CHAPTER 1

INTRODUCTION

In today's digital age, the proliferation of social media platforms and online communities has led to the rise of fake profiles, posing significant challenges for users, businesses, and platforms alike. Fake profiles, also known as impostor accounts or bots, are created with deceptive intent, often to spread misinformation, engage in fraudulent activities, or manipulate public opinion. Detecting and identifying fake profiles has become a crucial endeavor in safeguarding online ecosystems and maintaining trust among users.

The identification of fake profiles is a multifaceted problem that requires a combination of technological solutions, data analysis techniques, and human intervention. With the advancement of artificial intelligence and machine learning algorithms, automated methods for detecting fake profiles have become more sophisticated. These methods leverage various signals and features, such as profile information inconsistencies, unusual activity patterns, and content analysis, to flag suspicious accounts.

However, despite the progress in automated detection techniques, the identification of fake profiles remains challenging due to the evolving tactics employed by malicious actors. Furthermore, false positives and unintended consequences can occur, highlighting the need for a nuanced approach that combines automated algorithms with human judgment and contextual understanding.

By examining the strengths and limitations of existing approaches and proposing novel strategies, we strive to contribute to the development of effective solutions for combating fake profiles and preserving the integrity of online communities. Through interdisciplinary collaboration and ongoing research efforts, we can mitigate the risks posed by fake profiles and foster a safer and more trustworthy online environment for all users.

In the era of online interaction, the proliferation of fake profiles has become a pressing concern, leading to issues ranging from identity theft to misinformation propagation. Leveraging advancements in machine learning and natural language processing (NLP), researchers have devised innovative techniques to identify these deceptive profiles. By analyzing patterns in user behavior, linguistic cues, and semantic inconsistencies, these systems can effectively differentiate between genuine and fraudulent accounts. Through the fusion of NLP with machine learning algorithms, such as neural networks and ensemble methods, these models can continuously adapt to

evolving strategies employed by malicious actors, thereby fortifying online platforms against the proliferation of fake profiles and enhancing user trust and safety.

1.1 PYTHON

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Developed by Guido van Rossum and first released in 1991, Python has gained widespread adoption across various domains including web development, data analysis, artificial intelligence, scientific computing, and more. Here are some key aspects and areas where Python excels:

- Ease of Learning and Readability: Python's syntax is designed to be intuitive and readable, making it accessible to beginners. Its straightforward syntax reduces the cost of program maintenance and development.
- Versatility: Python supports multiple programming paradigms including procedural, object-oriented, and functional programming. This versatility allows developers to choose the most suitable approach for their projects.
- Large Standard Library: Python comes with a vast standard library that provides support for a wide range of tasks such as file I/O, networking, database manipulation, and more. This extensive library reduces the need for third-party modules for many common programming tasks.
- Community and Ecosystem: Python has a large and active community of developers contributing to its ecosystem. This community-driven development has led to the creation of numerous libraries, frameworks, and tools that extend Python's capabilities for various applications. Popular libraries include NumPy, pandas, TensorFlow, Django, Flask, and more.
- Data Science and Machine Learning: Python has become the de facto language for data science and machine learning due to its simplicity and the availability of powerful libraries such as NumPy, pandas, scikit-learn, and TensorFlow. These libraries provide robust tools for data manipulation, analysis, visualization, and machine learning model development.
- Web Development: Python offers several frameworks for web development such as Django, Flask, and Pyramid. Django, in particular, is a high-level web framework that encourages rapid development and clean, pragmatic design.
- Automation and Scripting: Python is commonly used for automation tasks and scripting due to its ease of use and cross-platform compatibility. It's often used for tasks like system administration, web scraping, and process automation.
- Education: Python is widely used in education due to its simplicity and readability. Many educational institutions and programming boot camps use Python as an introductory language for teaching

programming concepts.

- Community Support and Documentation: Python has extensive documentation and a vibrant community that provides support through forums, tutorials, and online resources. This makes it easier for developers to learn, troubleshoot, and collaborate on projects.

1.2 MACHINE LEARNING

Machine learning is a branch of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to learn and improve their performance on a specific task without being explicitly programmed. It is based on the idea that systems can learn from data, identify patterns, and make decisions or predictions with minimal human intervention. Machine learning has become increasingly prevalent in various fields and applications, revolutionizing industries and driving innovation in areas such as:

- Data Mining and Pattern Recognition: Machine learning algorithms excel at discovering patterns and insights within large datasets. They can identify hidden trends, correlations, and anomalies in data, facilitating data-driven decision-making and predictive analytics.
- Natural Language Processing (NLP): NLP techniques leverage machine learning to understand and interpret human language. Applications include sentiment analysis, language translation, chatbots, and text summarization, among others.
- Computer Vision: Machine learning algorithms are used to analyze and interpret visual data, enabling computers to understand images and videos. Computer vision applications include object detection, image classification, facial recognition, and medical image analysis.
- Speech Recognition: Machine learning algorithms power speech recognition systems that can transcribe spoken language into text. These systems are used in virtual assistants, voice-controlled devices, dictation software, and automated customer service.
- Recommendation Systems: Machine learning algorithms drive recommendation engines that provide personalized suggestions to users based on their preferences and past behavior. Examples include product recommendations on e-commerce platforms, content recommendations on streaming services, and friend suggestions on social media.
- Healthcare: Machine learning is transforming healthcare by enabling predictive analytics, disease diagnosis, personalized treatment plans, and drug discovery. It is used in medical imaging, genomics, electronic health records analysis, and remote patient monitoring.
- Finance and Banking: Machine learning algorithms are utilized in fraud detection, credit scoring, algorithmic trading, risk management, and customer service automation within the financial industry.

- Autonomous Vehicles: Machine learning plays a crucial role in the development of autonomous vehicles by enabling perception, decision-making, and navigation capabilities based on sensor data analysis.
- Robotics: Machine learning algorithms are used in robotics for tasks such as object recognition, grasping, motion planning, and adaptive control, enabling robots to operate autonomously in complex environments.
- Industrial Automation: Machine learning is applied in manufacturing and industrial settings for predictive maintenance, quality control, supply chain optimization, and process optimization.

Machine learning techniques include supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, and deep learning. These techniques, coupled with advancements in computing power and data availability, continue to drive innovation and unlock new possibilities across diverse domains, shaping the future of technology and society. However, ethical considerations, privacy concerns, and algorithmic biases remain important considerations in the deployment and use of machine learning systems.

1.3 NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and human language. It encompasses the development of algorithms and techniques that enable computers to understand, interpret, and generate natural language text or speech. NLP has numerous applications across various domains, revolutionizing how humans interact with technology and how machines process and analyze textual data. Here are some key aspects and applications of NLP:

- Language Understanding: NLP enables computers to understand the meaning and intent behind human language. This includes tasks such as named entity recognition (identifying names of people, organizations, locations, etc.), part-of-speech tagging, syntactic parsing, and semantic analysis.
- Machine Translation: NLP powers machine translation systems that can translate text or speech from one language to another. Examples include Google Translate, DeepL, and Microsoft Translator. These systems utilize techniques such as statistical machine translation, rule-based translation, and neural machine translation.
- Sentiment Analysis: NLP techniques are used to analyze and classify the sentiment expressed in text data. Sentiment analysis applications range from social media monitoring and customer feedback analysis to brand reputation management and market research.
- Text Generation: NLP enables computers to generate human-like text based on input data or prompts. This includes tasks such as language modeling, text summarization, dialogue generation, and content generation for chatbots and virtual assistants.

- Information Retrieval: NLP techniques are used to extract relevant information from large volumes of text data. This includes document classification, topic modeling, keyword extraction, and search engine optimization (SEO).
- Question Answering: NLP powers question-answering systems that can understand and respond to natural language questions. Examples include virtual assistants like Siri, Alexa, and Google Assistant, as well as specialized question-answering systems for specific domains like healthcare and customer support.
- Named Entity Recognition (NER): NLP techniques are used to identify and classify named entities within text data, such as names of people, organizations, locations, dates, and numerical values. NER is essential for information extraction, knowledge graph construction, and entity linking.
- Text Summarization: NLP enables automatic summarization of large volumes of text into concise and coherent summaries. Summarization techniques include extractive methods, which select important sentences or phrases from the original text, and abstractive methods, which generate summaries using natural language generation techniques.
- Dialogue Systems: NLP powers conversational agents and dialogue systems that can engage in natural language conversations with users. These systems range from simple chatbots for customer service to advanced virtual assistants capable of understanding context and maintaining coherent dialogue.
- Language Understanding for AI: NLP is a fundamental component of AI systems that require language understanding capabilities, such as autonomous vehicles, smart home devices, and healthcare applications. NLP enables these systems to process and respond to natural language commands and queries from users.

