VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA", MACHHE, BELAGAVI-590018

Project Report
on

"Fake News Detection Using Machine Learning Classifiers"

Submitted in partial fulfillment of the requirements for the VIII semester
Bachelor of Engineering
in
Information Science and Engineering
of
Visvesvaraya Technological University, Belagavi
by

CHITRA S M (1CD17IS008)
NANDINI S (1CD17IS044)
SURYA K S (1CD17IS055)

Under the Guidance of
Prof. Bharani B R
Asst. Professor
Dept. of ISE

Department of Information Science and Engineering
CAMBRIDGE INSTITUTE OF TECHNOLOGY, BANGALORE-560 036

2020-2021

CAMBRIDGE INSTITUTE OF TECHNOLOGY

K.R. Puram, Bangalore-560 036

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

Certified that **Ms. Chitra S M** (**1CD17IS008), Ms. Nandini S** (**1CD17IS044), Mr. Surya K S** (**1CD17IS055)** a bonafide student of **Cambridge Institute of Technology,** has successfully completed the Project entitled "**Fake News Detection using Machine Learning Classifiers**" in partial fulfillment of the requirements for VIII semester **Bachelor of Engineering** in **Information Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during academic year 2020-2021. It is certified that all Corrections/Suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The Project report has been approved as it satisfies the academic requirements prescribed for the Bachelor of Engineering degree.

Project Guide, **Prof. Bharani B.R** **Dept. of ISE, CITech**	**Head of the Department,** **Dr. K Satyanarayan Reddy,** **Dept. of ISE, CITech**	**PRINCIPAL** **Dr. Suresh .L** **CiTech**

Name of the Examiners **Signature**

1.

2.

DECLARATION

I**, Chitra S M, Nandini S, Surya K S** of VIII semester BE, Information Science and Engineering, Cambridge Institute of Technology, hereby declare that the Project entitled ***"Fake News Detection Using Machine Learning Classifiers"*** has been carried out by me and submitted in partial fulfillment of the course requirements of VIII semester **Bachelor of Engineering** in **Information Science and Engineering** as prescribed b**y Visvesvaraya Technological University, Belagavi**, during the academic year 2020-2021.

I also declare that, to the best of my knowledge and belief, the work reported here does not form part of any other report on the basis of which a degree or award was conferred on an earlier occasion on this by any other student.

Chitra S M (**1CD17IS008)**

Nandini S (**1CD17IS0044)**

Surya K S (**1CD17IS0055)**

Date:

Place: Bangalore

ACKNOWLEDGEMENT

I would like to place on record my deep sense of gratitude to **Shri. D. K. Mohan,** Chairman, Cambridge Group of Institutions, Bangalore, India for providing excellent Infrastructure and Academic Environment at CITech without which this work would not have been possible**.**

I am extremely thankful to **Dr. Suresh L,** Principal, CITech, Bangalore, for providing me the academic ambience and everlasting motivation to carry out this work and shaping our careers.

I express my sincere gratitude to **Dr. K Satyanarayan Reddy,** Professor & HOD, Dept. of Information Science and Engineering, CITech, Bangalore, for his stimulating guidance, continuous encouragement and motivation throughout the course of present work.

I also wish to extend my thanks to Project Coordinator, **Prof. Vinayaka S P**, Assistant Professor, Dept. of ISE, CITech, Bangalore for the critical, insightful comments, guidance and constructive suggestions to improve the quality of this work.

I also wish to extend my thanks to Project guide, **Prof. Bharani B R,** Asst. Professor, Dept. of ISE, CITech for her guidance and impressive technical suggestions to complete my Project work.

Finally, to all my friends, classmates who always stood by me in difficult situations also helped me in some technical aspects and last but not the least, I wish to express deepest sense of gratitude to my parents who were a constant source of encouragement and stood by me as pillar of strength for completing this work successfully.

Chitra S M

Nandini S

Surya K S

ABSTRACT

Information sharing on the web particularly via web-based networking media is increasing. Ability to identify, evaluate and address such information is significantly important. Fake news is a phenomenon which is having a significant impact on our social life, in particular in the political world. Fake news detection is an emerging research area which is gaining interest but involved some challenges due to the limited number of resources (i.e., datasets, published literature) available. A fake news detection model is proposed that use machine learning techniques. It is investigated and compared with three different evaluation models namely Count Vectorizer, Tfidf Vectorizer and ngram and four machine learning techniques were used namely Naive Bayes, Random Forest, Logistic Regression, SVM. Experimental evaluation yields the best performance using Term Frequency-Inverted Document Frequency (TF-IDF) as feature extraction technique, and Logistic regression.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

In the recent years, online content has been playing a significant role in swaying users decisions and opinions. Opinions such as online reviews are the main source of information for e-commerce customers to help with gaining insight into the products they are planning to buy. Recently it has become apparent that opinion spam does not only exist in product reviews and customers' feedback. In fact, fake news and misleading articles is another form of opinion spam, which has gained traction. Some of the biggest sources of spreading fake news or rumors are social media websites such as Google Plus,

Even though the problem of fake news is not a new issue, detecting fake news is believed to be a complex task given that humans tend to believe misleading information and the lack of control of the spread of fake content. Fake news has been getting more attention in the last couple of years, especially since the US election in 2016. It is tough for humans to detect fake news. It can be argued that the only way for a person to manually identify fake news is to have a vast knowledge of the covered topic. Even with the knowledge, it is considerably hard to successfully identify if the information in the article is real or fake. The open nature of the web and social media in addition to the recent advance in computer science simplify the process of creating and spreading fake news.

In general, Fake news could be categorized into three groups. The first group is fake news, which is news that is completely fake and is made up by the writers of the articles.

- The second group is fake satire news, which is fake news whose main purpose is to provide humor to the readers.
- The third group is poorly written news articles, which have some degree of real news, but they are not entirely accurate.
- In short, it is news that uses, for example, quotes from political figures to report a fully fake story.
- Usually, this kind of news is designed to promote certain agenda or biased opinion.

Fake news is a phenomenon which is having a significant impact on our social life, in particular in the political world. Fake news detection is an emerging research area which is gaining interest but involved some challenges due to the limited amount of resources (i.e., datasets, published literature) available. A fake news detection model is proposed that use machine learning techniques. It is investigated and compared with three different evaluation models namely CountVectorizer, Tfidf Vectorizer and ngram and four machine learning techniques were used namely NaiveBayes, Random Forest, Logistic Regression, SVM.

1.1 PROBLEM STATEMENT

Detecting fake news is believed to be a complex task and much harder than detecting fake product reviews given that they spread easily using social media and word of mouth.

While it is easier to understand and trace the intention and the impact of fake reviews, the intention, and the impact of creating propaganda by spreading fake news cannot be measured or understood easily. For instance, it is clear that fake review affects the product owner, customer and online stores; on the other hand, it is not easy to identify the entities affected by the fake news. This is because identifying these entities require measuring the news propagation, which has shown to be complex and resource intensive. Trend Micro, a cyber security company, analyzed hundreds of fake news services provider around the globe. They reported that it is effortless to purchase one of those services. In fact, according to the report, it is much cheaper for politicians and political parties to use those services to manipulate election outcomes and people opinions about certain topics.

1.2 OBJECTIVES

- The first step involves the creation of dataset we are using the web scraping techniques to scrap the headlines and its contents .
- We are applying the three vertorized techniques to preprocess and extract the collected data .

- Then extracted we are applying the machine learning algorithms to classify the news is news is original or fake.

1.3 SCOPE

The proposed framework helps to identify good quality news. It can also be used in media industry as well as social media. The proposed framework can also be utilized to prevent spreading of fake news in social media. However, the proposed framework is implemented to detect the news is real or fake.

CHAPTER 2

LITERATURE SURVEY

Deception detection for news: three types of fakes

Rubin, V.L., Chen, Y., Conroy, N.J, Proceedings of the 78th ASIS&T Annual Meeting: Information Science with Impact: Research in and for the Community (ASIST 2015). Article 83, p. 4, American Society for Information Science, Silver Springs (2015)

A fake news detection system aims to assist users in detecting and filtering out varieties of potentially deceptive news. The prediction of the chances that a particular news item is intentionally deceptive is based on the analysis of previously seen truthful and deceptive news. A scarcity of deceptive news, available as corpora for predictive modeling, is a major stumbling block in this field of natural language processing (NLP) and deception detection. This paper discusses three types of fake news, each in contrast to genuine serious reporting, and weighs their pros and cons as a corpus for text analytics and predictive modeling. Filtering, vetting, and verifying online information continues to be essential in library and information science (LIS), as the lines between traditional news and online information are blurring

Fake News Detection on Social Media: A Data Mining Perspective

Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017), ACM SIGKDD Explorations Newsletter, 19(1), 22-36.

