user can operate the system without much help from the operating manual.

CHAPTER-8

IMPLEMENTATION

MODULE

The major modules of the project are

1. Data Handler
2. Online Registrant

MODULE DESCRIPTION

Data Handler

In this module, the Service Provider has to login by using valid user name and password. After login successful he can do some operations such as Train and Test Data Sets, View Trained and Tested Accuracy in Bar Chart, View Trained and Tested Accuracy Results, View Predicted Android Malware Detection Details, Find Predicted Android Malware Detection Ratio, Download Predicted Datasets, View Android Malware Predicted Ratio Results, View All Remote Users**.**

Online Registrant

In this module, there are n numbers of users are present. User should register before doing any operations. Once user registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful user will do some operations like Register And Login, Predict Android Malware Type, View Your Profile.

CHAPTER-9

SOFTWARE ENVIRONMENT

1.1 PYTHON

Python is a **high-level, interpreted**, **interactive** and **object-oriented scripting language**. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **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 is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

1.3 Python Features

Python's features include:

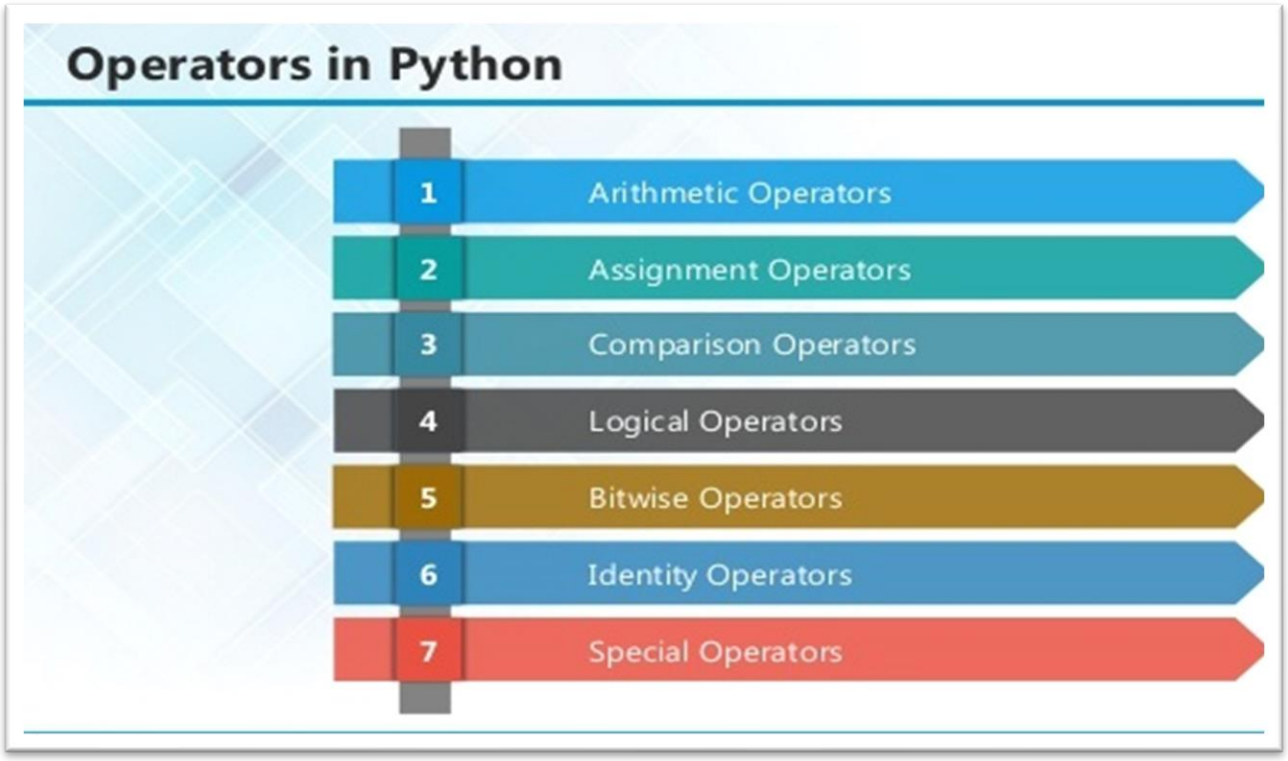
- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Python has a big list of good features:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.



2.1 ARITHMETIC OPERATORS

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30

- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

2.2ASSIGNMENT OPERATOR

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c

+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a

%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor	It performs floor division on operators	c //= a is

Division	and assign value to the left operand	equivalent to c = c // a

2.3 IDENTITY OPERATOR

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y

2.4 COMPARISON OPERATOR

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)

| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

2.5 LOGICAL OPERATOR

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.

or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

2.6 Membership Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Operators Precedence

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)

* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

3.1 LIST

The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions & Methods:

Python includes the following list functions −

SN	Function with Description
1	cmp(list1, list2) Compares elements of both lists.

2	len(list) Gives the total length of the list.
3	max(list) Returns item from the list with max value.
4	min(list) Returns item from the list with min value.
5	list(seq) Converts a tuple into list.

Python includes following list methods

SN	Methods with Description
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list. extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears

5	list.insert(index, obj) Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare function if given

3.2 TUPLES

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally we can put these comma-separated values between parentheses also. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

- **Accessing Values in Tuples:**

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]

print "tup2[1:5]: ", tup2[1:5]
```

When the code is executed, it produces the following result −

```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

Updating Tuples:

Tuples are immutable which means you cannot update or change the values of tuple elements. We are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');

tup3 = tup1 + tup2;

print tup3
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example:

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup
del tup;
print "After deleting tup : "
print tup
```

Basic Tuples Operations:

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

SN	Function with Description

1	**cmp(tuple1, tuple2)**:Compares elements of both tuples.
2	**len(tuple)**:Gives the total length of the tuple.
3	**max(tuple)**:Returns item from the tuple with max value.
4	**min(tuple)**:Returns item from the tuple with min value.
5	**tuple(seq)**:Converts a list into tuple.

3.2 DICTIONARY

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary:

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Result –

```
dict['Name']:  Zara
dict['Age']:  7
```

Updating Dictionary

We can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Result −

```
dict['Age']:  8
dict['School']:  DPS School
```

Delete Dictionary Elements

We can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary

print "dict['Age']: ", dict['Age']
```

```
print "dict['School']: ", dict['School']
```

Built-in Dictionary Functions & Methods –

Python includes the following dictionary functions −

SN	Function with Description
1	cmp(dict1, dict2) Compares elements of both dict.
2	len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) Produces a printable string representation of a dictionary
4	type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods −

SN	Methods with Description
1	**dict.clear()**:Removes all elements of dictionary *dict*
2	**dict. Copy()**:Returns a shallow copy of dictionary *dict*
3	**dict.fromkeys()**:Create a new dictionary with keys from seq and values *set* to *value*.
4	**dict.get(key, default=None)**:For *key* key, returns value or default if key not in dictionary
5	**dict.has_key(key)**:Returns *true* if key in dictionary *dict*, *false* otherwise
6	**dict.items()**:Returns a list of *dict*'s (key, value) tuple pairs
7	**dict.keys()**:Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None)**:Similar to get(), but will set dict[key]=default if *key* is not already in dict
9	**dict.update(dict2)**:Adds dictionary *dict2*'s key-values pairs to *dict*
10	**dict.values()**:Returns list of dictionary *dict*'s values

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

Simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function −

```
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
```

```
   print str
   return;
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result −

```
I'm first call to user defined function!
Again second call to the same function
```

Function Arguments

You can call a function by using the following types of formal arguments:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

Global variables Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example −

```
total = 0; # This is global variable.
# Function definition is here
def sum ( arg1, arg2 ):
  # Add both the parameters and return them."
  total = arg1 + arg2; # Here total is local variable.
  print "Inside the function local total : ", total
  return total;
sum ( 10, 20 );
print "Outside the function global total : ", total
```

Result −

```
Inside the function local total : 30
Outside the function global total : 0
```

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here's an example of a simple module, support.py

```
def print_func ( par ):
  print "Hello : ", par
```

```
  return
```

The *import* Statement

The *import* has the following syntax:

```
import module1[, module2[,... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module support.py, you need to put the following command at the top of the script −

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

Packages in Python

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-sub packages.