NLP techniques include statistical methods, rule-based approaches, machine learning algorithms, and deep learning models such as recurrent neural networks (RNNs) and transformers. The advancement of NLP technology, coupled with the availability of large-scale datasets and computational resources, continues to drive innovation and unlock new possibilities for natural language understanding and interaction between humans and machines. However, ethical considerations, privacy concerns, and biases in NLP models remain important challenges that require careful attention in the development and deployment of NLP systems.

1.4 SCOPE

The scope of fake profile identification in social networks is substantial and continues to expand due to the growing prevalence of fake profiles and the need to maintain the integrity and security of online communities. Also the scope of fake profile identification in social networks is multi-faceted and dynamic, as it evolves in response to changing tactics used by those who create fake profiles. Social platforms must continually adapt and invest in technologies and strategies to effectively address this issue and protect their users.

1.5 OBJECTIVE

The objective of fake profile detection is multifaceted and crucial in today's digital landscape. It encompasses the identification and mitigation of deceptive accounts across a wide array of online platforms. Paramount among these objectives is the preservation of trust among users. By swiftly identifying and removing fake profiles, platforms can maintain their credibility and foster an environment where users feel safe and confident in their interactions. Additionally, the prevention of fraud and misinformation is a key goal. Fake profiles are often utilized for nefarious purposes such as identity theft, scamming, or spreading false information. By detecting and eliminating these accounts, platforms can protect users from financial harm and ensure the integrity of shared information. Moreover, enhancing the overall user experience is a priority. Fake profiles can disrupt the online experience by flooding platforms with spam, fake reviews, or inappropriate content. By proactively addressing these issues, platforms can improve the quality of interactions and cultivate a more positive environment for legitimate users. Furthermore, safeguarding user privacy and combating cybercrime are essential components of fake profile detection. These efforts aim to protect users from potential privacy breaches and mitigate the risk of cybercriminal activities such as phishing or malware distribution. Ultimately, by preserving platform integrity through effective fake profile detection measures, online platforms can uphold their standards and continue to serve as valuable and trusted spaces for digital interactions

1.6 LIMITATIONS

Despite the advancements in fake profile detection, several limitations persist, posing challenges to the effectiveness and scalability of detection methods. One significant limitation is the cat-and-mouse game between detection algorithms and adversaries who continuously adapt their tactics to evade detection. As detection algorithms become more sophisticated, malicious actors develop more sophisticated techniques to create convincing fake profiles, leading to a perpetual arms race.

Another limitation lies in the reliance on static features for detection, which may not capture the dynamic nature of user behavior and profile information. Fake profile creators can gradually adjust their behavior to mimic genuine users, making it difficult for static feature-based algorithms to distinguish between real and fake profiles accurately.

Furthermore, the issue of imbalanced datasets presents a challenge for machine learning-based detection approaches. Fake profiles are often less prevalent than genuine ones, leading to imbalanced datasets that can bias

detection algorithms towards the majority class. As a result, detection models may struggle to effectively identify fake profiles, particularly those with subtle characteristics.

Moreover, the lack of ground truth labels for training datasets can hinder the development and evaluation of detection algorithms. Labeling profiles as fake or genuine often requires manual inspection or crowdsourcing, which can be time-consuming, subjective, and prone to errors. This limitation can impact the reliability and generalizability of detection models trained on such datasets.

Additionally, privacy concerns arise in the context of fake profile detection, particularly when algorithms analyze user-generated content or social network connections to infer profile authenticity. Balancing the need for effective detection with user privacy rights and data protection regulations presents a delicate ethical challenge for researchers and platform operators.

Lastly, the scalability of detection methods remains an ongoing concern, especially for large-scale social media platforms with millions or billions of users. Developing efficient algorithms that can handle the sheer volume of data generated on these platforms while maintaining high detection accuracy poses a significant technical challenge.

Addressing these limitations requires interdisciplinary collaboration and ongoing research efforts to develop robust, adaptive, and privacy-preserving detection methods capable of effectively mitigating the proliferation of fake profiles in online ecosystems.

CHAPTER 2
LITERATURE SURVEY

Conducting a comprehensive literature survey on fake profile detection reveals a rich landscape of research spanning various domains and methodologies. Numerous studies have explored the development and evaluation of algorithms and techniques for identifying fake profiles across different online platforms. Early research focused on rule-based approaches and manual feature engineering to detect inconsistencies in profile information and behavior. Subsequent studies have embraced machine learning techniques, including supervised, unsupervised, and semi-supervised learning, to extract patterns and classify fake profiles based on features such as social network structure, user behavior, and textual content. Additionally, deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have shown promise in capturing complex relationships and nuances in data for more accurate detection. Furthermore, researchers have investigated the effectiveness of ensemble methods, feature selection techniques, and domain-specific features to enhance detection performance. Beyond algorithmic approaches, studies have also explored the role of user-generated content, social network analysis, and community-based reporting mechanisms in augmenting detection capabilities. Moreover, research has addressed the challenges of data imbalance, scalability, and generalizability in real-world deployment scenarios. Overall, the literature survey underscores the interdisciplinary nature of fake profile detection research, drawing insights from computer science, social network analysis, psychology, and other fields, to advance our understanding and develop effective solutions for combating deceptive accounts in online environments.Guidelines can be used to identify bot accounts, however, they are insufficient to identify fraudulent accounts that have been created by humans. Machine learning without supervision is used to detect bots. Information was compiled using this technical technique based on proximity rather than tagging. Co-attributes allowed grouping functions to effectively separate the bots. The Sybil rank regression method was developed in 2012. The profiles are arranged by interaction, tagging, and wall postings True accounts have a better ranking than false accounts, which are ranked lower. Yet this method was unreliable.

2.1 OVERVIEW

Fake profile identification in social media is a vital process designed to enhance user trust, safety, and the overall quality of online communities. By identifying and addressing fake profiles, social media platforms aim to provide users with a safer and more authentic environment. This process promotes user trust, safeguards privacy, improves the quality of information, and mitigates issues like online harassment and the spread of misinformation. Advances in technology, user education, and community involvement are integral components of this ongoing

effortto create a more secure and enjoyable online space while adapting to the evolving tactics of those creating fake profiles.

2.2 EXISTING SYSTEM

Chai et al awarded on this paper is a proof-of inspiration gain knowledge of. Even though the prototype approach has employed most effective normal systems in normallanguage processing and human-pc interplay, the results realized from the user trying out are significant. By using comparing this simple prototype approach with a wholly deployed menu procedure, they've discovered that users, principally beginner users, strongly pick the common language dialog-based approach. They have additionally learned that in an ecommerce environment sophistication in dialog administration is most important than the potential to manage complex typical language sentences. In addition, to provide effortless access to knowledge on ecommerce web sites, natural language dialog-based navigation and menu-pushed navigation should be intelligently combined to meet person's one-of-a-kind wants. Not too long ago, they have got accomplished development of a new iteration of the approach that includes enormous enhancements in language processing, dialog administration and information management. They believed that average language informal interfaces present powerful personalized alternatives to conventional menu pushed or search-based interfaces to web sites. LinkedIn is greatly preferred through the folks who're in the authentic occupations. With the speedy development of social networks, persons are likely to misuse them for unethical and illegal conducts. Creation of a false profile turns into such adversary outcomes which is intricate to identify without apt research. The current solutions which were virtually developed and theorized to resolve this contention, mainly viewed the traits and the social network ties of the person's social profile. However, in relation to LinkedIn such behavioral observations are tremendously restrictive in publicly to be had profile data for the customers by the privateness insurance policies.

The limited publicly available profile data of LinkedIn makes it ineligible in making use of the existing tactics in fake profile identification. For that reason, there is to conduct distinctive study on deciding on systems for fake profile identification in LinkedIn. Shalinda Adikari and Kaushik Dutta researched and identified the minimal set of profile data that are crucial for picking out false profiles inLinkedIn and labeled the appropriate knowledge mining procedure for such project.