Social media for news consumption is a double-edged sword. On the one hand, its low cost, easy access, and rapid dissemination of information lead people to seek out and consume news from social media. On the other hand, it enables the wide spread of \fake news", i.e., low quality news with intentionally false information. The extensive spread of fake news has the potential for extremely negative impacts on individuals and society. Therefore, fake news detection on social media has recently become an emerging research that is attracting tremendous attention. Fake news detection on social media presents unique characteristics and challenges that make existing detection algorithms from traditional news media inductive or not applicable. First, fake news is intentionally written to mislead readers to believe false information, which makes it difficult and

nontrivial to detect based on news content; therefore, we need to include auxiliary information, such as user social engagements on social media, to help make a determination. Second, exploiting this auxiliary information is challenging in and of itself as users' social engagements with fake news produce data that is big, incomplete, unstructured, and noisy. Because the issue of fake news detection on social media is both challenging and relevant, we conducted this survey to further facilitate research on the problem. In this survey, we present a comprehensive review of detecting fake news on social media, including fake news characterizations on psychology and social theories, existing algorithms from a data mining perspective, evaluation metrics and representative datasets. We also discuss related research areas, open problems, and future research directions for fake news detection on social media.

Fake News or Truth? Using Satirical Cues to Detect Potentially Misleading News

Rubin, V., Conroy, N., Chen, Y., & Cornwell, S. Proceedings of the Second Workshop on Computational Approaches to Deception Detection, 2016

Satire is an attractive subject in deception detection research: it is a type of deception that intentionally incorporates cues revealing its own deceptiveness. Whereas other types of fabrications aim to instill a false sense of truth in the reader, a successful satirical hoax must eventually be exposed as a jest. This paper provides a conceptual overview of satire and humor, elaborating and illustrating the unique features of satirical news, which mimics the format and style of journalistic reporting. Satirical news stories were carefully matched and examined in contrast with their legitimate news counterparts in 12 contemporary news topics in 4 domains (civics, science, business, and “soft” news). Building on previous work in satire detection, we proposed an SVM-based algorithm, enriched with 5 predictive features (Absurdity, Humor, Grammar, Negative Affect, and Punctuation) and tested their combinations on 360 news articles. Our best predicting feature combination (Absurdity, Grammar and Punctuation) detects satirical news with a 90% precision and 84% recall (F-score=87%). This work in algorithmically identifying satirical news pieces can aid in minimizing the potential deceptive impact of satire.

Automatic deception detection: Methods for finding fake news

Conroy, N., Rubin, V., & Chen, Y, Proceedings of the Association for Information Science and Technology, 52(1), 1-4, 2015

This research surveys the current state-of-the-art technologies that are instrumental in the adoption and development of fake news detection. " Fake news detection " is defined as the task of categorizing news along a continuum of veracity, with an associated measure of certainty. Veracity is compromised by the occurrence of intentional deceptions. The nature of online news publication has changed, such that traditional fact checking and vetting from potential deception is impossible against the flood arising from content generators, as well as various formats and genres. The paper provides a typology of several varieties of veracity assessment methods emerging from two major categories – linguistic cue approaches (with machine learning), and network analysis approaches. We see promise in an innovative hybrid approach that combines linguistic cue and machine learning, with network-based behavioral data. Although designing a fake news detector is not a straightforward problem, we propose operational guidelines for a feasible fake news detecting system.

To summarize, the existing systems fails to completely address the problems of identification real news and fake news. In order to overcome this the proposed framework uses three vectorize model along with the machine learning algorithms.

2.1 EXISTING SYSTEM

Research on fake news detection is still at an early stage, as this is a relatively recent phenomenon, at least regarding the interest raised by society.

Rubin et al. discuss three types of fake news. Each is a representation of inaccurate or deceptive reporting. Furthermore, the authors weigh the different kinds of fake news and the pros and cons of using different text analytics and predictive modeling methods in detecting them. In this paper, they separated the fake news types into three groups:

– Serious fabrications are news not published in mainstream or participant media, yellow press or tabloids, which as such, will be harder to collect.

– Large-Scale hoaxes are creative and unique and often appear on multiple platforms. The authors argued that it may require methods beyond text analytics to detect this type of fake news.

– Humorous fake news, are intended by their writers to be entertaining, mocking, and even absurd. According to the authors, the nature of the style of this type of fake news could have an adverse effect on the effectiveness of text classification techniques.

The authors argued that the latest advance in natural language processing (NLP) and deception detection could be helpful in detecting deceptive news. However, the lack of available corpora for predictive modeling is an important limiting factor in designing effective models to detect fake news.

Horne et al. illustrated how obvious it is to distinguish between fake and honest articles. According to their observations, fake news titles have fewer stop-words and nouns, while having more nouns and verbs. They extracted different features grouped into three categories as follows:

– Complexity features calculate the complexity and readability of the text.

– Psychology features illustrate and measure the cognitive process and personal concerns underlying the writings, such as the number of emotion words and casual words.

– Stylistic features reflect the style of the writers and syntax of the text, such as the number of verbs and the number of nouns.

The aforementioned features were used to build an SVM classification model.

The authors used a dataset consisting of real news from BuzzFeed and other news websites, and Burfoot and Baldwin's satire dataset to test their model. When they compared real news against satire articles (humorous article), they achieved 91% accuracy.

Wang et al. introduced LIAR, a new dataset that can be used for automatic fake news detection. Thought LIAR is considerably bigger in size, unlike other data sets, this data set does not contain full articles, it contains 12800 manually labeled short statements from politicalFact.com.

Rubin et al. proposed a model to identify satire and humor news articles. They examined and inspected 360 Satirical news articles in mainly four domains, namely, civics, science, business, and what they called "soft news". They proposed an SVM classification model using mainly five features developed based on their analysis of the satirical news.

2.2 Disadvantages

The accuracy dropped to 71% when predicting fake news against real news.

2.3 PROPOSED SYSTEM

- Machine learning technique is used to detect fake news, which consists of using text analysis based on classification techniques.
- Two different supervised classification techniques, namely, Support Vector Machine (SVM), Logistic Regression (LR) are used.
- Experimental evaluation is conducted using a dataset compiled from real and fake news websites, yielding very encouraging results.

2.4 Advantages

- High accuracy on detection
- Fake news can be detected using machine learning techniques
- Real time data is extracted using web scrapping technique.

CHAPTER 3

REQUIREMENT SPECIFICATION AND ANALYSIS

System Requirement specification (SRS) is a complete description of the behaviour of the system that is to be developed. It is a structured collection of information that embodies the requirements of the system. System design is a process by which the software requirements and hardware requirements are translated into representation of system components, interface and data necessary for the implementation phase.

The system requirement specification is documented in such a way that it breaks the deliverables into smaller components. The information is organized in such a way that the developers will not only understand the boundaries within which they need to work, but also what functionally needs to be developed and in what order. In principle, the system requirements specification of a system should be both complete and consistent. Completeness means that all services required by the user should be defined. Consistency means that requirement should not have contradictory definitions.

3.1 TYPES OF SYSTEM REQUIREMENTS SPECIFICATIONS

System requirements are often classified as functional requirements, non-functional requirements, hardware requirements and software requirements.

3.1.1 FUNCTIONAL REQUIREMENTS

These are statement of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some case the functional requirements may also explicitly state what the system should not do. The functional requirements for a system describe should do.

These requirements depend on the type of system being developed, the expected users of the system and the general approach taken by the organization when writing requirements. However, functional system requirements describe the system function in detail, its input and outputs, exceptions, and so on. For a system may be expressed in a number of ways.

These functional user requirements define specific facilities to be provided by the system. These have been taken from the user requirements document and they illustrate that functional requirements may be written at different levels of details.

This section describes the functional requirements of the system for those requirements which are expressed in the natural language style.

1. Create an web application.
2. Load the dataset as input.
3. System should preprocess and extract the features from the dataset using TF-IDF/n-gram.
4. Applying the Logistic regression algorithm System should classify the news is fake or original.
5. Application should provides high accuracy of classification of news dataset.

Product Requirements:

These requirements specify product behaviour. It consumes less memory, less system resource and low failure rates.

3.1.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements, as the name suggests, are requirements that are not directly concerned with the specific function delivered by the system. They may relate to emergent system properties such as reliability, response time and store occupancy. Alternatively, they may define constraints on the system such as the capabilities of input output devices and the data representation used in system interfaces.

Non-functional requirements are rarely associated with individual system features. Rather, these requirements specify or constraint the emergent properties of the system. Therefore, they may specify system performance, security, availability, and other emergent properties. This means that they are often more critical than individual functional requirements.

System users can usually find ways to work around a system function that doesn't really meet their needs.

Non-functional requirements to be considered are:

Reliability: It is the ability of the system to perform its required functions under the stated conditions for a specific period of time.

Scalability: The capability of a system to increase the total throughput under an increase load when resources (typically hardware) are added.

Security: Unauthorized access to the system and its data is not allowed ensure the integrity of the system from accidental or malicious damage.

Usability: It is the ease with which a user can learn to operate, prepare inputs for and interrupt outputs of system or component.

Reusability: It is the use of existing assets in some form within the software development process.

3.2 HARDWARE REQUIREMENTS

Hardware requirements describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware.

Following are the hardware requirements in order to carry out the proposed work.