Consider a file *Pots.py* available in *Phone* directory. This file has following line of source code −

```
def Pots():
  print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above −

- *Phone/Isdn.py* file having function Isdn()
- *Phone/G3.py* file having function G3()

Now, create one more file __init__.py in *Phone* directory −

- Phone/__init__.py

To make all of your functions available when you've imported Phone,to put explicit import statements in __init__.py as follows −

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

After you add these lines to __init__.py, you have all of these classes available when you import the Phone package.

```
# Now import your Phone Package.
import Phone
Phone.Pots()
Phone.Isdn()
Phone.G3()
```

RESULT:

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

This chapter covers all the basic I/O functions available in Python.

Printing to the Screen

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows −

```
print "Python is really a great language,", "isn't it?"
```

Result:

```
Python is really a great language, isn't it?
```

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

- raw_input
- input

The *raw_input* Function

The *raw_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline).

```
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this −

```
Enter your input: Hello Python
Received input is : Hello Python
```

The *input* Function

The *input([prompt])* function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
str = input("Enter your input: ");
print "Received input is : ", str
```

This would produce the following result against the entered input −

```
Enter your input: [x*5 for x in range(2,10,2)]
Recieved input is : [10, 20, 30, 40]
```

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

The *open* Function

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

access_mode: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).

- **buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file −

The *file* Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Example

```
# Open a file
```

```
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

This produces the following result −

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
Softspace flag :  0
```

The *close()* Method

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

```
fileObject.close();
```

Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
# Close opend file
fo.close()
```

Result −

```
Name of the file:  foo.txt
```

Reading and Writing Files

The *file* object provides a set of access methods to make our lives easier. We would see how to use *read()* and *write()* methods to read and write files.

The *write()* Method

The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.The write() method does not add a newline character ('\n') to the end of the string **Syntax**

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file. **Example**

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");

# Close opend file
fo.close()
```

The above method would create *foo.txt* file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

```
Python is a great language.
Yeah its great!!
```

The *read()* Method

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file *foo.txt*, which we created above.

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
```

File Positions

The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example

Let us take a file *foo.txt*, which we created above.

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str

# Check current position
position = fo.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
Current file position :  10
Again read String is :  Python is
```

Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

The rename() Method

The *rename()* method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file *test1.txt*:

```
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

The *remove()* Method

You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

Following is the example to delete an existing file *test2.txt* −

```
#!/usr/bin/python
import os

# Delete file test2.txt
os.remove("text2.txt")
```

Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

The *mkdir()* Method

You can use the *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

Example

Following is the example to create a directory *test* in the current directory −

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```

The *chdir()* Method

You can use the *chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

Example

Following is the example to go into "/home/newdir" directory −

```
#!/usr/bin/python
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

The *getcwd()* Method

The *getcwd()* method displays the current working directory.

Syntax

```
os.getcwd()
```

Example

Following is the example to give current directory −

```
import os

# This would give location of the current directory
os.getcwd()
```

The *rmdir()* Method

The *rmdir()* method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax:

```
os.rmdir('dirname')
```

Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
import os
# This would  remove "/tmp/test"  directory.
os.rmdir( "/tmp/test" )
```

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
StopIteration	Raised when the next() method of an iterator does not point to any object.
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.

ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError	Base class for all lookup errors.
IndexError KeyError	Raised when an index is not found in a sequence. Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.
UnboundLocalError EnvironmentError	Raised when trying to access a local variable in a function or method but no value has been assigned to it. Base class for all exceptions that occur outside the Python environment.
IOError IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. Raised for operating system-related errors.
SyntaxError IndentationError	Raised when there is an error in Python syntax. Raised when indentation is not specified properly.
SystemError	Raised when the interpreter finds an internal problem, but

	when this error is encountered the Python interpreter does not exit.
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows −

- File Object Methods: The *file* object provides functions to manipulate files.
- OS Object Methods: This provides methods to process files as well as directories.

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

- **Exception Handling:** This would be covered in this tutorial. Here is a list standard Exceptions available in Python: Standard Exceptions.
- **Assertions:** This would be covered in Assertions in Python

List of Standard Exceptions −

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix

- Interbase
- Oracle
- Sybase

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection

SOURCE CODE:

Settings.py

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'm+1edl5m-5@u9u!b8-=4-4mq&o1%agco2xpl8c!7sn7!eowjk#'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Remote_User',
    'Service_Provider',
]


MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]


ROOT_URLCONF = 'identifying_hot_topic_trends.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [(os.path.join(BASE_DIR,'Template/htmls'))],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'identifying_hot_topic_trends.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'identifying_hot_topic_trends',
```

```
        'USER':'root',
        'PASSWORD': '',
        'HOST' :'127.0.0.1',
        'PORT' :'3306',
    }
}



# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR,'Template/images')]
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'Template/media')

STATIC_ROOT = '/static/'
```

```
STATIC_URL = '/static/'
```

Views.py

```
from django.db.models import  Count, Avg
from django.shortcuts import render, redirect
from django.db.models import Count
from django.db.models import Q
import datetime
import xlwt
from django.http import HttpResponse
import numpy as np



import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.tree import DecisionTreeClassifier
#model selection
from sklearn.metrics import confusion_matrix, accuracy_score, plot_confusion_matrix, classification_report

# Create your views here.
from Remote_User.models import ClientRegister_Model,predict_hot_topic,detection_ratio,detection_accuracy



def serviceproviderlogin(request):
    if request.method  == "POST":
        admin = request.POST.get('username')
        password = request.POST.get('password')
        if admin == "Admin" and password =="Admin":
            detection_accuracy.objects.all().delete()
            return redirect('View_Remote_Users')
```

```
    return render(request,'SProvider/serviceproviderlogin.html')

def View_Predicted_Identifying_Hot_Topic_Trends_Ratio(request):
    detection_ratio.objects.all().delete()

    ratio = ""
    kword = 'Hot Topic'
    print(kword)
    obj = predict_hot_topic.objects.all().filter(Q(Prediction=kword))
    obj1 = predict_hot_topic.objects.all()
    count = obj.count();
    count1 = obj1.count();
    ratio = (count / count1) * 100
    if ratio != 0:
        detection_ratio.objects.create(names=kword, ratio=ratio)

    ratio1 = ""
    kword1 = 'Normal Topic'
    print(kword1)
    obj1 = predict_hot_topic.objects.all().filter(Q(Prediction=kword1))
    obj11 = predict_hot_topic.objects.all()
    count1 = obj1.count();
    count11 = obj11.count();
    ratio1 = (count1 / count11) * 100
    if ratio1 != 0:
        detection_ratio.objects.create(names=kword1, ratio=ratio1)


    obj = detection_ratio.objects.all()
    return render(request, 'SProvider/View_Predicted_Identifying_Hot_Topic_Trends_Ratio.html', {'objs': obj})

def View_Remote_Users(request):
    obj=ClientRegister_Model.objects.all()
    return render(request,'SProvider/View_Remote_Users.html',{'objects':obj})
```