Z. Halim et al. Proposed spatio-temporal mining on social network to determine circleof customers concerned in malicious events with the support of latent semantic analysis. Then compare the results comprised of spatial temporal co incidence with that of original organization/ties with in social network, which could be very encouraging as the organization generated by spatio-temporal co-prevalence and actual one are verynearly each other. Once they set the worth of threshold to right level we develop the number of nodes i.e. Actor so that they are able to get higher photo. Total, scan indicatethat Latent Semantic Indexing participate in very good for picking out malicious contents, if the feature set is competently chosen. One obvious quandary of this technique is how users pick their function set and the way rich it's. If the characteristicset is very small then most of the malicious.

2.3 DISADVANTAGES OF EXISTING SYSTEM

Existing systems for fake profile identification in social media have several disadvantages and challenges:

False Positives: Automated detection methods may incorrectly flag legitimate users as fake profiles, resulting in unintended consequences such as account restrictions or bans.

Evolving Tactics: Those creating fake profiles continually adapt their strategies, making it difficult for detection systems to keep pace with new and sophisticated tactics.

Privacy Concerns: The extensive data analysis often required to identify fake profiles raises privacy concerns and ethical dilemmas regarding user data usage.

User Reporting Bias: Relying solely on user reports can lead to inconsistenciesand biases, as some users may report others for personal reasons or misunderstandings.

Resource Intensive: Maintaining effective fake profile detection systems requires substantial computational resources and human moderation, which can be costly.

Multilingual : Identifying fake profiles in multiple languages and cultural contexts can be complex, as language nuances and behaviors vary.

Scale: As social media platforms grow, the sheer number of profiles andactivities makes it challenging to track and identify fake profiles effectively.

Detection Lag: Identifying fake profiles often lags behind their creation, allowing them to engage in malicious activities before being caught.

Socks and Proxies: Sophisticated actors can use VPNs and proxies to hide theirtrue location and evade IP-based detection.

Resource Intensive: Maintaining effective fake profile detection systems requires substantial computational resources and human moderation, which can be costly.

Multilingual and Multicultural Challenges: Identifying fake profiles in multiple languages and cultural contexts can be complex, as language nuances and behaviors vary.

Scale: As social media platforms grow, the sheer number of profiles andactivities makes it challenging to track and identify fake profiles effectively.

Detection Lag: Identifying fake profiles often lags behind their creation, allowing them to engage in malicious activities before being caught.

Socks and Proxies: Sophisticated actors can use VPNs and proxies to hide theirtrue location and evade IP-based detection.

AI-generated Content: The rise of AI-generated content, including deepfakesand text generated by GPT-like models, complicates the identification process, as fake profiles can generate highly convincing content.

User Deception: Fake profiles can behave similarly to genuine users, making it challenging to differentiate based on behavior alone.

Platform Hopping: Perpetrators can easily move between social media platforms, avoiding detection on one platform by creating fake profiles on another.

Overcoming these limitations and challenges is an ongoing endeavor, involving a combination of advanced technology, user education, and community involvement,to maintain the authenticity and security of social media platforms.

Safety measures: The extensive data analysis often required to identify fake profiles raises privacy concerns and ethical dilemmas regarding user data usage.

Multicultural Challenges: Identifying fake profiles in multiple languages and cultural contexts can be complex, as language nuances and behaviors vary.

AI-generated Content: The rise of AI-generated content, including deepfakesand text generated by GPT-like models, complicates the identification process, as fake profiles can generate highly convincing content.

2.4 PROPOSED SYSTEM

On this paper we presented a machine learning & natural language processing system to observe the false profiles in online social networks. Moreover, we are adding the SVM classifier and naïve bayes algorithm to increase the detection accuracy rate of the fake profiles. An SVM classifies information by means of finding the exceptional hyperplane that separates all information facets of 1 type fromthose of the other classification. The best hyperplane for an SVM method that the

one with the biggest line between the two classes. An SVM classifies data through discovering the exceptional hyperplane that separates all knowledge facets of one category from those of the other class. The help vectors are the info aspects which are closest to the keeping apart hyperplane.Naive Bayes algorithm is the algorithm that learns the chance of an object with designated features belonging to a unique crew/category. In brief, it's a probabilistic classifier. The Naive Bayes algorithm is called "naive" on account that it makes the belief that the occurrence of a distinct feature is independent of the prevalence of other aspects. For illustration, if we're looking to determine false profiles based on its time, date of publication or posts, language and geoposition. Even if these points depend upon each and every different or on the presence of the other facets, all of these properties in my view contribute to the probability that the false profile.

2.5 ADVANTAGES OF PROPOSED SYSTEM

The advantages of fake profile identification in social media are numerous:

Enhanced User Trust: Users can have more confidence in the authenticity of interactions and information on the platform, leading to improved trust in the social network.

User Safety: Identifying and removing fake profiles can help protect users from scams, harassment, and potentially harmful interactions, contributing to a safer online environment.

Quality Content: By reducing the presence of fake profiles, the platform can maintain the quality of content and information shared by users

Privacy Protection: Identifying fake profiles helps prevent the misuse of personal information, safeguarding user privacy.

Improved User Experience: Removing fake profiles results in a more enjoyableand relevant user experience with less spam and deceptive activity.

Mitigation of Online Harassment: Fake profile identification can reduce online harassment, making the platform more welcoming and respectful.

Compliance with Regulations: Platforms can ensure they comply with legal and regulatory requirements regarding the prevention of fake profiles and fraudulent activities.

Integrity of Business and Advertising: Accurate user data and interactions foster trust in advertising and analytics, supporting the platform's business model.

Reputation Management: Effectively dealing with fake profiles can enhance the platform's reputation for user safety and authenticity.

Technological Advancements: Advancements in technology, such as machine learning and natural language processing, can be leveraged to improve detectionaccuracy and efficiency.

Community Engagement: Allowing users to participate in identifying and reporting fake profiles fosters a sense of community and involvement.

Global Considerations: Addressing fake profiles from diverse regions and languages reflects the global nature of social media platforms.

By achieving these advantages, social media platforms can create a more trustworthy, secure, and enjoyable environment for their users, which in turn supports their growth and success.

CHAPTER 3

ANALYSIS

3.1 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).

3.2 OPERATING SYSTEM: WINDOWS 7

In the realm of fake profile detection, users of Windows 7 must remain vigilant despite the lack of ongoing support from Microsoft. Detecting and mitigating fake profiles is crucial for safeguarding personal information and preventing potential online scams or identity theft. While Windows 7 may not receive security updates, users can still utilize a combination of proactive measures and third-party security solutions to enhance their defenses. Implementing robust antivirus software and regularly updating it can help detect and remove malicious software that may be used to create or interact with fake profiles. Additionally, leveraging firewalls and intrusion detection systems can provide an added layer of protection against unauthorized access attempts and suspicious activities associated with fake profiles. Educating oneself about common tactics employed by fake profiles, such as fraudulent friend requests or phishing messages, can also help users recognize and avoid
potential threats. Furthermore, staying informed about emerging trends and best practices in fake profile detection through reputable cybersecurity resources and online communities enables users to adapt their strategies

accordingly. Despite the challenges posed by the lack of official support for Windows 7, proactive engagement with cybersecurity measures and continued vigilance remain essential in mitigating the risks associated with fake profile detection in the digital landscape.

Windows 7 is a personal computer operating system developed by Microsoft.It is a part of the Windows NT family of operating systems. Windows 7 was released to manufacturing on July 22, 2009 and became generally available on October 22, 2009, less than three years after the release of its predecessor,Windows Vista. Windows 7's server counterpart, Windows Server 2008 R2, was released at the same time.

Windows 7 was primarily intended to be an incremental upgrade to the operatingsystem, intended to address Windows Vista's poor critical reception while maintaining hardware and software compatibility. Windows 7 continued improvements on Windows Aero (the user interface introduced in Windows Vista) with the addition of a redesigned taskbar that allows applications to be "pinned" to it, and new window management features.

Other new features were added to the operating system, including libraries, the new file sharing system Home Group, and support for multi touch input and maintenance information, and tweaks were made to the User Account Control system to make it less intrusive. Windows 7 also shipped with updated versions of several stock applications, including Internet Explorer 8, Windows Media Player, and windowsmedia center.