Table 3.2: Hardware Requirements

Processor	Pentium i3 and above	It is a small chip that resides in computer to receive input and provide appropriate output.
RAM	4GB	It is a hardware inside a computer that temporarily stores data, serving as the

		computers working memory.
Hard Disk	20 GB free space	It is a secondary storage device used to store data permanently.
Other	Keyboard, mouse, monitor	Used for communication

3.3 SOFTWARE REQUIREMENTS

Software describes the connection between the product and other specific software components (name and version), including databases, operating systems, tool, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each.

A software requirement describes the services needed and the nature of the communications. Referred to documents that describe the detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for e.g.: use of a global data area in multitasking operating system), specify this as an implementation constraints.

Following are the software requirements in order to carry out the proposed work.

Table 3.3: Software Requirements

Operating System	Windows7
Front End	Python

3.3.1 WINDOWS OPERATING SYSTEM

Windows 10 is a personal computer operating system developed and released by Microsoft as part of the Windows NT family of operating systems. The Windows user interface was revised to

handle transition between a mouse-oriented interface and a touch screen-optimized interface based on available input devices-particularly on 2-in-1 PCs.

3.3.2 MATLAB

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

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

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software[30] and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

History

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum's long influence on Python is reflected in the title given to him by the Python community: Benevolent Dictator For Life (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x[37] and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler to improve performance under concurrent workloads.

Features and philosophy

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta programming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator

expressions.[47] The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

Beautiful is better than ugly

Explicit is better than implicit

Simple is better than complex

Complex is better than complicated

Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at

the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers.

Syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

Statements and control flow

The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing.

The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).

The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.

The while statement, which executes a block of code as long as its condition is true.

The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.

The raise statement, used to raise a specified exception or re-raise a caught exception.

The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.

The def statement, which defines a function or method.

The with statement, from Python 2.5 released on September 2006,[59] which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.

The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.

The assert statement, used during debugging to check for conditions that ought to apply.

The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.

The import statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import * or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ...

The print statement was changed to the print() function in Python 3.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will. However, better support for coroutine-like functionality is

provided in 2.5, by extending Python's generators. Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

Expressions

Some Python expressions are similar to languages such as C and Java, while some are not:

Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division and integer division. Python also added the ** operator for exponentiation.

From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.

In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c.

Python uses the words and, or, not for its boolean operators rather than the symbolic &&, ||, ! used in Java and C.

Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression.

Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.

Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).

Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python).

Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.

Python features sequence unpacking where multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.

Python has a "string format" operator %. This functions analogous to printf format strings in C, e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2". In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'.

Python has various kinds of string literals:

Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".

Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.

Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.

Python has array index and array slicing expressions on lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

List comprehensions vs. for-loops

Conditional expressions vs. if blocks

The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code but if c = 1: ... causes a syntax error in Python.

Methods

Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the metaclass type (itself an instance of itself), allowing metaprogramming and reflection.

Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

Mathematics

Python has the usual C language arithmetic operators (+, -, *, /, %). It also has ** for exponentiation, e.g. 5**3 == 125 and 9**0.5 == 3.0, and a new matrix multiply @ operator is included in version 3.5.[80] Additionally, it has a unary operator (~), which essentially inverts all the bits of its one argument. For integers, this means ~x=-x-1. Other operators include bitwise shift operators x << y, which shifts x to the left y places, the same as x*(2**y) , and x >> y, which shifts x to the right y places, the same as x//(2**y) .

The behavior of division has changed significantly over time:

Python 2.1 and earlier use the C division behavior. The / operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. 7/3 == 2 and -7/3 == -2.

Python 2.2 changes integer division to round towards negative infinity, e.g. 7/3 == 2 and -7/3 == -3. The floor division // operator is introduced. So 7//3 == 2, -7//3 == -3, 7.5//3 == 2.0 and -7.5//3 == -3.0. Adding from __future__ import division causes a module to use Python 3.0 rules for division (see next).

Python 3.0 changes / to be always floating-point division. In Python terms, the pre-3.0 / is classic division, the version-3.0 / is real division, and // is floor division.

Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation (a + b)//b == a//b + 1 is always true. It also means that the equation b*(a//b) + a%b == a is valid for both positive and negative values of a. However, maintaining the validity of this equation means that while the result of a%b is, as expected, in the half-open interval [0, b), where b is a positive integer, it has to lie in the interval (b, 0] when b is negative.

Python provides a round function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: round(0.5) is 1.0, round(-0.5) is −1.0. Python 3 uses round to even: round(1.5) is 2, round(2.5) is 2.

Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression a < b < c tests whether a is less than b and b is less than c. C-derived languages interpret this expression differently: in C, the expression would first evaluate a < b, resulting in 0 or 1, and that result would then be compared with c.

Python has extensive built-in support for arbitrary precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type int, to arbitrary precision, belonging to the Python type long, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the int and long types was eliminated; this behavior is now entirely contained by the int class.) The Decimal type/class in module decimal (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes.[90] The Fraction type in module fractions (since version 2.6) provides arbitrary precision for rational numbers.

Due to Python's extensive mathematics library, and the third-party library NumPy that further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation.

Libraries

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However,

because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

Graphical user interfaces

Web frameworks

Multimedia

Databases

Networking

Test frameworks

Automation

Web scraping

Documentation

System administration

Scientific computing

Text processing

Image processing

Development environments

See also: Comparison of integrated development environments § Python

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which the user enters statements sequentially and receives results immediately.

Other shells, including IDLE and IPython, add further abilities such as auto-completion, session state retention and syntax highlighting.

As well as standard desktop integrated development environments, there are Web browser-based IDEs; SageMath (intended for developing science and math-related Python programs); PythonAnywhere, a browser-based IDE and hosting environment; and Canopy IDE, a commercial Python IDE emphasizing scientific computing.

Implementations

CPython is the reference implementation of Python. It is written in C, meeting the C89 standard with several select C99 features. It compiles Python programs into an intermediate bytecode[101] which is then executed by its virtual machine. CPython is distributed with a large standard library written in a mixture of C and native Python. It is available for many platforms, including Windows and most modern Unix-like systems. Platform portability was one of its earliest priorities.

CHAPTER 4

SYSTEM DESIGN

A software product is a high level complex entity. Its development usually follows Software Development Life Cycle (SDLC). The second stage in the SDLC is the system design stage. The objective of the design stage is to produce the overall design of the software.

The design stage involves two sub stages:

1. High-Level Design
2. Detailed Design

4.1 HIGH-LEVEL DESIGN

High-Level Design is the first of the two-stage design process. High-Level Design gives an overview of the system flow and gives the solutions architecture to the proposed application requirements and provides overall solution architecture of the application.

High-Level Design describes the logic. Here, the basic knowledge about the system design and the architecture can be visualized. The issues that can be seen in this part are primary components for any design. A software design may be platform independent or platform specific, depending on the availability of the technology called for by the design.

System design aims to identify the modules that should be included in the system, the specification of these modules and interaction between them to produce the desired results. At the end of the system design, all the major data structures, file formats, output format as well as major modules in the system and their specifications are decided. High-Level Design is defined as to check the functionality of the internal code in the program.

The following high level design components are discussed below with reference to the proposed work. They are:

1. System Architecture
2. Data Flow Diagram

The image processing and sensor based method is used to differentiate between naturally and artificially ripened bananas. This involves feature extraction of bananas followed by feature analysis to find the discriminatory behaviour between the categories of banana.

Fig.4.1 represents the block diagram of proposed method to differentiate between banana samples using image processing.