```
def ViewTrendings(request):
    topic = predict_hot_topic.objects.values('topics').annotate(dcount=Count('topics')).order_by('-dcount')
    return  render(request,'SProvider/ViewTrendings.html',{'objects':topic})

def charts(request,chart_type):
    chart1 = detection_ratio.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts.html", {'form':chart1, 'chart_type':chart_type})

def charts1(request,chart_type):
    chart1 = detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts1.html", {'form':chart1, 'chart_type':chart_type})

def View_Predicted_Identifying_Hot_Topic_Trends(request):
    obj =predict_hot_topic.objects.all()
    return render(request, 'SProvider/View_Predicted_Identifying_Hot_Topic_Trends.html', {'list_objects': obj})

def likeschart(request,like_chart):
    charts =detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/likeschart.html", {'form':charts, 'like_chart':like_chart})



def Download_Trained_DataSets(request):

    response = HttpResponse(content_type='application/ms-excel')
    # decide file name
    response['Content-Disposition'] = 'attachment; filename="Predicted_Data.xls"'
    # creating workbook
    wb = xlwt.Workbook(encoding='utf-8')
    # adding sheet
    ws = wb.add_sheet("sheet1")
    # Sheet header, first row
    row_num = 0
    font_style = xlwt.XFStyle()
    # headers are bold
```

```
    font_style.font.bold = True
    # writer = csv.writer(response)
    obj = predict_hot_topic.objects.all()
    data = obj  # dummy method to fetch data.
    for my_row in data:
        row_num = row_num + 1
        ws.write(row_num, 0, my_row.Sno, font_style)
        ws.write(row_num, 1, my_row.PDate, font_style)
        ws.write(row_num, 2, my_row.Headline, font_style)
        ws.write(row_num, 3, my_row.Description, font_style)
        ws.write(row_num, 4, my_row.Source, font_style)
        ws.write(row_num, 5, my_row.Prediction, font_style)



    wb.save(response)
    return response

def train_model(request):
    detection_accuracy.objects.all().delete()
    data = pd.read_csv("Datasets.csv", encoding='latin-1')

    def apply_results(label):
        if (label == 0):
            return 0  # Normal Topic
        elif (label == 1):
            return 1  # Hot Topic

    data['Results'] = data['Label'].apply(apply_results)

    x = data['Description']
    y = data['Results']

    cv = CountVectorizer()

    print(x)
```

```
print("Y")
print(y)

x = cv.fit_transform(x)



models = []
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
X_train.shape, X_test.shape, y_train.shape

print("Naive Bayes")

from sklearn.naive_bayes import MultinomialNB

NB = MultinomialNB()
NB.fit(X_train, y_train)
predict_nb = NB.predict(X_test)
naivebayes = accuracy_score(y_test, predict_nb) * 100
print("ACCURACY")
print(naivebayes)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, predict_nb))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, predict_nb))
detection_accuracy.objects.create(names="Naive Bayes", ratio=naivebayes)



# SVM Model
print("SVM")
from sklearn import svm

lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
predict_svm = lin_clf.predict(X_test)
```

```
    svm_acc = accuracy_score(y_test, predict_svm) * 100
    print("ACCURACY")
    print(svm_acc)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, predict_svm))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, predict_svm))
    detection_accuracy.objects.create(names="SVM", ratio=svm_acc)

    print("Logistic Regression")

    from sklearn.linear_model import LogisticRegression

    reg = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
    y_pred = reg.predict(X_test)
    print("ACCURACY")
    print(accuracy_score(y_test, y_pred) * 100)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, y_pred))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, y_pred))
    detection_accuracy.objects.create(names="Logistic Regression", ratio=accuracy_score(y_test, y_pred) * 100)

    print("Decision Tree Classifier")
    dtc = DecisionTreeClassifier()
    dtc.fit(X_train, y_train)
    dtcpredict = dtc.predict(X_test)
    print("ACCURACY")
    print(accuracy_score(y_test, dtcpredict) * 100)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, dtcpredict))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, dtcpredict))
    detection_accuracy.objects.create(names="Decision Tree Classifier", ratio=accuracy_score(y_test, dtcpredict)
* 100)
```

```
    print("Gradient Boosting Classifier")
    from sklearn.ensemble import GradientBoostingClassifier
    clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0).fit(
        X_train,
        y_train)
    clfpredict = clf.predict(X_test)
    print("ACCURACY")
    print(accuracy_score(y_test, clfpredict) * 100)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, clfpredict))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, clfpredict))
    models.append(('GradientBoostingClassifier', clf))
    detection_accuracy.objects.create(names="GradientBoostingClassifier", ratio=accuracy_score(y_test,
clfpredict) * 100)

    labeled = 'Labeled_Data.csv'
    data.to_csv(labeled, index=False)
    data.to_markdown

    obj = detection_accuracy.objects.all()
    return render(request,'SProvider/train_model.html', {'objs': obj})
```

Remote User

Predict_Identifying_Hot_Topic_Trends.html

```
{% extends 'RUser/design.html' %}
{% block userblock %}


<link rel="icon" href="images/icon.png" type="image/x-icon" />

        <link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet">
                <link href="https://fonts.googleapis.com/css?family=Righteous" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Fredoka+One" rel="stylesheet">



                <style>
                        body {background-color:#000000;}
                        .container-fluid {padding:50px;}
                        .container{background-color:white;padding:50px;   }
                        #title{font-family: 'Fredoka One', cursive;
}
                        .text-uppercase{
                        font-family: 'Righteous', cursive;

                        }
                        .tweettext{

   border: 2px solid yellowgreen;
   width: 904px;
   height: 202px;
   overflow: scroll;
   background-color:;
}
```

```
                .style1 {
        color: #FF0000;
        font-weight: bold;
}
.style6 {
        font-size: 18px;
        color: #FFFF00;
        font-weight: bold;
}
.style8 {color: #FFFF00; font-weight: bold; }
    .style10 {font-size: 18px; color: #FF0000; font-weight: bold; }
    .style11 {color: #FFFF00}
    </style>

        <body>
        <div class="container-fluid">
                <div class="container">

                        <div class="row">
                                <div class="col-md-5">

            <form role="form" method="POST" >
                                                {% csrf_token %}
                                                <fieldset>
                                                        <p class="text-uppercase pull-center
style1">PREDICTION OF <span class="style10">IDENTIFYING HOT TOPIC  TYPE</span> !!! </p>
                    <hr>

                        {% csrf_token %}
                        <table width="998" align="center">
                          <tr>
                            <td height="44" bgcolor="#FF0000"><div align="center" class="style8">Enter Sno
</div></td>
                            <td><input name="Sno" type="text" value=""></td>
                            <td bgcolor="#FF0000"><div align="center"><span class="style11">Enter Post Date
```