3.3 PYTHON IDLE

IDLE has the following features

- coded in 100% pure Python, using the **tkinter** GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent,call tips, auto completion, and other features
- search within any window, replace within editor windows, and search throughmultiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global andlocal namespaces
- configuration, browsers, and other dialogs

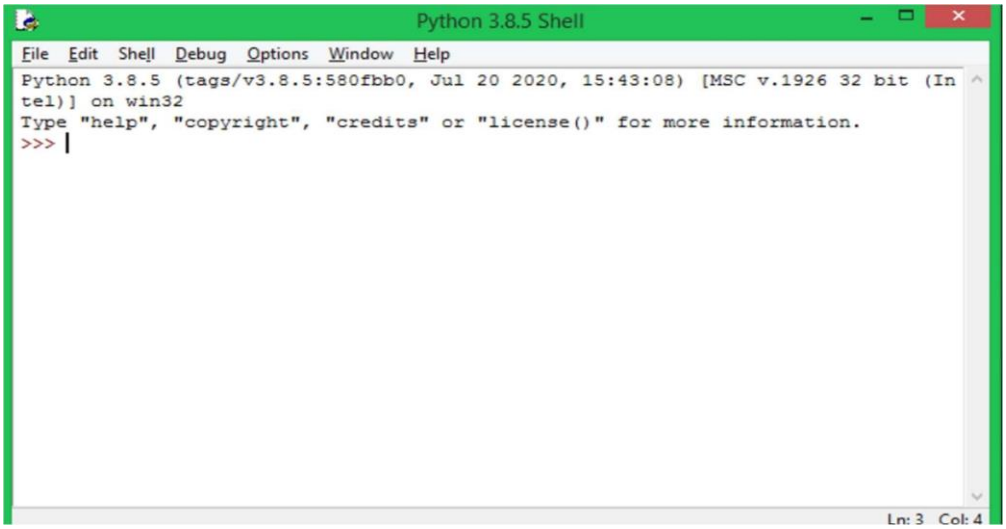


Fig 3.1 Python IDLE

Python IDLE (Integrated Development and Learning Environment) serves as a fundamental tool for both beginner and experienced Python programmers. Its simplicity and accessibility make it an excellent platform for learning Python basics, experimenting with code snippets, and prototyping projects. With features like syntax highlighting, code completion, and interactive mode, IDLE facilitates a smooth development process, allowing users to write, test, and debug Python code seamlessly. Furthermore, its integration with the Python standard library and support for various third-party libraries enable developers to extend its capabilities for a wide range of applications, from scientific computing to web development. Despite its lightweight nature, Python IDLE remains a powerful tool for both educational purposes and professional development, fostering a welcoming environment for Python enthusiasts of all skill levels to explore and expand their programming horizons.

3.4 FLOW CHART

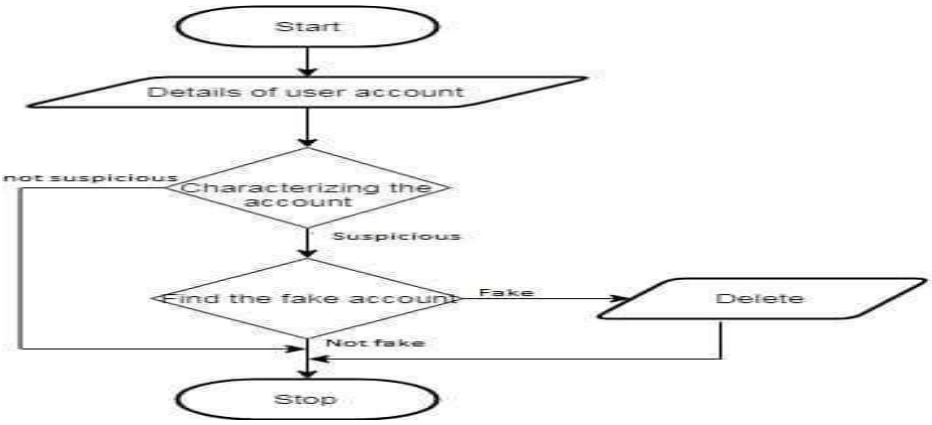


Fig 3.2 Flow chart

The flow chart for identifying fake profiles on social media employs a systematic approach leveraging machine learning and NLP. It begins with data collection, followed by preprocessing and feature extraction. Through model training and evaluation, profiles are classified based on established thresholds. Post-processing ensures accuracy, balancing precision and recall. Ultimately, the flow chart offers a structured framework for efficiently combating fraudulent accounts and safeguarding online communities.

3.5 ALGORITHMS

Decision Tree

Decision Tree Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

The decision tree algorithm is a powerful tool in machine learning and data mining that is widely used for classification and regression tasks. It works by recursively partitioning the data into subsets based on the values of different attributes, with the goal of maximizing the purity of each subset. At each step of the partitioning process, the algorithm selects the attribute that best separates the data into homogeneous groups, typically using metrics such as information gain or Gini impurity. This process continues until a stopping criterion is met, such as reaching a maximum tree depth or having all instances within a subset belong to the same class. The resulting decision tree can be interpreted as a series of if-else rules that describe the decision-making process for classifying new instances. Decision trees have several advantages, including their interpretability, ease of use, and ability to handle both numerical and categorical data. However, they are also prone to overfitting, especially when the trees are deep or the dataset is noisy. Techniques such as pruning, which involves removing parts of the tree that do not improve its performance on a validation set, can help mitigate this issue. Overall, the decision tree algorithm offers a versatile and intuitive approach to classification and regression tasks, making it a valuable tool in the machine learning toolkit.

Random forest

This classifier classifies a collection of decision trees to a subset of randomly generated training sets. Then it augments the likes from decision sub trees to known subclasses of handling objects for tests. Random forest will generate NA missing values for attributes to increase accuracy for larger sets of data. If more number of trees, it doesn't allow to trees to fit model.

The Random Forest algorithm is a powerful ensemble learning technique that is widely used for both classification and regression tasks in machine learning. It operates by constructing a multitude of decision trees during the training phase and outputs the mode of the classes (classification) or the average prediction (regression) of the individual trees. Each decision tree in the Random Forest is trained on a random subset of the training data and a random subset of the features, which introduces variability and reduces the risk of overfitting. The algorithm aggregates the predictions of multiple trees, resulting in a more robust and accurate model. Random Forests are highly flexible and can handle high-dimensional datasets with mixed data types, including categorical and numerical features. They are also robust to noisy data and outliers due to the averaging effect of multiple trees. Additionally, Random Forests provide important insights into feature importance, allowing users to identify the most relevant features in the dataset. However, Random Forests may suffer from high computational costs, especially with large datasets and a large number of trees. Furthermore, the interpretability of Random Forest models may be limited compared to simpler models such as decision trees. Despite these limitations, Random Forests are widely used in various applications such as image classification, bioinformatics, and financial modeling, owing to their robustness, accuracy, and ease of use.

Support Vector Machine

Support Vector Machine is a binary classification algorithm that finds the maximum separation hyper plane between two classes. It is a supervised learning algorithm that gives enough training examples, divides two classes fairly well and classifies new examples .It offers a principle approach to machine learning problems because of their mathematical foundation in statistical learning theory. SVM constructs their solution as a weighted sum of SVs , which are only a subset of the training input .It is effective in cases where the number of dimensions is greater than the number of samples given. The Support Vector Machine (SVM) algorithm is a powerful supervised learning technique used for both classification and regression tasks. SVM aims to find the hyperplane that best separates data points of different classes while maximizing the margin, which is the distance between the hyperplane and the closest data points (support vectors).

This margin maximization principle allows SVM to have a strong generalization ability, making it effective in dealing with complex and high-dimensional datasets. SVM is particularly well-suited for datasets with clear margin of separation between classes and can handle both linearly separable and non-linearly separable data

through the use of kernel functions. Kernel functions enable SVM to implicitly map the input data into a higher-dimensional space where it becomes linearly separable, thus allowing for the construction of non-linear decision boundaries. SVM has several advantages, including its ability to handle high-dimensional data, resistance to overfitting, and effectiveness in small to medium-sized datasets. However, SVM's performance may degrade with large datasets due to its computational complexity, especially when using non-linear kernels. Moreover, SVM does not provide probabilistic outputs directly, which may limit its interpretability in certain applications. Nonetheless, SVM remains a popular and versatile algorithm in machine learning, with applications in areas such as image classification, text categorization, and bioinformatics.