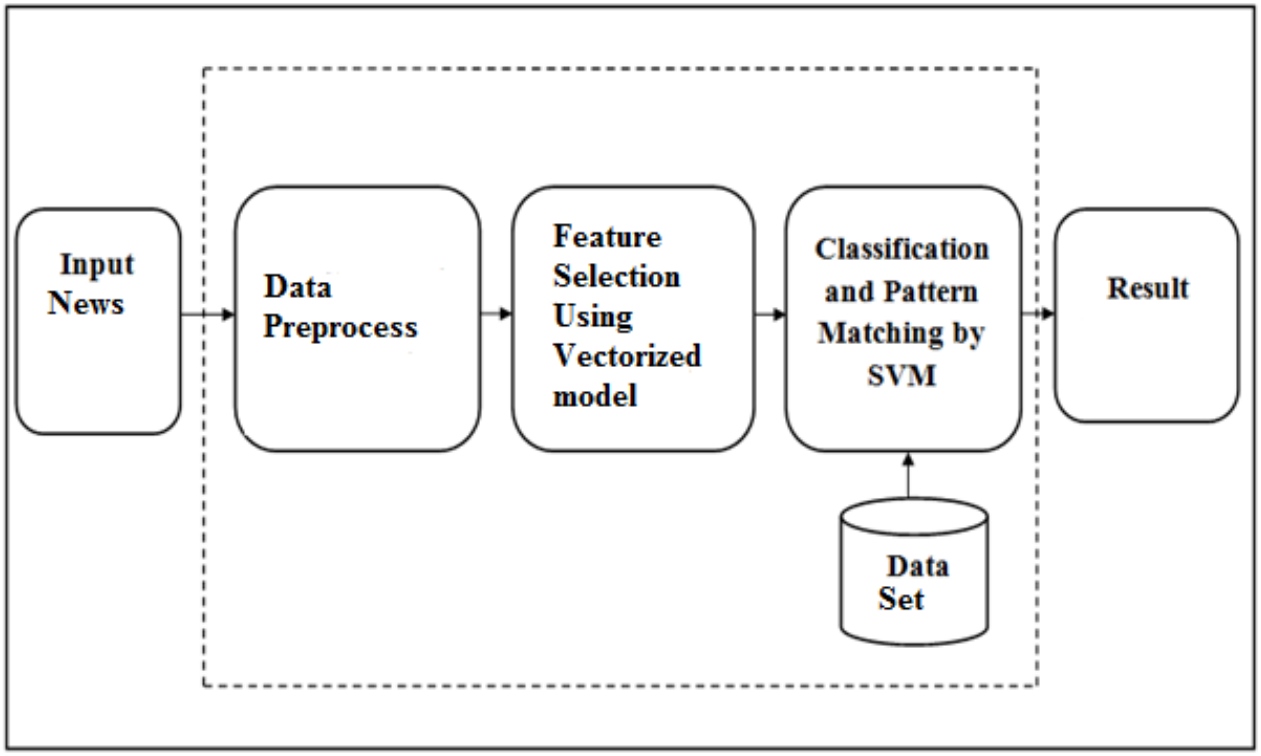


Figure 4.1: Block diagram of Proposed framework

Proposed framework includes four modules: Data Pre-process, Feature Extraction, classification. The Creation of dataset is done by using the web scraping techniques. It collects the real news from The Deccan herald and twitter website. Followed by Feature selection is done by ngram vectorized model. In the classification module, SVM Classifier based on supervised learning, which is widely used for classification tasks is utilized. It consists of learning algorithms that analyse data and identify patterns, used for classification and analysis.SVM classifier depends on the kernel function used to distribute the data into different classes. The resultant extracted features are matched using SVM classifier which decides whether the news is fake or real. The modules are described as follows:

a. CountVectorizer

CountVectorizer converts a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix. The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

Create an instance of the CountVectorizer class. If a string, it is passed to _check_stop_list and the appropriate stop list is returned is currently the only supported string value.

If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens.

The result bag of words or vocabulary matrix is give as train and test input.

Then the four machine learning algorithm is applied to arrive different result and compared.

b. TfidfVectorizer

One issue with simple counts is that some words like “the” will appear many times and their large counts will not be very meaningful in the encoded vectors. Thus an alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF (Term Frequency – Inverse Document Frequency) which are the components of the resulting scores assigned to each word. Term Frequency: This summarizes how often a given word appears within a document. Inverse Document Frequency: This downscales words that appear a lot across documents.

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.

Here we used stop_words='english', max_df=0.7, which removes the stops words from dataset text and removes the words which appears in more than 70% of documents.

c. ngram model

N-gram modeling is a popular feature identification and analysis approach used in language modeling and Natural language processing fields. The most used n-gram models in text categorization are word-based and character-based n-grams. Here, we use word-based n-gram to represent the context of the document and generate features to classify the document. We develop a simple n-gram based classifier to differentiate between fake and honest news articles. The idea is to generate various sets of n-gram frequency profiles from the training data to represent fake and truthful news articles. We used several baseline n-gram features based on words and examined the effect of the n-gram length on the accuracy of four different classification algorithms.

d. Machine learning algorithm

The above three feature selection output as train and test input given to following four machine learning classification algorithm as input and arrive the results. The classification algorithm used are NaiveBayes, Random Forest, LogisticRegression, SVM.

4.2 SYSTEM ARCHITECTURE

System Architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structure of the system architecture can comprise system components, the externally visible properties of these components, the relationships between them. It can provide a plan from which products can be produced, and system developed, that will work together to implement the overall system.

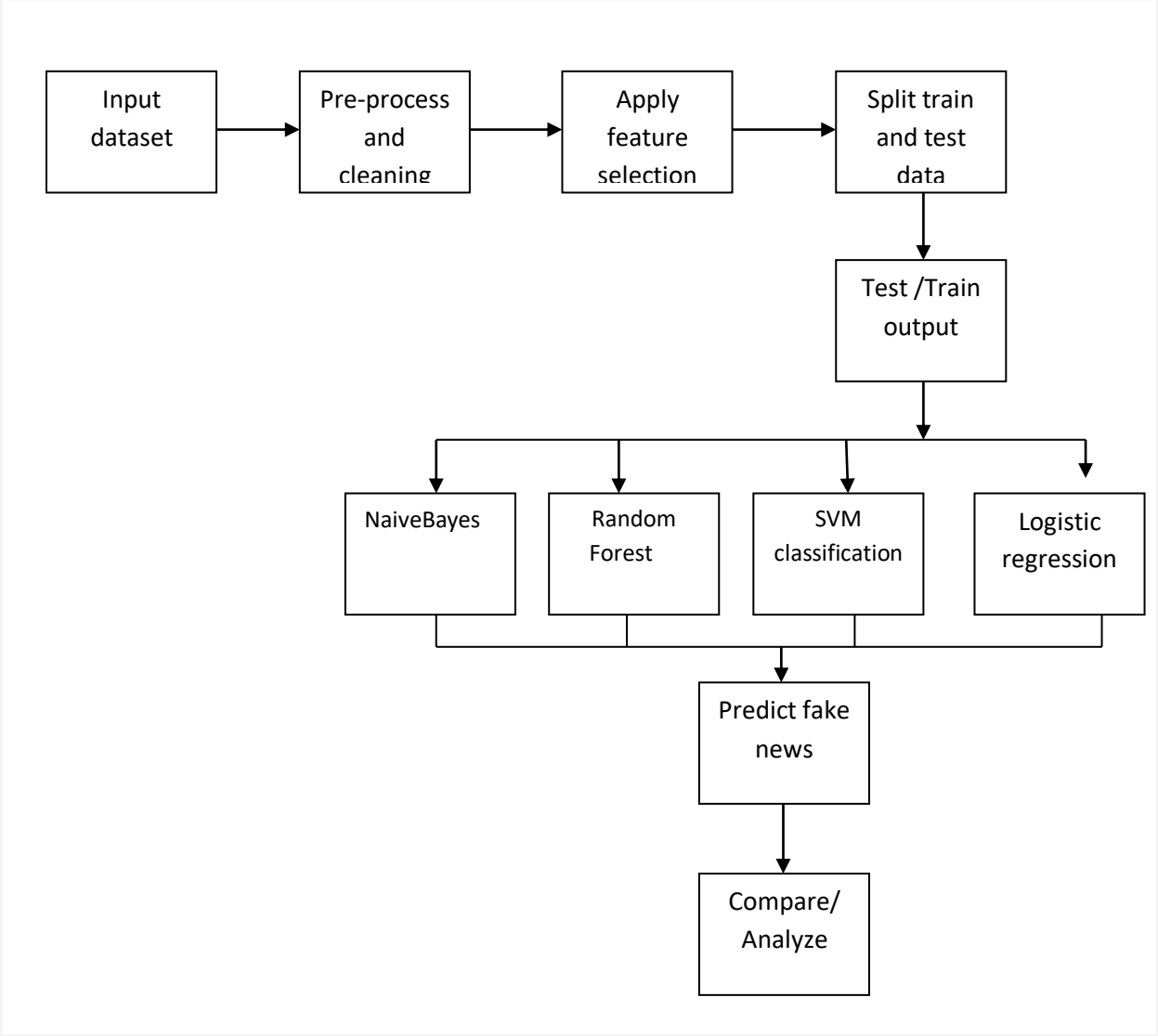


Figure 4.2: System Architecture

Figure 4.2 shows the system architecture of the proposed framework. In this news dataset is given as input to the data preprocess module where the data get cleaned. The architecture involves feature selection, classification steps .

4.3 DATABASE DESIGN

Database design is the process of producing a detailed data model of a database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

Figure 4.3 describes the database design of the project. In this database trained news data is stored. Whenever the test news is given as input it will mapped with the trained dataset. It contains two classes where c1 is for real and c2 for fake.

	title	text	label
8476	You Can Si	Daniel	FAKE
10294	Watch The	Google	FAKE
3608	Kerry to g	U.S.	REAL
10142	Bernie su	â€"	FAKE
875	The Battle	It's	REAL
6903	Tehran, U:		FAKE
7341	Girl Horrif	Share	FAKE
95	â€˜Britain	A Czech st	REAL
4869	Fact check	Hillary	REAL
2909	Iran repor	Iranian	REAL
1357	With all th	CEDAR	REAL
988	Donald Tr	Donald	REAL
7041	Strong Sol	Click	FAKE
7623	10 Ways A	October	FAKE
1571	Trump tak	Killing Ob	REAL

Figure 4.3: Database design

- Determine the data to be stored in the database.
- Determine the relationships between the different data elements.
- Superimpose a logical structure upon the data on the basis of these relationships.

4.4 DATA FLOW DIAGRAM

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one.