```
</span></div></td>
                    <td><input type="text" name="PDate"></td>
                  </tr>
                  <tr>
                    <td height="44" bgcolor="#FF0000"><div align="center" class="style8">Enter
Headline</div></td>
                    <td><textarea name="Headline"></textarea></td>
                    <td bgcolor="#FF0000"><div align="center"><span class="style11">Enter
Description</span></div></td>
                    <td><textarea name="Description"></textarea></td>
                  </tr>
                  <tr>
                    <td width="289" height="44" bgcolor="#FF0000"><div align="center" class="style8">
                      <div align="center">Enter Source</div>
                    </div></td>
                    <td width="226"><input type="text" name="Source"></td>
                    <td width="314" bgcolor="#FF0000"> </td>
                    <td width="273"> </td>
                  </tr>
                  <tr>
                    <td height="44" bgcolor="#FFFFFF"><div align="center" class="style8">
                      <div align="center"></div>
                    </div></td>
                    <td> </td>
                    <td bgcolor="#FFFFFF"><input name="submit" type="submit" class="style1"
value="Predict"></td>
                    <td> </td>
                  </tr>
                </table>
                                    </fieldset>
                                </form>


                                    <form role="form" method="POST" >
                                        {% csrf_token %}
```

<fieldset>

<hr>

<table width="549" height="52" border="0" align="center" >
<tr><td width="308" bgcolor="#FF0000"> PREDICTED HOT TOPIC TYPE :: </td>

<td width="163" bgcolor="#FFFFFF" style="color:red; font-size:20px; font-family:fantasy" > {{objs}} </td></tr>
</table>

</fieldset>
</form>

<!-------null------>

{% endblock %}
<tr>

CHAPTER-10

SYSTEM TESTING

What do you mean by software testing?

Testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should. It is oriented to 'detection'.

Unit Testing:

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually. This testing mode is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

Unit tests are written from a programmer's perspective. They ensure that a particular method of a class successfully performs a set of specific tasks. Each test confirms that a method produces the expected output when given a known input.

Performance Testing:

Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions. Qualitative attributes such as

Reliability, scalability and interoperability may also be evaluated. Performance testing is often done in conjunction with stress testing.

Performance testing can verify that a system meets the specifications claimed by its manufacturer or vendor. The process can compare two or more devices or programs in terms of parameters such as speed, data transfer rate, bandwidth, throughput, efficiency or reliability.