CHAPTER 4
DESIGN

4.1 SYSTEM MODELS

System models play a crucial role in the design, development, and understanding of fake profile identification systems in social media. Thesemodels provide a visual and conceptual framework for stakeholders to grasp the architecture, components, and interactions within the system. Use case diagrams depict the user interactions, entity-relationship diagrams define the data structure, and deployment diagrams illustrate the system's physical layout. Activity and sequence diagrams showcase the flow of activities and the sequence of events involved in identifying fake profiles, while class diagrams offer insights into the object-oriented design. Additionally, feedback loop models demonstrate how feedback from users and actions taken contribute to system improvement. Overall, these system models facilitate effective communication, collaboration, andproblem-solving during system development and operation, aiding in the creation of more secure and reliable social media platforms.system modelsare used to describe, analyze, and design software systems. They provide ahigh-level view of the system, allowing stakeholders to understand the system's behaviour, structure, and interaction with its environment.

4.2 SYSTEM REQUIREMENTS

H/W System Configuration:-

- Processor - Pentium –IV
- RAM - 4 GB (min)
- Hard Disk - 20 GB
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - SVGA

Clear system requirements are pivotal for developing effective fake profile identification systems on social media. Hardware specifications, including a Pentium-IV processor, 4GB RAM, and 20GB hard disk space, ensure optimal system performance. Standard input devices and SVGA monitors facilitate user-friendly interaction. These requirements are essential for creating a robust and efficient system capable of detecting and mitigating fraudulent activities on social media platforms.

4.3 CONCEPTUAL DIAGRAMS

- **ER DIAGRAM**

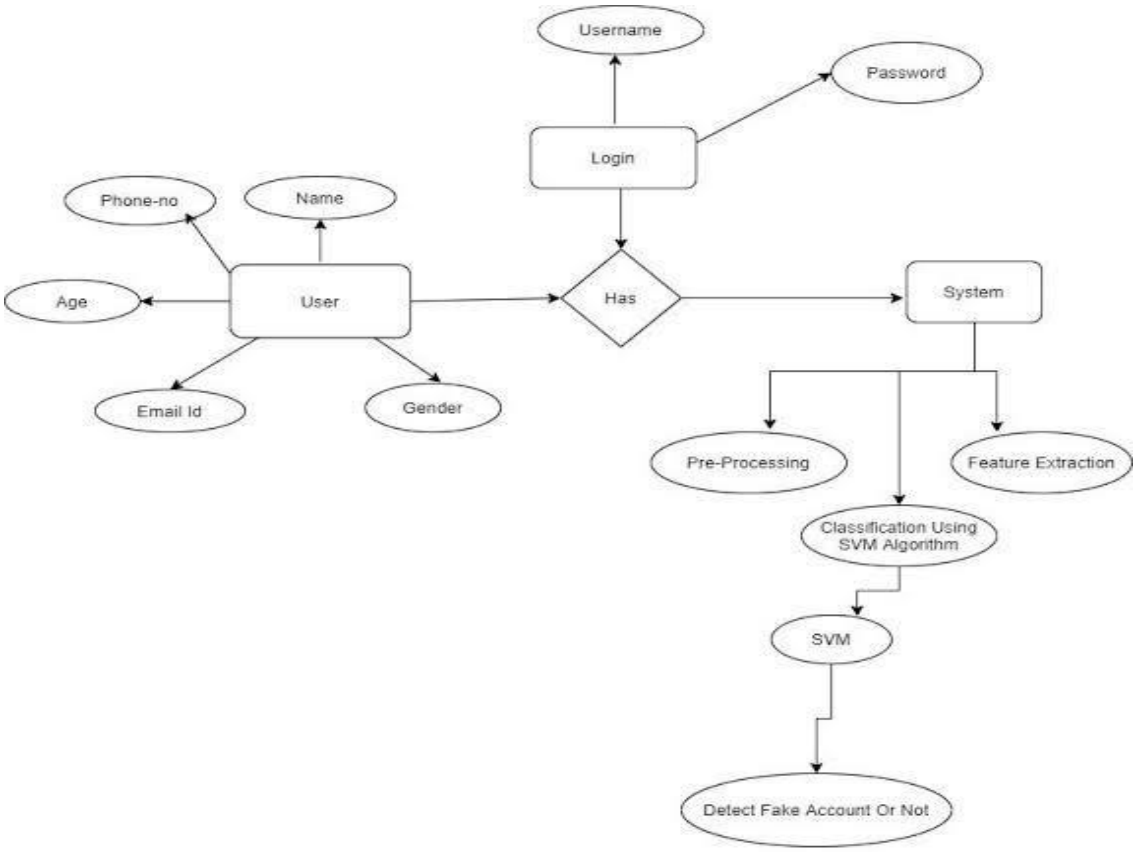


Fig 4.1 ER Diagram

In an ER diagram for a fake profile identification system on social media, several key entities, attributes, and relationships are depicted to capture the system's data model comprehensively. Entities include User Profiles, representing individual accounts, Posts for shared content, Interactions for user engagements like likes or comments, and Reports for flagged content. Attributes within these entities encompass details such as usernames, post content, interaction types, and report reasons. Relationships between entities are defined, indicating that a user can create multiple posts but each post belongs to only one user (One-to-Many between User Profile and Posts). Similarly, a user can have multiple interactions with posts, establishing another One-to-Many relationship between User Profile and Interactions. Additionally, reports are associated with users who flag suspicious profiles or content. By incorporating these elements, the ER diagram provides a structured overview of the data model, facilitating the understanding and development of the fake profile identification system on social media.

- **USE CASE DIAGRAM**

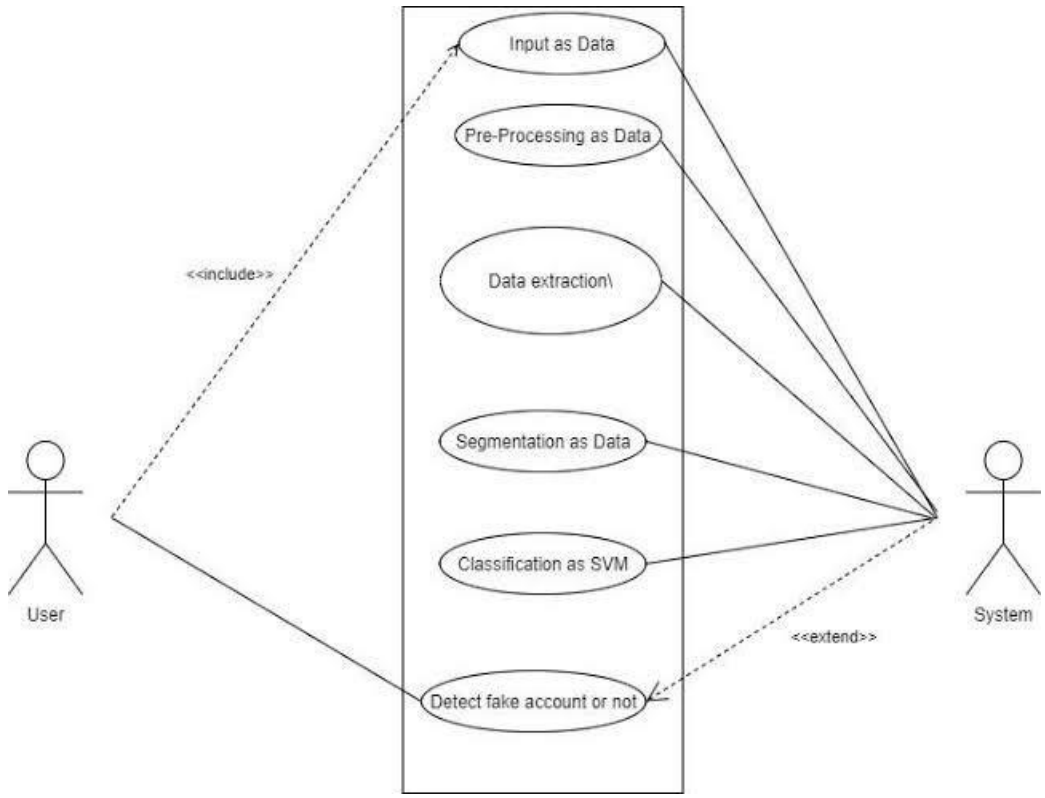


Fig 4.2 USECASE DIAGRAM

In a use case diagram for a fake profile identification system on social media, various actors and their interactions with the system are delineated to depict the system's functionality from a user's perspective. The primary actor, typically a registered user, initiates several use cases, including Profile Creation, Post Sharing, Interaction (like, comment), and Report Submission. Additionally, secondary actors such as Administrators or Moderators may be involved in reviewing flagged content or taking actions against fake profiles. Each use case represents a specific action or functionality that the system offers to its users, such as creating a profile, sharing posts, engaging with content, or reporting suspicious activity. Relationships between actors and use cases illustrate how different users interact with the system to achieve their goals. By encapsulating these interactions, the use case diagram provides a clear and concise representation of the system's functionalities, aiding in requirements analysis, system design, and communication among stakeholders. The use case diagram also outlines scenarios for user interactions, including authentication processes and profile management functionalities. It highlights the system's role in facilitating user engagement while emphasizing the importance of user feedback through reporting mechanisms. Ultimately, the use case diagram serves as a valuable tool for understanding user-system interactions and guiding the development of a robust fake profile identification system on social media.