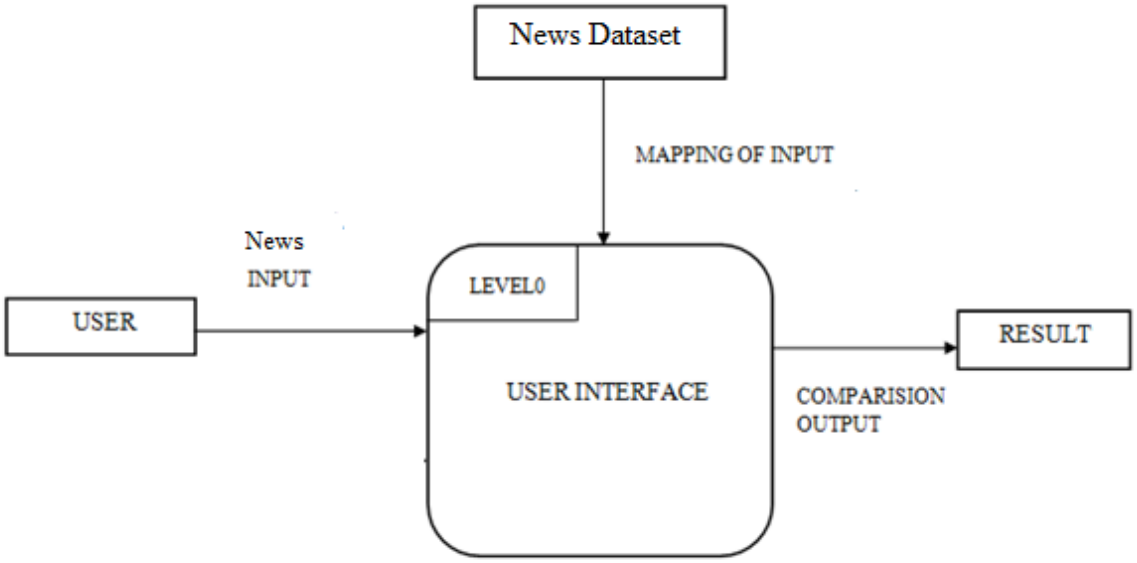


Figure 4.4: Level 0 DFD

Figure 4.4 shows the level 0 dataflow diagram where it indicates the interaction of the user, news dataset with the user interface to display the appropriate result. The user will give news as input to the user interface then it undergoes various stages of machine learning process and the features are extracted, those features are mapped with the news dataset and then after mapping the result is displayed.

4.5 USECASE DIAGRAM

A use case is a coherent piece of functionality that a system provides interacting with actors. It describes a system which involves a set of use cases and a set of actors. In our project use case diagram indicates interaction between the user and system. The user will give the news as a input with the help of user interface. The news undergoes various stages of machine learning like vectorization, feature extraction and classification after that it will display the result in the user interface. Figure 4.5 explains the use case diagram of the proposed framework.

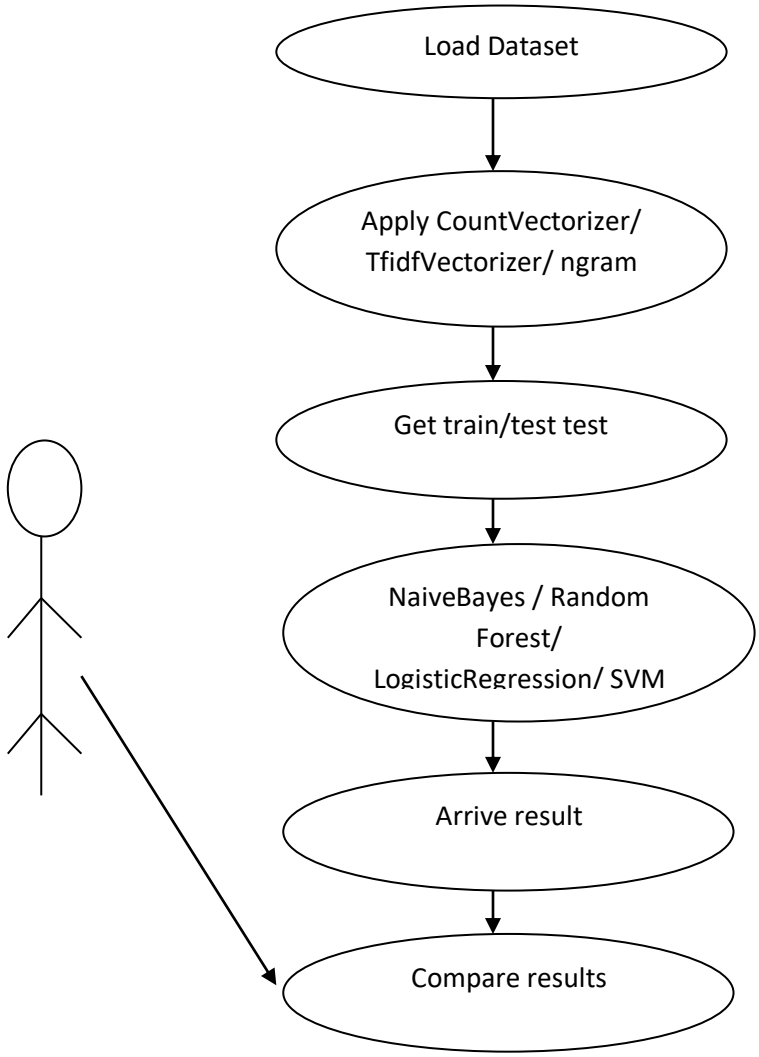


Figure 4.5: Use case diagram

4.6 SEQUENCE DIAGRAM

A Sequence diagram given in 4.6 shows how a set of objects communicate with each other to perform a Complex task. This type of diagram allows the other developer to verify that the interaction is correct. A Sequence diagram shows, a parallel vertical lines(lifelines), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. In our project sequence diagram indicates that the user will give load the dataset dataset will go under the preprocess, feature extraction and classification process.

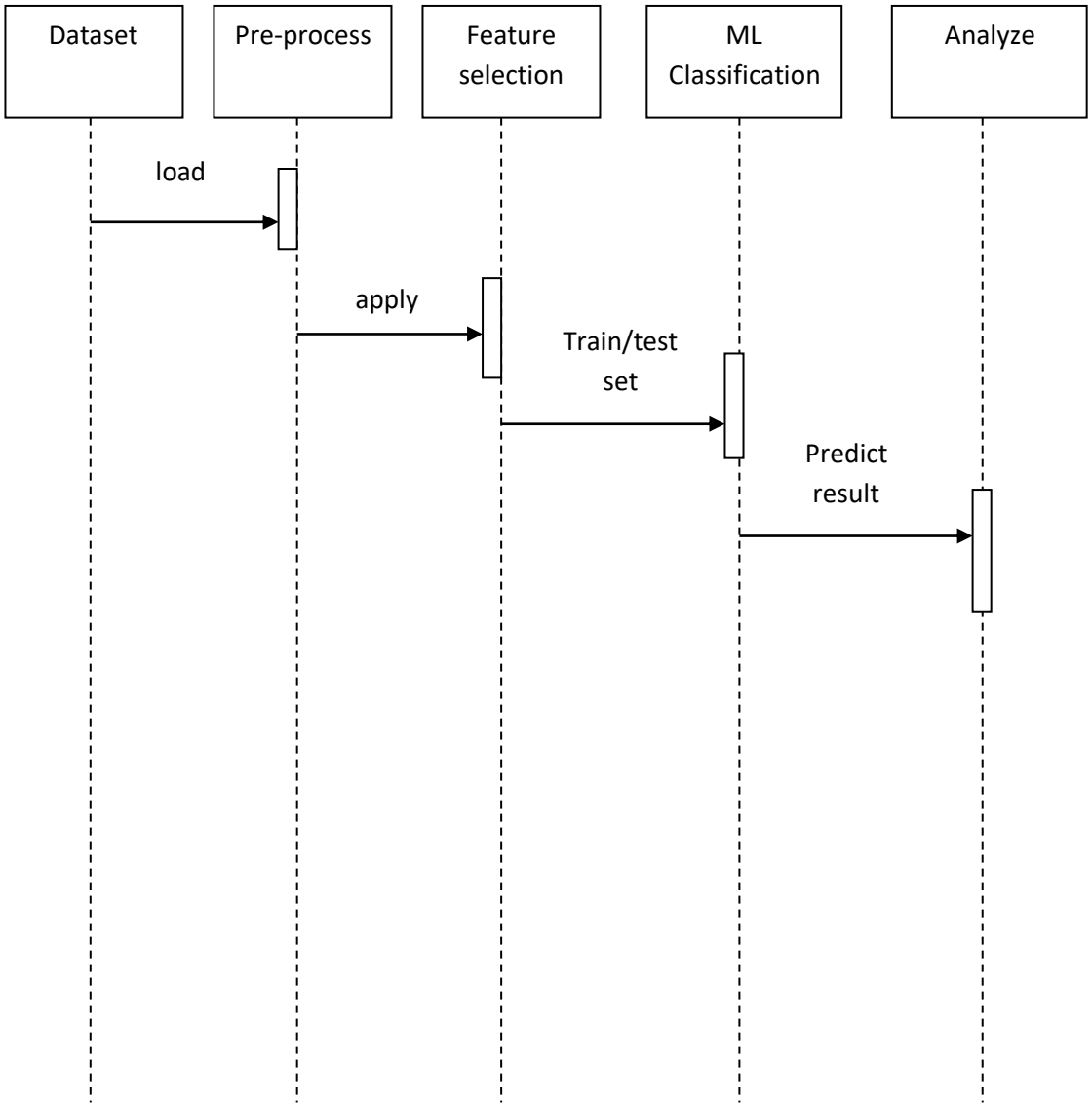


Figure 4.6: Sequence diagram

CHAPTER 5

IMPLEMENTATION

An implementation is a realization of a technical specification or algorithm as a program, software component or other computer system through computer programming and deployment. Many implementation may exists for a given specification or standard. To implement a system successfully, a large number of inter-related tasks need to be carried out in an appropriate sequence. System implementation generally benefits from high levels of user involvement and management support.