Performance testing can also be used as a diagnostic aid in locating communications bottlenecks. Often a system will work much better if a problem is resolved at a single point or in a single component. For example, even the fastest computer will function poorly on today's Web if the connection occurs at only 40 to 50 Kbps (kilobits per second).

Integration Testing:

Integration testing, also known as integration and testing (I&T), is a software development process which program units are combined and tested as groups in multiple ways. In this context, a unit is defined as the smallest testable part of an application. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution. Integration testing is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

Test cases:

Test case for Login form:

FUNCTION:	**LOGIN**
EXPECTED RESULTS:	Should Validate the user and check his existence in database
ACTUAL RESULTS:	Validate the user and checking the user against the database
LOW PRIORITY	**No**
HIGH PRIORITY	**Yes**

Test case2:

Test case for Remote Access User Registration form:

FUNCTION:	**USER REGISTRATION**
EXPECTED RESULTS:	Should check if all the fields are filled by the user and saving the user to database.
ACTUAL RESULTS:	Checking whether all the fields are field by user or not through validations and saving user.
LOW PRIORITY	**No**
HIGH PRIORITY	**Yes**

Test case3:

Test case for Change Password:

When the old password does not match with the new password ,then this results in displayingan error message as " OLD PASSWORD DOES NOT MATCH WITH THE NEW PASSWORD".

FUNCTION:	**Change Password**

EXPECTED RESULTS:	Should check if old password and new password fields are filled by the user and saving the user to database.
ACTUAL RESULTS:	Checking whether all the fields are field by user or not through validations and saving user.
LOW PRIORITY	**No**
HIGH PRIORITY	**Yes**

Test case 4:

Test case for Forget Password:

When a user forgets his password he is asked to enter Login name, ZIP code, Mobile number. If these are matched with the already stored ones then user will get his Original password.

Module	Functionality	Test Case	Expected Results	Actual Results	Result	Priority
er	Login Usecase	Navigate To Www.Sample.Com Click On Submit Button Without Entering Username and Password	A Validation Should Be As Below “Please Enter Valid Username & Password”	A Validation Has Been Populated As Expected	Pass	High

		aNavigate To Www.Sample.Com Click On Submit Button With Out Filling Password And With Valid Username Test UsernameField	A Validation Should Be As Below “Please Enter Valid Password Or Password Field Can Not Be Empty “	A Validation Is Shown As Expected	Pass	High
		NNavigate To Www.Sample.Com Enter Both Username And	A Validation Shown As Below “The Username Entered Is Wrong”	A Validation Is Shown As Expected	Pass	High