➢ CLASS DIAGRAM

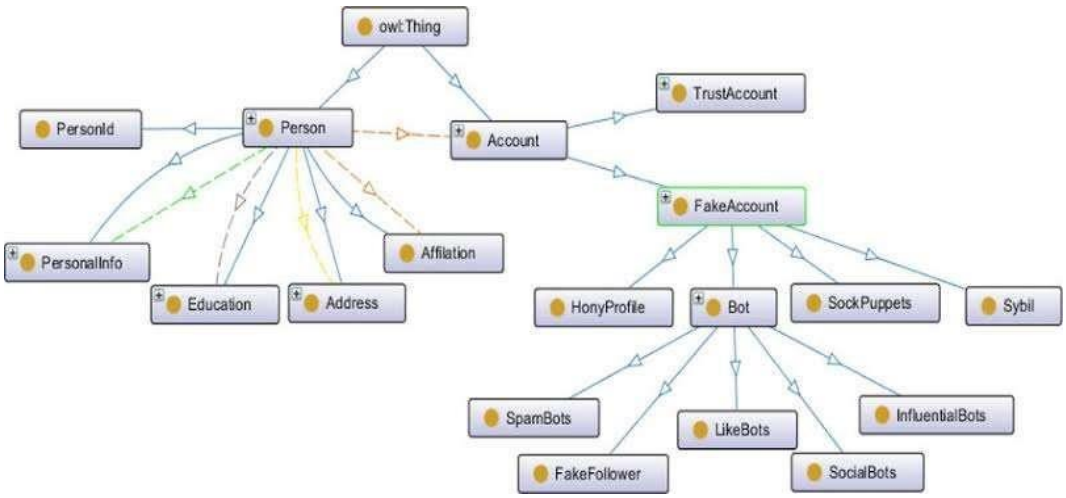


Fig 4.3 Class Diagram

In a class diagram for a fake profile identification system on social media, the system's structure and the relationships between its various components are depicted using classes and their attributes and methods. Key classes include User, Post, Interaction, Report, and Administrator. Attributes of the User class may include username, email, and account status, while the Post class could have attributes such as content and timestamp. Methods within these classes could encapsulate functionalities like creating a post or submitting a report. Associations between classes illustrate how they interact, such as a User creating multiple Posts or an Administrator reviewing Reports. Additionally, inheritance relationships may exist, such as an Administrator inheriting properties and behaviors from a User class. By capturing these elements, the class diagram provides a blueprint for the system's design and implementation, facilitating collaboration among developers and ensuring a coherent architecture for the fake profile identification system on social media.

- **SEQUENCE DIAGRAM**

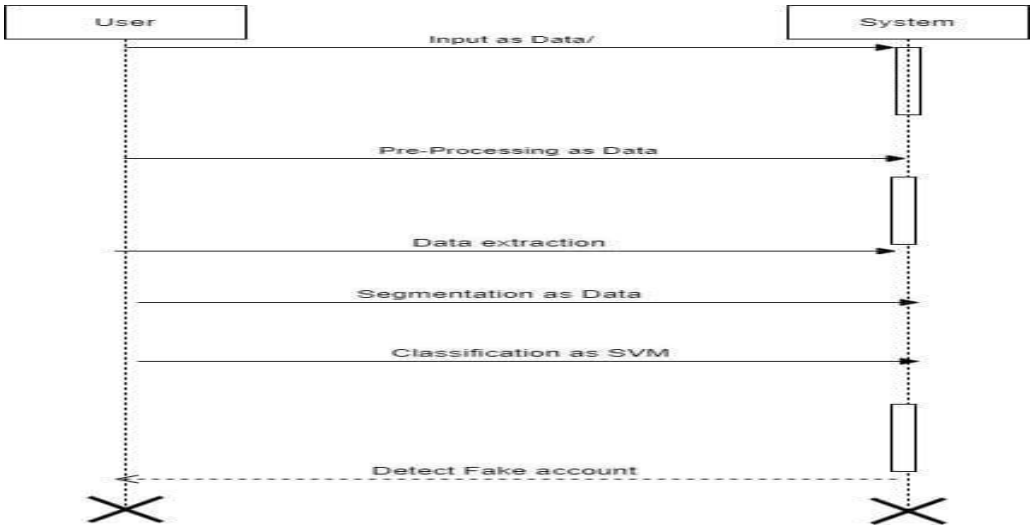


Fig 4.4 Sequence Diagram

It describes sequence diagram outlines the series of interactions and decisions involved in the process of fake profile identification, starting with user reporting and culminating in actions taken on reported profiles. In a sequence diagram for a fake profile identification system on social media, the interactions between various components of the system are illustrated over time. The diagram typically starts with a user initiating an action, such as creating a post or submitting a report. This triggers a sequence of messages exchanged between objects or classes involved in the process, such as the User object, Post object, Interaction object, and Administrator object. Each message represents a specific function call or communication between components, showcasing the flow of control and data throughout the system. For instance, the sequence might depict a User creating a Post, followed by interactions such as likes or comments, and eventually leading to the detection of suspicious activity and the involvement of an Administrator for further action. Through this visual representation, the sequence diagram provides insights into the dynamic behavior of the system, helping stakeholders understand the sequence of events and interactions necessary for identifying and addressing fake profiles on social media.

➢ DEPLOYMENT DIAGRAM

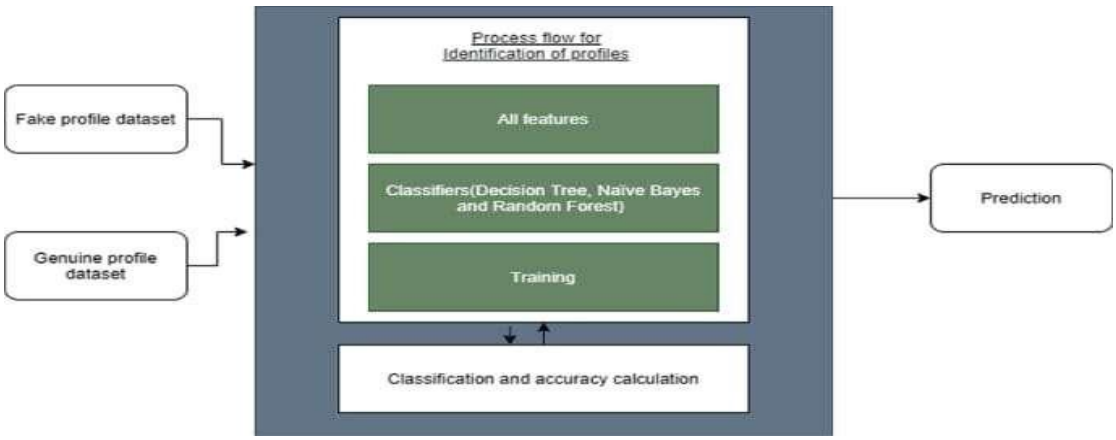


Fig 4.5 Deployment Diagram

In a deployment diagram for a fake profile identification system on social media, the physical arrangement of system components across hardware nodes is depicted. Nodes represent servers, databases, and client devices, while components signify software modules deployed on these nodes. Connections between nodes and components illustrate communication pathways, such as HTTP requests. This diagram offers insights into system scalability and performance, aiding administrators in efficient deployment and maintenance of the system. In a deployment diagram for a fake profile identification system on social media, the physical deployment of system components across hardware nodes or computing devices is illustrated to depict the system's architecture and deployment environment. Nodes represent hardware devices such as servers, databases, or client machines, while components represent software artifacts or modules deployed on these nodes. For example, nodes could include web servers hosting the application, databases storing user data, and client devices accessing the system through web browsers or mobile applications. Components deployed on these nodes may include the application server, database management system, and user interface modules. Connections between nodes and components indicate communication pathways, such as HTTP requests between client devices and web servers or database queries between the application server and database nodes. By visualizing the distribution of system components across physical infrastructure, the deployment diagram aids in understanding system scalability, reliability, and performance considerations. It provides valuable insights for system administrators and developers responsible for managing and maintaining the fake profile identification system on social media, ensuring efficient deployment and operation in real-world environments.