Implementation is the stage in the project where theoretical design is turned into a working system and is giving confidence on the new system for the users that it will work efficiently and effectively. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the changeover, an evaluation of change over methods. The implementation process begins with the preparing a plans for the implementation of the system. According to this plan, the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system.

Implementation is the most important and critical phase in achieving a successful new system, that gives the user confidence that the new system will work and is effective. The system can be implemented only after through testing is done and if it is found to be working according to the specification.

Fake news detection is done the taken dataset by applying feature extraction techniques

- `CountVectorizer`
- `Ngram model`
- `TfidfVectorizer`

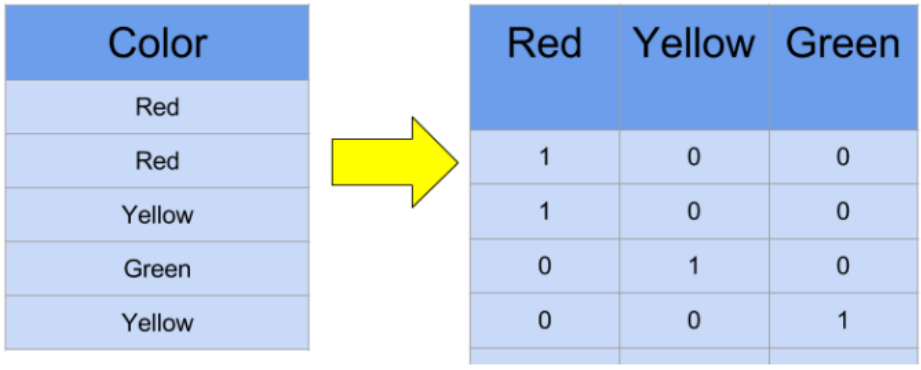
Machine learning algorithm applied on the above extracted features

- Naïve Bayes
- Random Forest
- Logistic Regression
- SVM

5.1 Count Vectorizer:

We will be creating vectors that have a dimensionality equal to the size of our vocabulary, and if the text data features that vocab word, we will put a one in that dimension. Every time we encounter that word again, we will increase the count, leaving 0s everywhere we did not find the word even once.

The result of this will be very large vectors, if we use them on real text data, however, we will get very accurate counts of the word content of our text data. Unfortunately, this won't provide use with any semantic or relational information, but that's okay since that's not the point of using this technique.



5.2 N-gram model:

An N-gram as the sequence of N words, by that notion, a 2-gram (or bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”, and a 3-gram (or trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”.

Let’s start with equation P(w|h), the probability of word *w,* given some history, *h*. For example,

$$P(the \mid its\ water\ is\ so\ transperant\ that)$$

Here,

w =The

h = its water is so transparent that

And, one way to estimate the above probability function is through the relative frequency count approach, where you would take a substantially large corpus, count the number of times you see ***its water is so transparent that***, and then count the number of times it is followed by ***the***.

5.3 Tfidf Vectorizer

Often times, when building a model with the goal of understanding text, you’ll see all of stop words being removed. Another strategy is to score the relative importance of words using TF-IDF.

Term Frequency (TF)

The number of times a word appears in a document divided by the total number of words in the document. Every document has its own term frequency.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Data Frequency (IDF)

The log of the number of documents divided by the number of documents that contain the word ***w***. Inverse data frequency determines the weight of rare words across all documents in the corpus.

$$idf(w) = log(\frac{N}{df_t})$$

5.4 Fakenews

```
#-----------------------------------------------------------
# Include Libraries
#-----------------------------------------------------------

import pandas as pd
from sklearn.model_selection import train_test_split
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import metrics
from matplotlib import pyplot as plt
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.feature_extraction.text import HashingVectorizer
import itertools
import numpy as np
import re
import csv
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem import SnowballStemmer
```

```
#---------------------------------------------------------------
# Function to plot the confusion matrix
#---------------------------------------------------------------
```

```
def plot_confusion_matrix(cm, classes,normalize=False,title='',cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
```

```
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig("results/"+title+".png")
    plt.pause(5)
    plt.show(block=False)
    plt.close()

#------------------------------------------------------------
```

```
# Naive Bayes classifier for Multinomial model
#------------------------------------------------------------
def NaiveBayes(xtrain,ytrain,xtest,ytest,ac,title1):
                                        clf = MultinomialNB(alpha=.01, fit_prior=True)
                                        clf.fit(xtrain, ytrain)
                                        pred = clf.predict(xtest)
                                        score = metrics.accuracy_score(ytest, pred)
                                        print("accuracy:   %0.3f" % score)
                                        cm = metrics.confusion_matrix(ytest, pred, labels=['FAKE', 'REAL'])
                                        plot_confusion_matrix(cm, classes=['FAKE', 'REAL'],title=title1 +' Confusion matrix Naive Bayes')
                                        print(cm)
                                        ac.append(score)


def Logreg(xtrain,ytrain,xtest,ytest,ac,title1):
    i=1
    logreg = LogisticRegression(C=9)
    logreg.fit(xtrain,ytrain)
    pred = logreg.predict(xtest)
    score = metrics.accuracy_score(ytest, pred)
    print("accuracy:   %0.3f" % score)
    cm = metrics.confusion_matrix(ytest, pred, labels=['FAKE', 'REAL'])
    plot_confusion_matrix(cm, classes=['FAKE', 'REAL'],title=title1 +' Confusion matrix Logistic')
```

```
    print(cm)
    ac.append(score)


def RForest(xtrain,ytrain,xtest,ytest,ac,title1):
    clf1 = RandomForestClassifier(max_depth=50, random_state=0,n_estimators=25)
    clf1.fit(xtrain,ytrain)
    pred = clf1.predict(xtest)
    score = metrics.accuracy_score(ytest, pred)
    print("accuracy:   %0.3f" % score)
    cm = metrics.confusion_matrix(ytest, pred, labels=['FAKE', 'REAL'])
    plot_confusion_matrix(cm, classes=['FAKE', 'REAL'],title=title1 +' Confusion matrix RForest')
    print(cm)
    ac.append(score)



def SVM(xtrain,ytrain,xtest,ytest,ac,title1):
    clf3 = SVC(C=100, gamma=0.1)
    clf3.fit(xtrain, ytrain)
    pred = clf3.predict(xtest)
    score = metrics.accuracy_score(ytest, pred)
    print("accuracy:   %0.3f" % score)
    cm = metrics.confusion_matrix(ytest, pred, labels=['FAKE', 'REAL'])
    plot_confusion_matrix(cm, classes=['FAKE', 'REAL'],title=title1 +' Confusion matrix SVM')
    print(cm)
```

```
    ac.append(score)


def process(xtrain,ytrain,xtest,ytest,ac,title):
                                        print("For Multinomial Naive BayesModel")
                                        NaiveBayes(xtrain,ytrain,xtest,ytest,ac,title)

                                        print("For Random Forest Classifiers")
                                        RForest(xtrain,ytrain,xtest,ytest,ac,title)

                                        print("For Support Vector Machine_Radial Basis Function Classifier")
                                        SVM(xtrain,ytrain,xtest,ytest,ac,title)

                                        print("For Logarithamic Classifier")
                                        Logreg(xtrain,ytrain,xtest,ytest,ac,title)


def MainProcessCount(path):
                                        df = pd.read_csv(path)
                                        print(df.head())
                                        y = df.label
                                        df.drop("label", axis=1)    #where numbering of news article is done that column is dropped in dataset
                                        X_train, X_test, y_train, y_test = train_test_split(df['text'], y, test_size=0.33, random_state=53)
```

```
count_vectorizer = CountVectorizer(stop_words='english')
count_train = count_vectorizer.fit_transform(X_train) # Learn the vocabulary dictionary and return term-document matrix.
count_test = count_vectorizer.transform(X_test)

colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#8c564b"]
explode = (0.1, 0, 0, 0, 0)

al=["NaiveBayes","Random Forest","LogisticRegressio","SVM"]
cac=[]

process(count_train,y_train,count_test,y_test,cac,"COUNT")
print(cac)

result2=open('results/CountAccuracy.csv', 'w')
result2.write("Algorithm,Accuracy" + "\n")
for i in range(0,len(cac)):
    print(al[i]+","+str(cac[i]))
    result2.write(al[i] + "," +str(cac[i]) + "\n")
result2.close()
```

```
                    fig = plt.figure(0)
                    df =  pd.read_csv('results/CountAccuracy.csv')
                    acc = df["Accuracy"]
                    alc = df["Algorithm"]
                    plt.bar(alc, acc, align='center', alpha=0.5,color=colors)
                    plt.xlabel('Algorithm')
                    plt.ylabel('Accuracy')
                    plt.title('Count Accuracy Value')
                    fig.savefig('results/CountAccuracy.png')
                    plt.pause(5)
                    plt.show(block=False)
                    plt.close()


def MainProcessTfidf(path):
                    df = pd.read_csv(path)
                    print(df.head())
                    y = df.label
                    df.drop("label", axis=1)      #where numbering of news article is done that column is dropped in dataset
                    X_train, X_test, y_train, y_test = train_test_split(df['text'], y, test_size=0.33, random_state=53)


                    tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)    # This removes words which appear in more than 70% of the articles
                    tfidf_train = tfidf_vectorizer.fit_transform(X_train)
                    tfidf_test = tfidf_vectorizer.transform(X_test)
```

```
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
"#8c564b"]

explode = (0.1, 0, 0, 0, 0)

al=["NaiveBayes","Random
Forest","LogisticRegressio","SVM"]

cac=[]

tac=[]

nac=[]

process(tfidf_train,y_train,tfidf_test,y_test,tac,"TFIDF")

print(tac)

result2=open('results/TfidfAccuracy.csv', 'w')

result2.write("Algorithm,Accuracy" + "\n")

for i in range(0,len(tac)):

    print(al[i]+","+str(tac[i]))

    result2.write(al[i] + "," +str(tac[i]) + "\n")

result2.close()

fig = plt.figure(0)

df =  pd.read_csv('results/TfidfAccuracy.csv')

acc = df["Accuracy"]

alc = df["Algorithm"]
```

```
                plt.bar(alc, acc, align='center', alpha=0.5,color=colors)
                plt.xlabel('Algorithm')
                plt.ylabel('Accuracy')
                plt.title('Tfidf Accuracy Value')
                fig.savefig('results/TfidfAccuracy.png')
                plt.pause(5)
                plt.show(block=False)
                plt.close()


def MainProcessNgram(path):
                df = pd.read_csv(path)
                print(df.head())
                y = df.label
                df.drop("label", axis=1)      #where numbering of news article is done that column is dropped in dataset
                X_train, X_test, y_train, y_test = train_test_split(df['text'], y, test_size=0.33, random_state=53)


                n_vect = CountVectorizer(min_df = 5, ngram_range = (1,2)).fit(X_train)
                n_train = n_vect.fit_transform(X_train)
                n_test = n_vect.transform(X_test)
```

```
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
"#8c564b"]

explode = (0.1, 0, 0, 0, 0)


al=["NaiveBayes","Random
Forest","LogisticRegressio","SVM"]

cac=[]

tac=[]

nac=[]


process(n_train,y_train,n_test,y_test,nac,"NGRAM")

print(nac)


result2=open('results/NgramAccuracy.csv', 'w')

result2.write("Algorithm,Accuracy" + "\n")

for i in range(0,len(nac)):

   print(al[i]+","+str(nac[i]))

   result2.write(al[i] + "," +str(nac[i]) + "\n")

result2.close()



fig = plt.figure(0)

df =  pd.read_csv('results/NgramAccuracy.csv')

acc = df["Accuracy"]

alc = df["Algorithm"]

plt.bar(alc, acc, align='center', alpha=0.5,color=colors)
```

```
plt.xlabel('Algorithm')
plt.ylabel('Accuracy')
plt.title('Ngram Accuracy Value')
fig.savefig('results/NgramAccuracy.png')
plt.pause(5)
plt.show(block=False)
plt.close()
```