		Password WrongAnd Hit Enter				
		Navigate To Www.Sample.Com Enter Validate Username And Password And Click On Submit	Validate Username And Password In DataBase And Once If They Correct Then Show The Main Page	Main Page/ Home Page Has Been Displayed	Pass	High

SCREENSHOTS

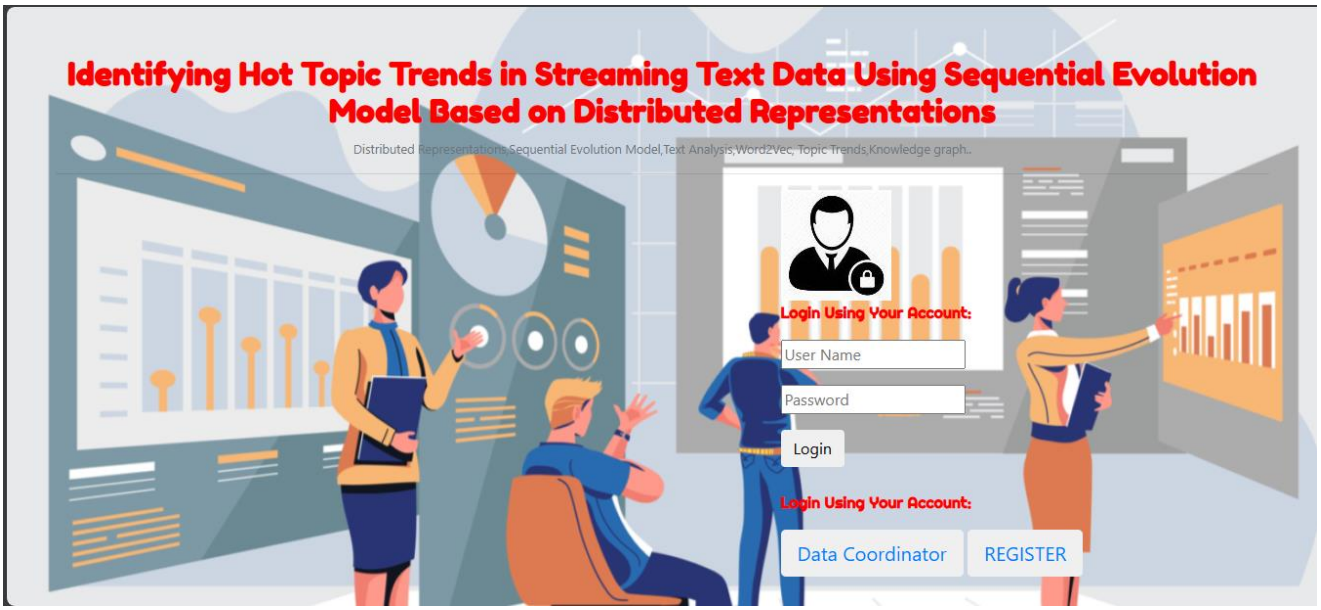


Fig 1: Distant User Login

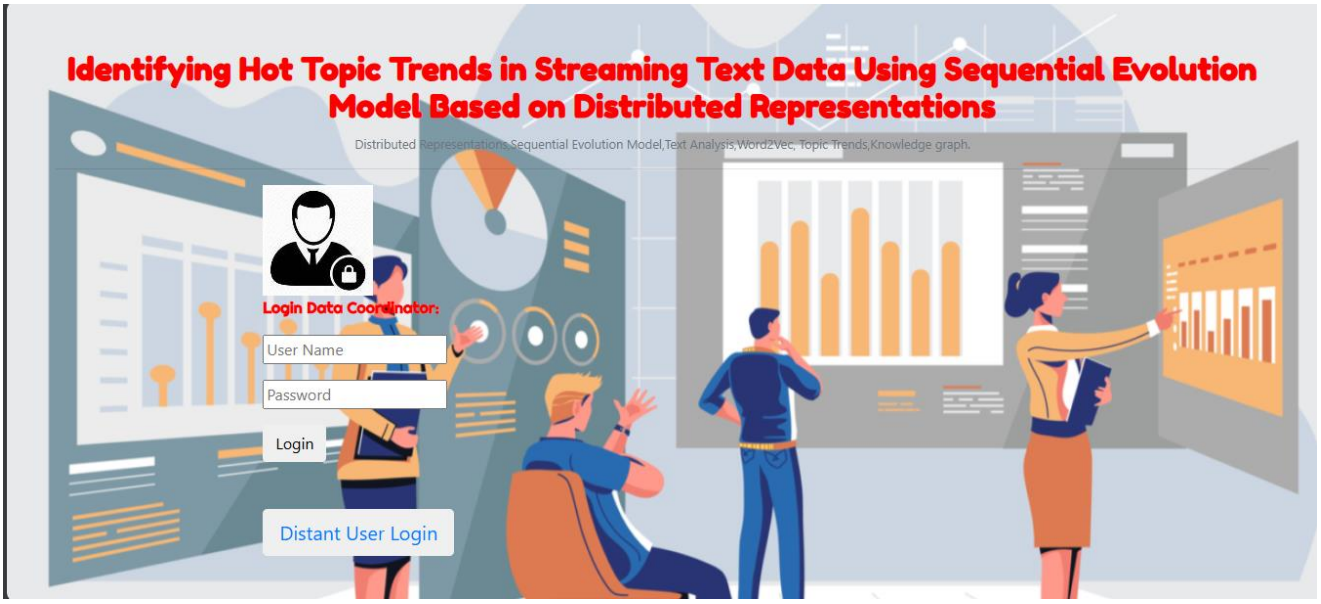


Fig 2: Data Coordinator Login

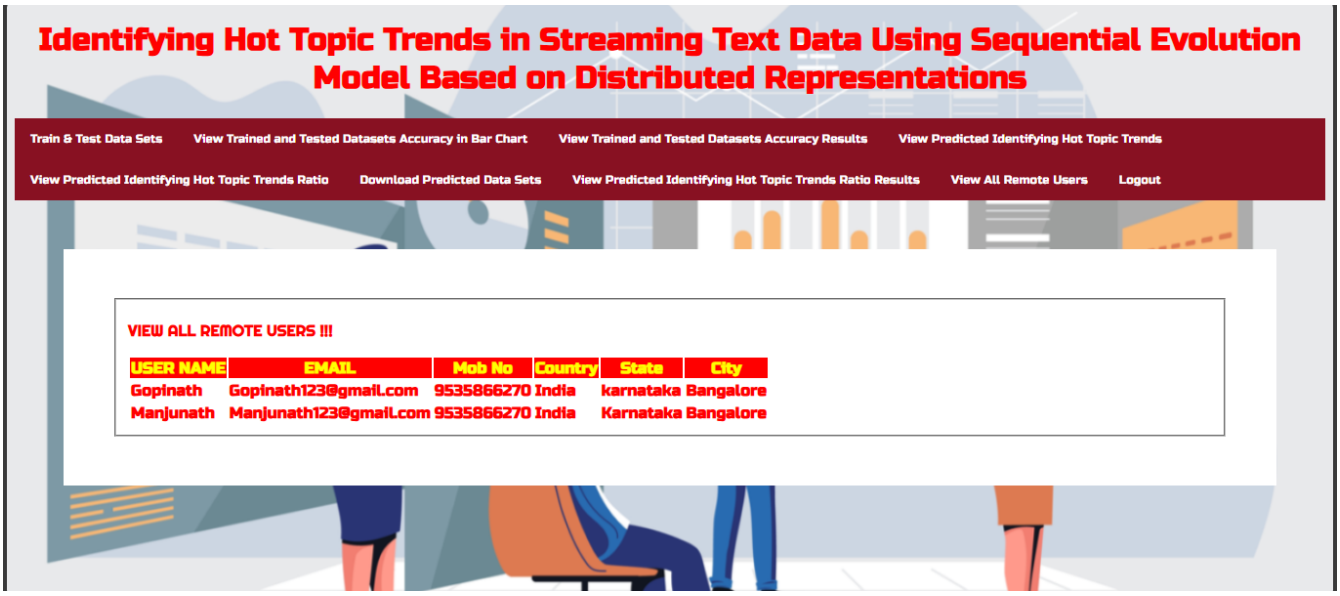


Fig 3: Data Coordinator Home Page

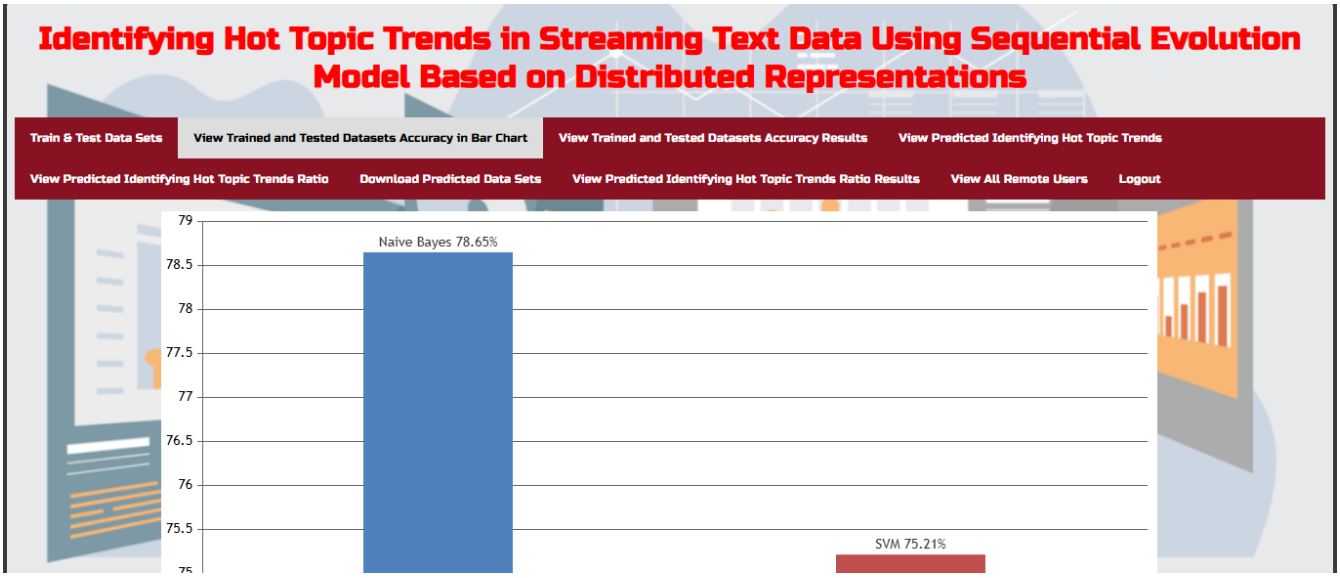


Fig 4: View Trained and Tested Dataset Accuracy in Bar Chart

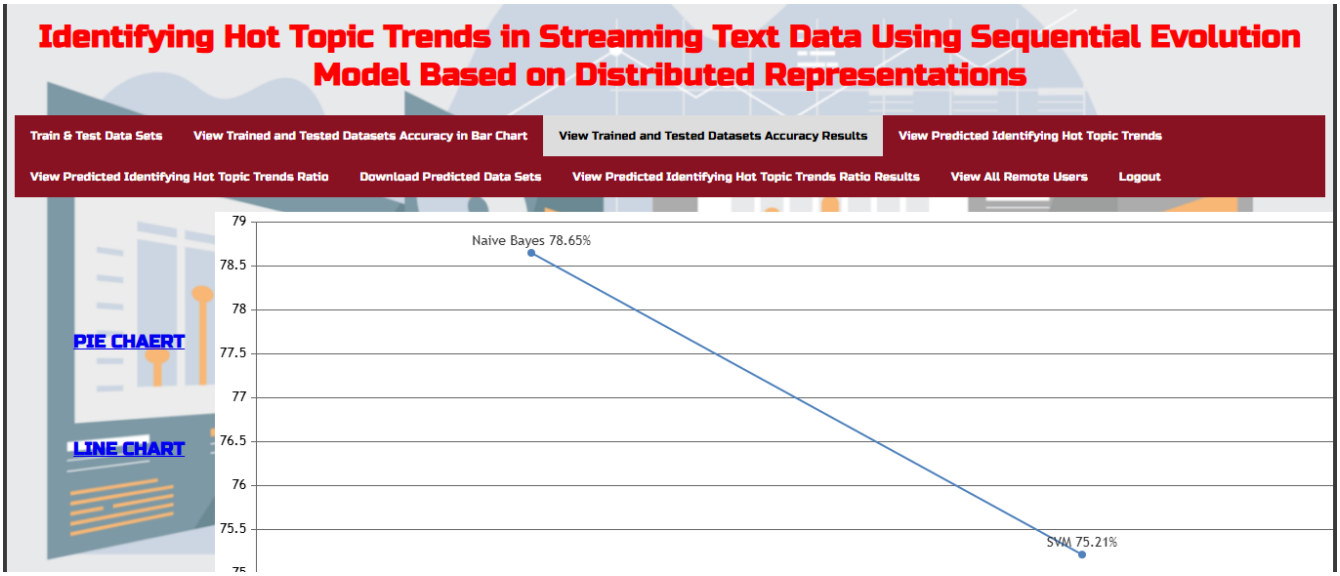


Fig 5: View Trained and Tested Dataset Accuracy Results

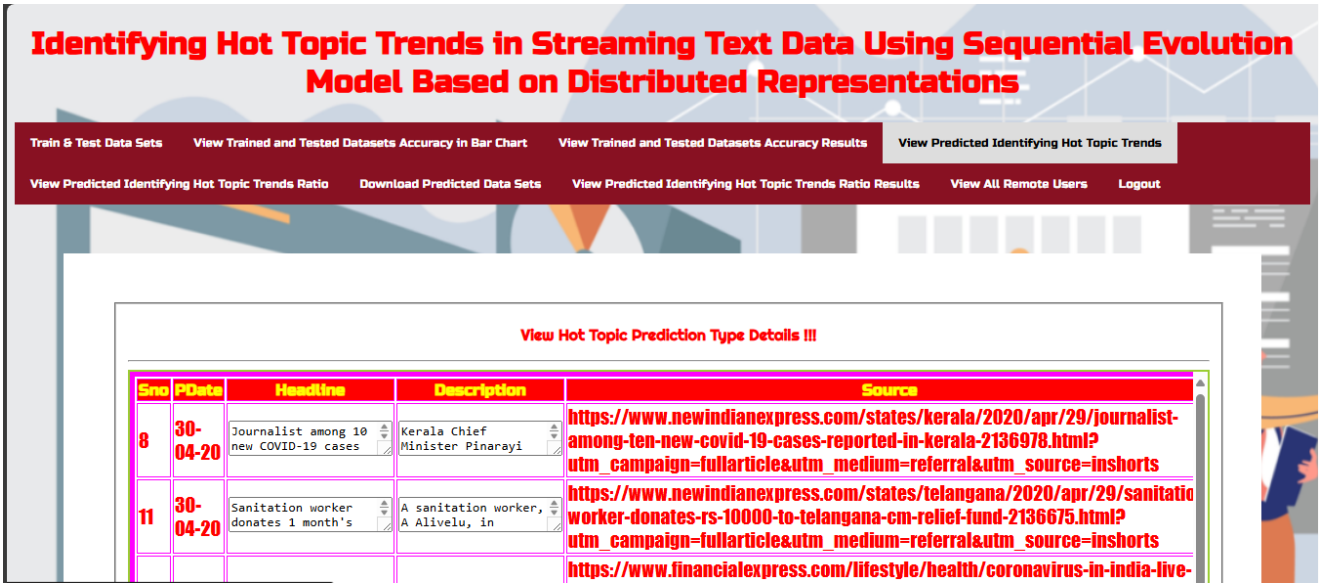


Fig 6: View Predicted Identifying Hot Topic Trends

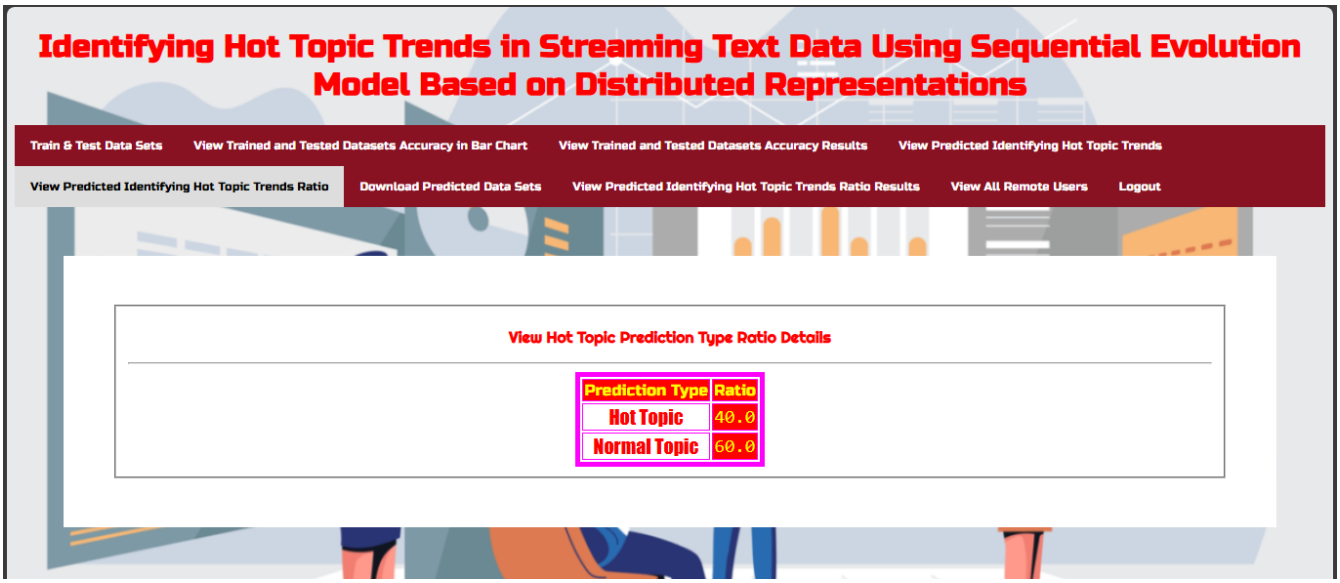


Fig 7: View Predicted Identifying Hot Topic Trends Ratio

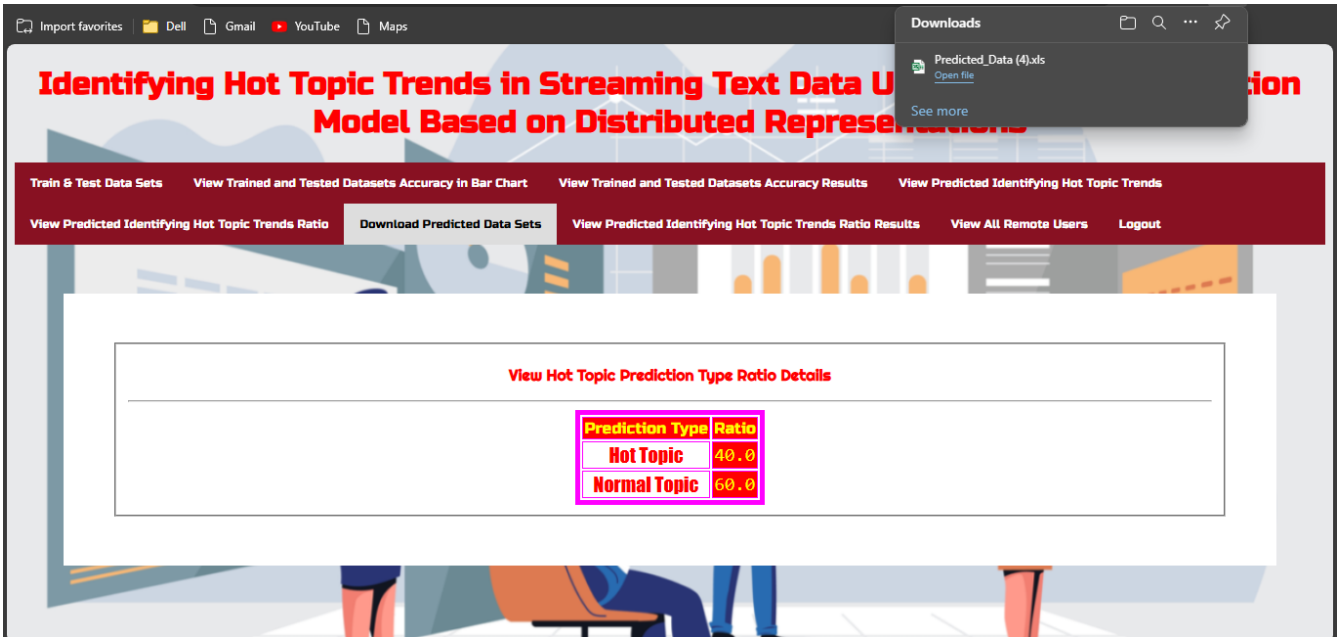


Fig 8: Download Predicted Dataset

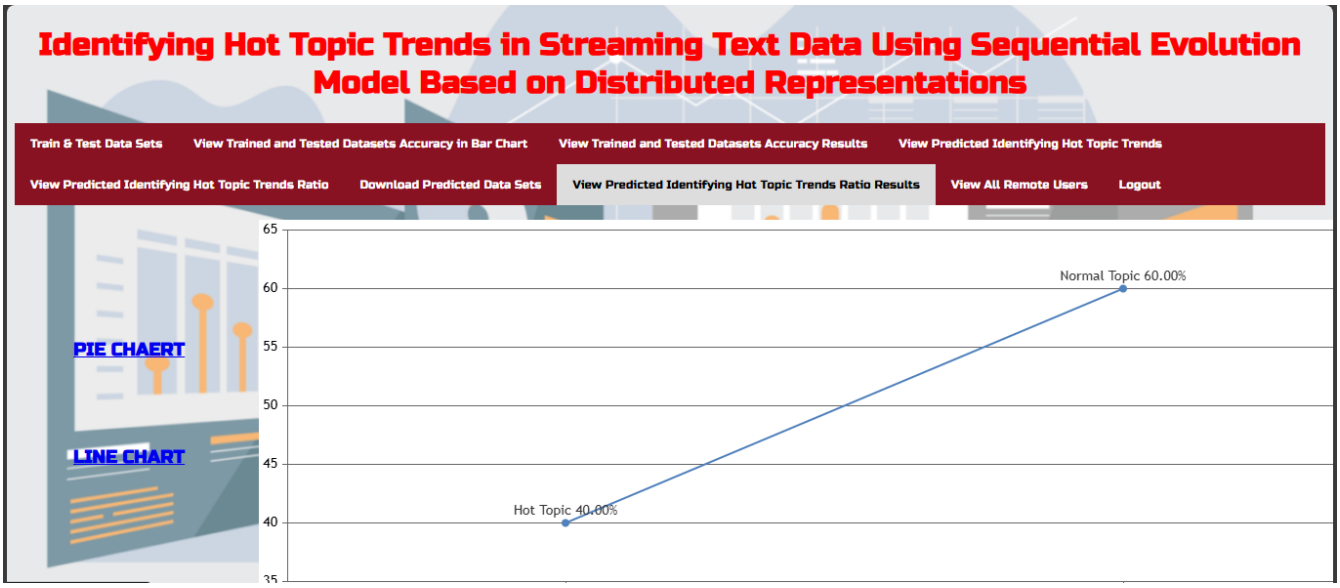


Fig 9: View Predicted Identifying Hot Topic Trends Ratio Results

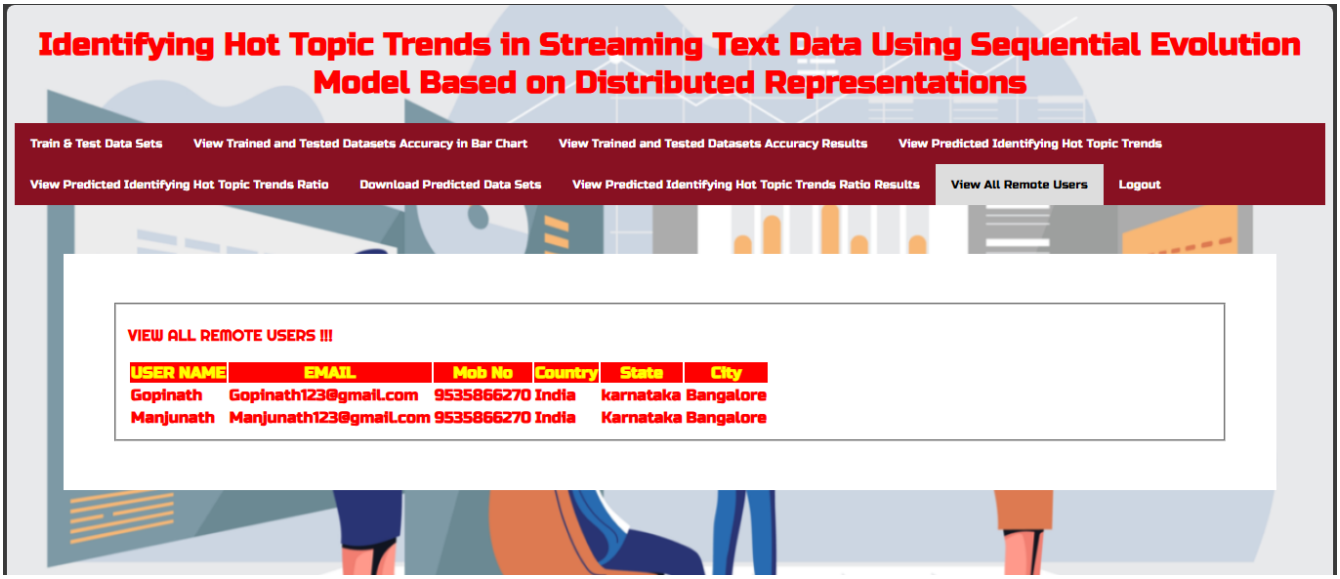


Fig 10: View all remote Users

CHAPTER-11

CONCLUSION

This research has demonstrated how to accurately detect and identify topic trends and analyze hotspot relationships intelligently using distributed representations. The proposed model displayed a linear structure that enables precise semantic relations. Secondly, we chronologically analyzed the differences in sequential periods of different datasets, making it capable of analyzing the relationships between words and taking into account how those relationships changed over different time periods. We successfully built several separate word2vec models for each dataset, instead of only one model. We also thank NSEM for its great improvement in the quality of our work, allowing us to find the evolutionary process of correlations in each model. NSEM resulted in faster training and significantly better representations of uncommon datasets. For the first time, we explain the insides of NSEM, which are crucial in analyzing the difference between sequential periods. Additionally, we constructed a visual display model for common words, making the method general and applicable to any category another vital outcome of this work is constructing a knowledge graph of topic trends using semantic similarity through word2vec. The presented tool box plays a pivotal role in providing valuable insights, facilitating data-driven decision-making, fostering research advancements, enhancing educational experiences, and empowering the general public with comprehensive and user-friendly information about current topics, making our research indispensable with multifaceted applications. Therefore, this work can be viewed as complementary to existing methods.