- **DATA FLOW DIAGRAM**

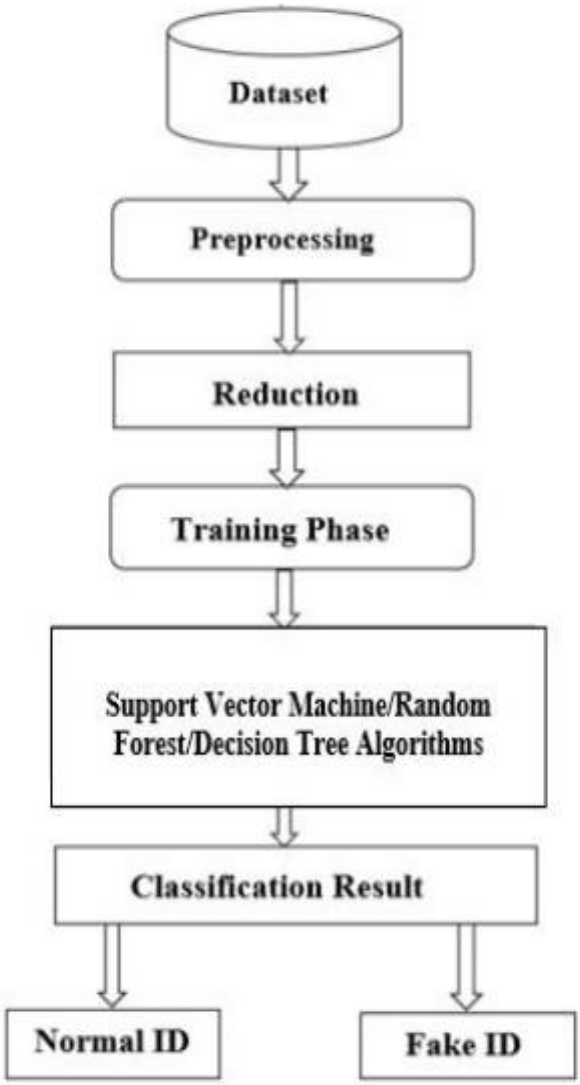


Fig 4.6 Data Flow Diagram

A DFD helps in understanding the flow of data within a system, identifying potential bottlenecks, and clarifying system requirements. It provides a high- level overview of how the system operates without delving into implementation details. DFDs are often used as a communication tool between system analysts, stakeholders, and developers to ensure everyone has a common understanding of the system's data flow.

CHAPTER 5
SOFTWARE ENVIRONMENT

5.1 TECHNOLOGIES USED

• Python:

Python is an interpreter, high-level, general-purpose programming language. Python is simple and easy to read syntax emphasizes readability and thus reduces system maintenance costs. Python supports modules and packages, which promote system layout and code reuse. It saves space but it takes a rather higher time when its code is compiled. Indentation must be taken care while coding. Python has many inbuilt library functions that can be used easily for working with machine learning algorithms. All the necessary python libraries must be pre-installed using “pip” command. Software used in the project include following Libraries, Modules and Packages. Web Application Framework or just Web Framework represents a set of libraries and modules that permits an internet application developer to write down applications without having to bother about low level details such as protocols, thread management, etc. Flask may be a web application framework written in Python. It is developed by Armin Ronacher, who leads a world group of Python enthusiasts named Pocco. Flask is predicated on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects. Python 2.6 or higher is usually required for the installation of Flask. The Flask and its dependencies work well with Python 3. Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application

• NumPy:

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python. In Alternatives of arrays in Python are lists, but they take a long time to execute. The goal of NumPy is to offer an array items that are up to 50 times quicker than conventional Python lists. The NumPy collection is referred to as ndarray, and it offers a number of supporting methods that make using ndarray very simple. In data science, where efficiency and resources are crucial, arrays are used a lot. Enter the following command to install numpy: pipinstallnumpy

• Pandas:

Python's Pandas module is used for modifying data collections. It has functions for data exploration, cleansing, analysis, and manipulation. Wes McKinney came up with the term "Pandas" in 2008, and it refers to both "Panel Data" and "Python Data Analysis". With the aid of Pandas, we can examine large data sets and draw predictions based on statistical principles. Pandas can organize disorganized data sets, making them readable and useful . In data science, relevant info is crucial. Pandas is an open-source tool designed primarily for dealing quickly and logically with relational or annotated data. It offers a range of data types and functions for working with time

series and numerical data. The NumPy library serves as the foundation for this library. Pandas move quickly and has high performance and productivity for users.

• Sklearn:

Scikit-learn, also known as sklearn, is a popular machine learning library for Python. It provides a wide range of supervised and unsupervised learning algorithms and tools for data preprocessing, feature selection, model selection, and evaluation. Scikit-learn is built on top of other popular libraries such as NumPy, SciPy, and matplotlib, and has a simple and consistent API, making it easy to learn and use. Scikit-learn supports a variety of classification, regression, and clustering algorithms, including linear and logistic regression, decision trees, random forests, support vector machines, k-nearest neighbors, k-means clustering, and more. It also provides a range of tools for feature selection and dimensionality reduction, such as Principal Component Analysis (PCA), and for model selection and evaluation, such as crossvalidation and grid search. Scikit-learn is widely used in both academia and industry and has a large and active community of developers and users who contribute to its development and documentation .

• Skimage :

Scikit-image or skimage is an open-source image processing library for Python that is built on top of NumPy, SciPy, and matplotlib. It provides various functions for image processing, manipulation, and analysis, including image filtering, segmentation, feature extraction, and more. One of the notable features of skimage is its ability to work with multidimensional images, making it suitable for processing not only 2D images but also 3D and even 4D images. One of the most useful functions in skimage is skimage.io.imwrite(), which allows you to save an image to disk in various file formats, including PNG, JPG, TIFF, BMP, and more. This function takes two arguments: the filename to save the image to, and the NumPy array representing the image data. You can also specify various options such as the image format, the image quality, and more. Overall, the skimage.io.imwrite() function is a simple and convenient way to save images in your Python code, and it integrates seamlessly with other functions in skimage for image processing and analysis.

• TensorFlow:

TensorFlow is an open-source machine learning framework developed by Google Brain. It is widely used for building and training machine learning models, supporting various neural network architectures. TensorFlow includes a high-level neural network library called Keras, simplifying model development. It is known for its flexibility, enabling deployment on various platforms, and has an active community providing support and resources. TensorFlow Lite is a version designed for mobile and edge devices with limited resources. TensorBoard, a visualization tool, aids in model understanding and optimization. TensorFlow is written in Python and integrates well with other machine learning libraries, making it a versatile choice for diverse applications.

• **Streamlit:**

Streamlit is an open-source Python library used for creating web applications with minimal effort and code. It simplifies the process of turning data scripts into interactive web apps by providing a high-level API. With Streamlit, developers can build data-driven applications quickly, using simple Python scripts to create UI components like sliders, buttons, and charts. The library automatically updates the app in real-time as users interact with the interface. Streamlit is known for its user-friendly syntax, making it accessible to those without extensive web development experience. It has gained popularity for prototyping and deploying data centric applications.

5.2 PYTHON HISTORY

• Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991. • Python's design philosophy emphasizes code readability. • Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3

The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported. Python interpreters are available for many operating systems. A global community of programmers develops and maintains C Python, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and C Python development.

5.3 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 sometimes termed the off-side rule, which some other languages share, but in most languages indentation doesn't have any semantic meaning.