CHAPTER 6

SYSTEM TESTING

Testing is an important phase in the development life cycle of the product. This is the phase where all the error remaining in all the phases will be detected. Hence testing performs a very critical role for quality assurance and ensuring the reliability of the software. During the test, the program to be tested is executed with a set of test cases and the output of the program for the tests cases is evaluated to determine whether the program is performing as expected.

Testing of software or hardware is conducted on complete system to evaluate its compliance with specified requirement. System testing is performed on entire system in the context of functional requirements specification and/or system requirement specification. Testing is an investigatory phase, where focus is to have almost a destructive attitude and test not only the design, but also the behaviour and even the believed expectation of the customer.

Errors which are found are corrected by using the following testing steps and correction will be recorded for future references. Thus a series of testing is performed on the system before it is ready for implementation.

6.1 TESTING OBJECTIVES

The testing objectives are as follows:

- Testing is the process of executing the program with the intent of finding an error.
- A good test case is one that has a high probability of finding an error.
- Testing cannot show the absence of defects.

6.2 TEST TYPES

Different types of tests are mentioned below

1. Unit testing
2. Integration testing
3. System testing
    - Validation testing

- Black box testing
- White box testing

1. Acceptance testing

6.2.1 UNIT TESTING

A unit is the smallest piece of software that can be tested. This usually means the software can be compiled, linked or loaded into memory. A typical example in a procedural programming language would be a function/procedure or a group of these contained in a source file. In an object-oriented programming language, this typically refers to simple classes and interfaces. An example unit testing session would involve a number of calls to the unit (function, procedure or method) under test where each call might be preceded by a setup code that generates appropriate method parameters wherever required and another call performed after the test to check whether the test was successful or unsuccessful. Unit testing is the lowest testing level. It helps to refine the testing process so that reasonable system reliability can be expected when testing is performed on the next hierarchical levels. Testing at the unit level helps to expose bugs that might appear to be hidden if a big-bang testing approach was used and unit testing omitted. When these stealth bugs are not uncovered and corrected they could cause unexpected system faults or crashes when running the system.

Unit testing is usually seen as a white-box testing technique. This is because its main focus is on the implementation of the unit being tested i.e. the class, interface, function or method under test. Unit testing seeks to find if the implementation satisfies the functional specification. It is not unusual to have system level requirements tested at the unit level for example a system might specify in its functional requirements to have a detailed complex algorithm. Since unit testing focuses on implementation and requires thorough understanding of the systems functional specification, it is usually performed by the developers. What then happens is that the programmer writes test cases after she/he has implemented the program. The obvious problem with this is that the test cases are unconsciously written to suit the programmer’s implementation rather than the initial functional specification.

6.2.2 SYSTEM TESTING

System testing begins after completion of integration testing. System testing is done to prove that the system implementation does not meet the system requirements specification. Test planning for system testing is usually one of the first to be processed as all that is needed is the system requirements specifications and this is usually available very early in the project development lifecycle. System test planning and system testing are normally performed by a test team if there is one. System test planning phase is very dependent on the high-level design specification in the development process. As a result any errors made in translating the requirements specification and the design specification would be very drastic as it would propagate downwards to the lower levels of test and development.

WHITE BOX TESTING

White box deals with internal working of code to ensure there is no redundant code written in software. This involves testing of line of code, program, flow, logic, loop, structure, functions, class communication testing and other internal testing of program

6.2.3 ACCEPTANCE TESTING

Acceptance testing is concerned with showing that the end product does not meet the user requirement i.e. that the user is not satisfied with the system. Since acceptance testing is based solely on user requirements specs, it is usually the first to receive full planning. The acceptance test plan is ideally written by the end-user. This is not possible in all projects due to the extremely large user-base for some application. In these cases a group of testers would be brought forward to represent the end-users and write the test plan. Acceptance testing is very user centered and is normally performed by the users in an environment that is just like the deployment environment. It could be carried out by either benchmark testing or pilot testing. The major objective of benchmark testing is to measure the system's performance in the end-user environment. This is usually simulated with the required configuration settings.

6.3 SAMPLE TEST CASES

Test cases with id1, id2, id3, id4, id5, id6, id7, id8 are illustrated in the following tables 6.1, 6.2, 6.3, 6.4 and 6.5 respectively.

Unit testing

Table 6.1: Training Data

TEST CASE ID	**TEST2**
Test Case Title	Loading the nes dataset
Action	Load the train dataset
Expected output	Creates training dataset
Actual Output	Creates a training dataset
Result	PASS

Table 6.2: Test case dealing with the image news

TEST CASE ID	**TEST1**
Test Case Title	News sentence as input for test phase
Action	Creating dataset of news
Expected output	Scrap the web news and store in train dataset
Actual Output	Scraps the web news and stores in train dataset
Result	PASS

Table 6.3: 4VM Classification

TEST CASE ID	**TEST3**
Test Case Title	SVM Prediction
Action	Press on the prediction button
Expected output	The result shows whether the news is fake or real
Actual Output	The result shows whether the news is fake or real
Result	PASS

CHAPTER 7

RESULTS

The current chapter deals with the analysis of results of the proposed framework. Result is one of the last and important phases in the project development. It explains how the project works and its results. In our project we are detecting whether the news is real or fake. Here, are some of the snapshots of the working project.