.

FUTURE SCOPE

The future scope of this research is expansive and multifaceted, promising significant advancements across various domains. One potential direction is the enhancement of the proposed model's accuracy and efficiency by incorporating more advanced machine learning techniques, such as deep learning and transformer-based models. Additionally, expanding the model's capability to handle larger and more diverse datasets could provide even deeper insights into topic trends and relationships. Further research could explore the integration of other natural language processing (NLP) techniques to improve semantic understanding and trend prediction. There is also considerable potential in applying this model to real-time data analysis, enabling dynamic tracking of evolving trends and hotspots.

Moreover, developing user-friendly interfaces and visualization tools could make the model more accessible to non-experts, facilitating broader applications in education, business intelligence, and public policy. Collaborative efforts with other fields, such as social sciences and economics, could enrich the model's applicability and impact. Lastly, ongoing refinement and validation of the NSEM's methodology could lead to more robust and generalizable findings, making this research an indispensable tool for various stakeholders seeking data-driven insights and strategic decision-making capabilities.

CHAPTER-12

REFERENCES

1. Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems 26 (2013).

2. Mikolov, Tomáš, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." Proceedings of the 2013 conference of the north American chapter of the association for computational linguistics: Human language technologies. 2013.

3. Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

4. Khan, Zohaib Ahmad, et al. "Streaming news sequential evolution model based on distributed representations." 2017 36th Chinese Control Conference (CCC). IEEE, 2017.

5. Di Gennaro, Giovanni, Amedeo Buonanno, and Francesco AN Palmieri. "Considerations about learning Word2Vec." The Journal of Supercomputing (2021): 1-16.

6. Raja, Rohit, et al. "Analysis of anomaly detection in surveillance video: recent trends and future vision." Multimedia Tools and Applications 82.8 (2023): 12635-12651.

7. Gurcan, Fatih, et al. "Detecting latent topics and trends in software engineering research since 1980 using probabilistic topic modeling." IEEE Access 10 (2022): 74638-74654.

8. Hsu, Ying-Shao, Kai-Yu Tang, and Tzu-Chiang Lin. "Trends and Hot Topics of STEM and STEM Education: a Co-word Analysis of Literature Published in 2011–2020." Science & Education (2023): 1-24.

9. Ozyurt, Ozcan, and Hacer Ozyurt. "A large-scale study based on topic modeling to determine the research interests and trends on computational thinking." Education and Information Technologies 28.3 (2023): 3557-3579.