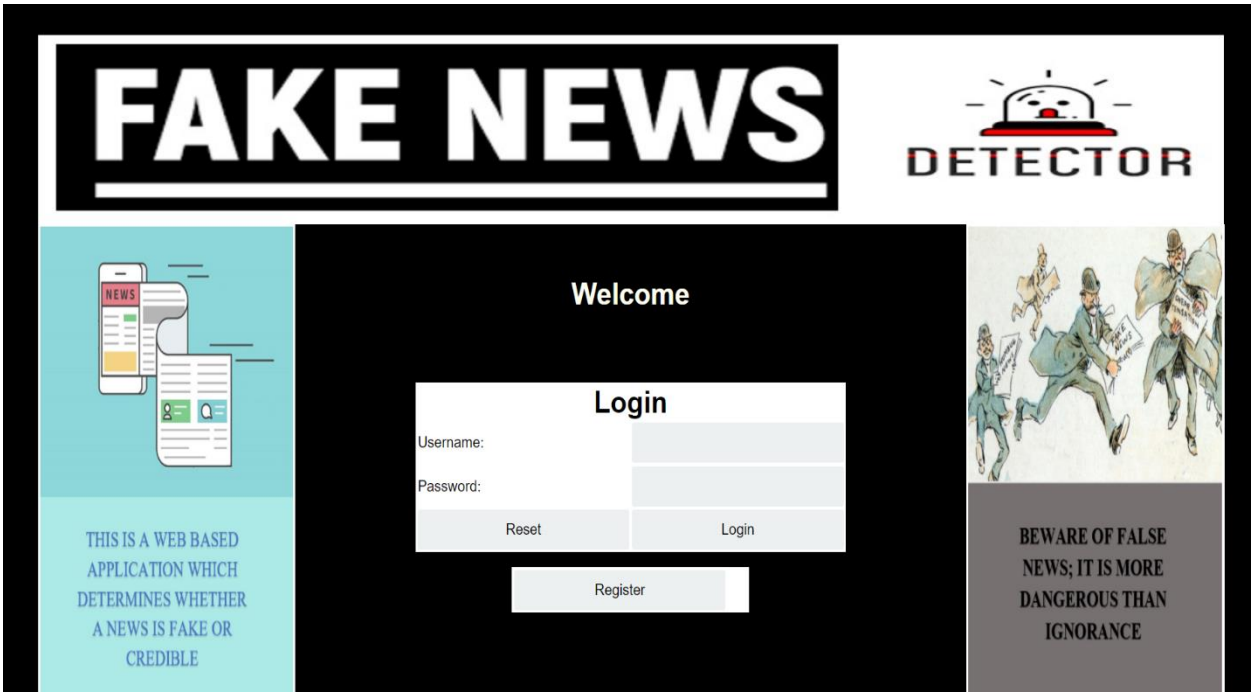


Fig 7.1 : Login Page

Fig 7.2 : Registration Page

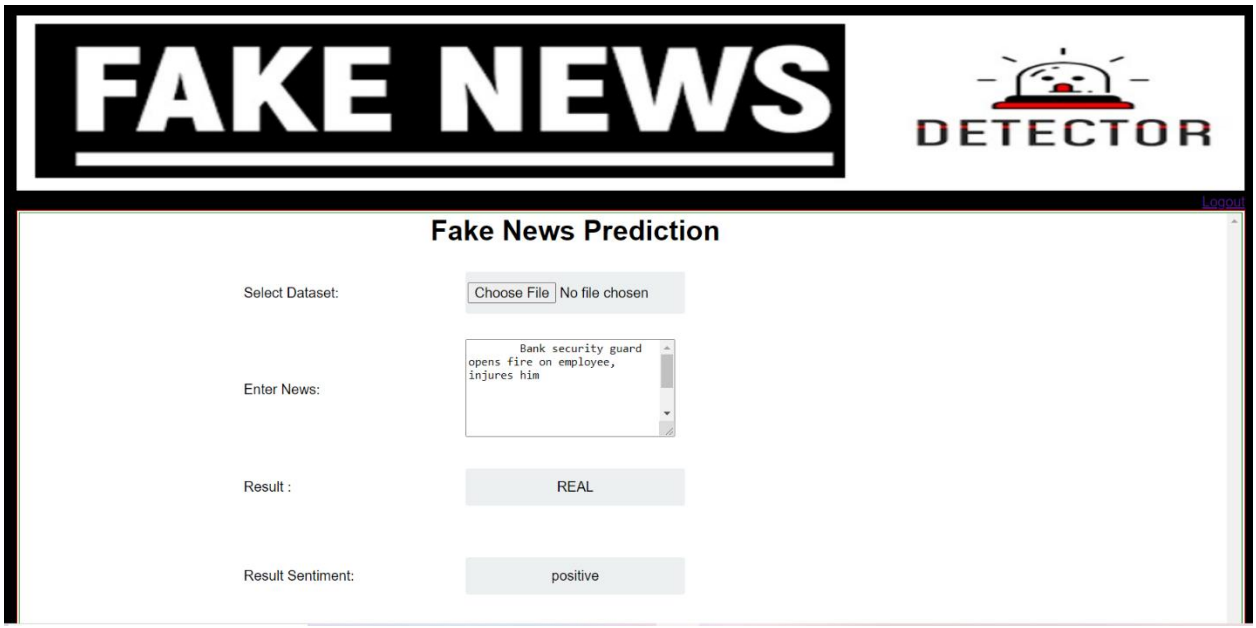


Fig 7.3 : Fake News Predictor

CONCLUSION

The problem of fake news has gained attention in 2016, especially in the aftermath of the last US presidential elections. Recent statistics and research show that 62% of US adults get news on social media. Most of the popular fake news stories were more widely shared on Facebook than the most popular mainstream news stories. A sizable number of people who read fake news stories have reported that they believe them more than news from mainstream media. Dewey claimed that fake news played a huge role in the 2016 US election and that they continue to affect people opinions and decisions.

In proposed work we have presented a detection model for fake news using n-gram analysis through the lenses of different features extraction techniques. Furthermore, we investigated three different features extraction techniques and four different machine learning techniques. The proposed model achieves its highest accuracy when using Tfidf Vectorizer on Logistics regression. The highest accuracy score is 93.7%. Fake news detection is an emerging research area with few public datasets. Proposed model is run on an existing dataset, showing that our model outperforms the original approach published by the authors of the dataset.

REFERENCES

[1] A. Bessi, M. Coletto, G. A. Davidescu, A. Scala, G. Caldarelli, and W. Quattrociocchi, “Science vs Conspiracy: Collective Narratives in the Age of Misinformation,” PLOS ONE, vol. 10, no. 2, p. e0118093, Feb. 2015.

[2] H. Allcott and M. Gentzkow, “Social Media and Fake News in the 2016 Election,” Journal of Economic Perspectives, vol. 31, no. 2, pp. 211–236, May 2017.

[3] A. Gupta, H. Lamba, and P. Kumaraguru, “$1.00 per RT #Boston- Marathon #PrayForBoston: Analyzing fake content on Twitter,” in 2013 APWG eCrime Researchers Summit, Sep. 2013, pp. 1–12.

[4] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, “Fake news detection on social media: A data mining perspective,” ACM SIGKDD Explorations Newsletter, vol. 19, no. 1, pp. 22–36, 2017.

[5] E. Tacchini, G. Ballarin, M. L. Della Vedova, S. Moret, and L. de Alfaro, “Some Like it Hoax: Automated Fake News Detection in Social Networks,” in Proceedings of the Second Workshop on Data Science for Social Good, vol. 1960. Skopje, Macedonia: CEUR-WS, 2017.

[6] G. L. Ciampaglia, P. Shiralkar, L. M. Rocha, J. Bollen, F. Menczer, and A. Flammini, “Computational Fact Checking from Knowledge Networks,” PLOS ONE, vol. 10, no. 6, p. e0128193, Jun. 2015.

[7] M. Del Vicario, A. Bessi, F. Zollo, F. Petroni, A. Scala, G. Caldarelli, H. E. Stanley, and W. Quattrociocchi, “The spreading of misinformation online,” Proceedings of the National Academy of Sciences, vol. 113, no. 3, pp. 554–559, Jan. 2016.

[8] J. C. Hernandez, C. J. Hernandez, J. M. Sierra, and A. Ribagorda, “A first step towards automatic hoax detection,” in Proceedings. 36th Annual 2002 International Carnahan Conference on Security Technology, 2002, pp. 102–114.

[9] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu, “Toward computational fact-checking,” Proceedings of the VLDB Endowment, vol. 7, no. 7, pp. 589–600, 2014.

[10] N. J. Conroy, V. L. Rubin, and Y. Chen, “Automatic deception detection: Methods for finding fake news,” Proceedings of the Association for Information Science and Technology, vol. 52, no. 1, pp. 1–4, 201