10. Kim, Y., M. Kim, and Hwankuk Kim. "Detecting IoT Botnet in 5G Core Network Using Machine Learning." CMC. Vol. 72. 2022.

11. Bao, Jie, et al. "Exploring topics and trends in Chinese ATC incident reports using a domain-knowledge driven topic model." Journal of Air Transport Management 108 (2023): 102374. 12. Kowsher, M., et al. "Bangla-BERT: transformer-based efficient model for transfer learning and language understanding." IEEE Access 10 (2022): 91855-91870.

13. Khan, Javed Ali, et al. "Analysis of requirements-related

arguments in user forums." 2019 IEEE 27th International Requirements Engineering Conference (RE). IEEE, 2019.
14. Blei, David M., and John D. Lafferty. "Dynamic topic models." Proceedings of the 23rd international conference on Machine learning. 2006.
15. Adjuik, Toby A., and Daniel Ananey-Obiri. "Word2vec neural model-based technique to generate protein vectors for combating COVID-19: a machine learning approach." International Journal of Information Technology 14.7 (2022): 3291-3299.
16. Winatmoko, Yosef Ardhito, and Masayu Leylia Khodra. "Automatic summarization of tweets in providing Indonesian trending topic explanation." Procedia Technology 11 (2013): 1027-1033.
17. Gurcan, Fatih, et al. "Evolution of software testing strategies and trends: Semantic content analysis of software research corpus of the last 40 years." IEEE Access 10 (2022): 106093-106109.
18. Suryadi, Dedy, Hanky Fransiscus, and Yoko Gunawan Chandra. "Analysis of Topic and Sentiment Trends in Customer Reviews Before and After Covid-19 Pandemic." 2022 International Visualization, Informatics and Technology Conference (IVIT). IEEE, 2022.
19. Behpour, Sahar, et al. "Automatic trend detection: Time-biased document clustering." Knowledge-Based Systems 220 (2021): 106907.
20. Lv, Fengmao, et al. "Latent Gaussian process for anomaly detection in categorical data." Knowledge-Based Systems 220 (2021): 106896.
21. Azizi, F., Hajiabadi, H., Vahdat-Nejad, H., & Khosravi, M. H. (2023). Detecting and analyzing topics of massive COVID-19 related tweets for various countries. Computers and Electrical Engineering, 106, 108561.
22. Yukselen, Murat, Alev Mutlu, and Pinar Karagoz. "Predicting the Trending Research Topics by Deep Neural Network Based Content Analysis." IEEE Access 10 (2022): 90887-90902.
23. Cavalli, Stefano, and Michele Amoretti. "CNN-based multivariate data analysis for bitcoin trend prediction." Applied Soft Computing 101 (2021): 107065.
24. Guo, Zhiwei, et al. "Fuzzy detection system for rumors through explainable adaptive learning." IEEE Transactions on Fuzzy Systems 29.12 (2021): 3650-3664.
25. Ravi, Vinayakumar, Tuan D. Pham, and Mamoun Alazab. "Attention-based multidimensional deep learning approach for cross-architecture IoMT malware detection and classification in healthcare cyber-physical systems." IEEE Transactions on Computational Social Syste