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 alocal 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 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 def statement, which defines a function or method.

● The with statement, from Python 2.5 released in September 2006, 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 break statement, exits from the loop.
- The continue statement, skips this iteration and continues with the next item.
- 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 tought 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.

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

5.3 PYTHON IDENTIFIERS:

• A Python identifier is a name used to identify a variable, function, class, module or other object.

• An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

• Python does not allow punctuation characters such as @, $, and % within identifiers.

• Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python. Here are naming conventions for Python identifiers –

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.

5.4 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 (or integerdivision) // and floating point/division. Python also added the operator forexponentiation.
- From Python 3.5, the new @ infix operator was introduced. It is intended to be usedby libraries such as NumPy for matrix multiplication.
- From Python 3.8, the syntax :=, called the 'walrus operator' was introduced. It assignsvalues to variables as part of a larger expression.

● 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 thanthe 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.

● Python makes a distinction between lists and tuples.

● Lists are written as [1, 2, 3], aremutable, and cannot be used as the keys of dictionaries.

● 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 modifytheir contents, but rather produces a new tuple containing the elements of both provided tuples.

Python features sequence unpacking wherein 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 various kinds of string literals:

Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perlinfluenced 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.

5.5 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).

5.6 APPLICATIONS OF PYTHON:

As mentioned before, Python is one of the most widely used language over the web. I'm going to list few of them here:

● **Easy-to-learn** − Python has few keywords, simple structure, and a clearly definedsyntax. 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 allowsinteractive 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.

● **Databases** − Python provides interfaces to all major commercial databases.

● **GUI Programming** − Python supports GUI applications that can be created and portedto 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 shellscripting.

5.7 INSTALLATION STEPS OF PYTHON:

Installing and using Python on Windows 10 is very simple. The installation procedure involves just three steps:

● Download the binaries

● Run the Executable installer

● Add Python to PATH environmental variables

To install Python, you need to download the official Python executable installer. Next, you need to run this

installer and complete the installation steps. Finally, you can configure the PATH variable to use python from the command line.

Step 1: Download the Python Installer binaries

● Open the official Python website in your web browser. Navigate to the Downloads tabfor Windows.

● Choose the latest Python 3 release. In our example, we choose the latest Python 3.7.3 version. Click on the link to download Windows x86 executable installer if you are using a 32-bit installer.

● In case your Windows installation is a 64-bit system, then download Windows x86-64executable installer.

Step 2: Run the Executable Installer 1. Once the installer is downloaded, run the Python installer. 2. Check the Install launcher for all users check box. Further, you may check the Add Python 3.7 to path check box to include the interpreter in the execution path.

Figure 5.1: Installing python

The Fig 5.1 describes install launcher of python The window Displays a dialog box to install python with version. we need to click on install launcher for all users.

3. Select Customize installation.

Choose the optional features by checking the following check boxes:

1. Documentation
2. pip
3. tcl/tk and IDLE (to install tkinter and IDLE)
4. Python testsuite (to install the standard library test suite of Python)
5. Install the global launcher for `.py` files. This makes it easier to start Python
6. Install for all users

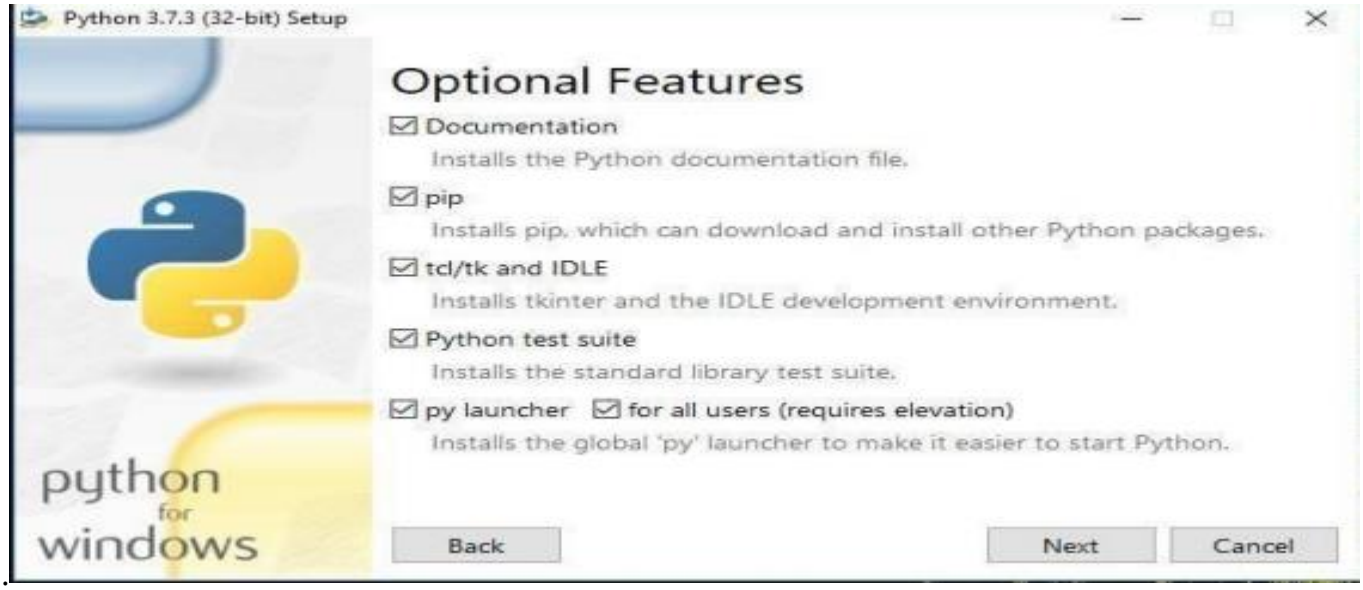


Fig:5.2 Selecting Features

The figure 5.2 describes about selecting optional features in python. We must select the features that we need it displays pip, documentation and few more options as its features Click Next.

4. This takes you to Advanced Options available while installing Python. Here, select the Install for all users and Add Python to environment variables check boxes.Optionally, you can select the Associate files with Python, Create shortcuts for installed applications and other advanced options. Make note of the python installation directory displayed in this step. You would need it for the next step.After selecting the Advanced options, click Install to start installation.

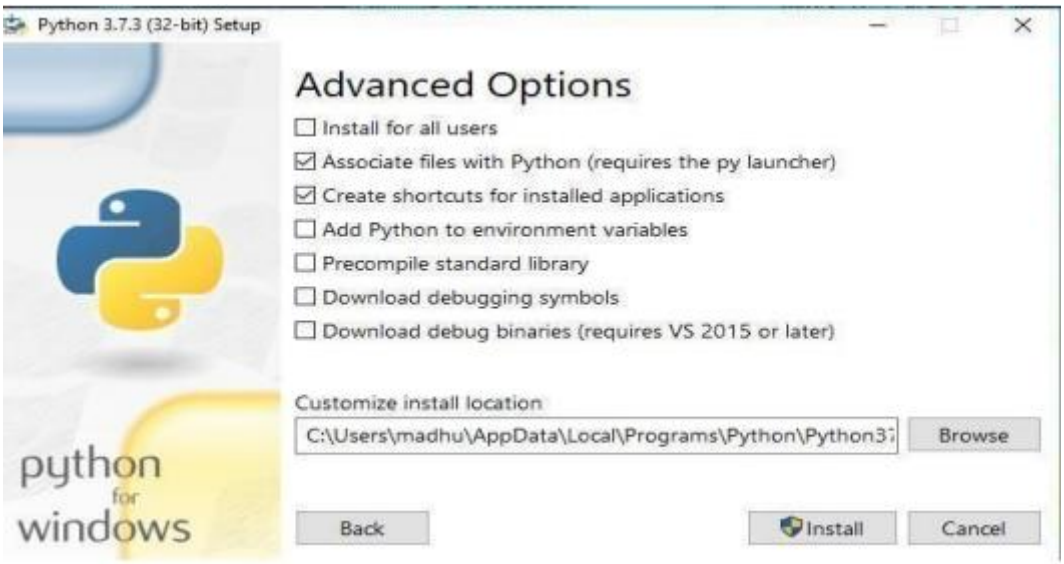


Fig:5.3 Selecting options

The Fig 5.3 describes about selecting advanced options in python

5. Once the installation is over, you will see a Python Setup Successful window.

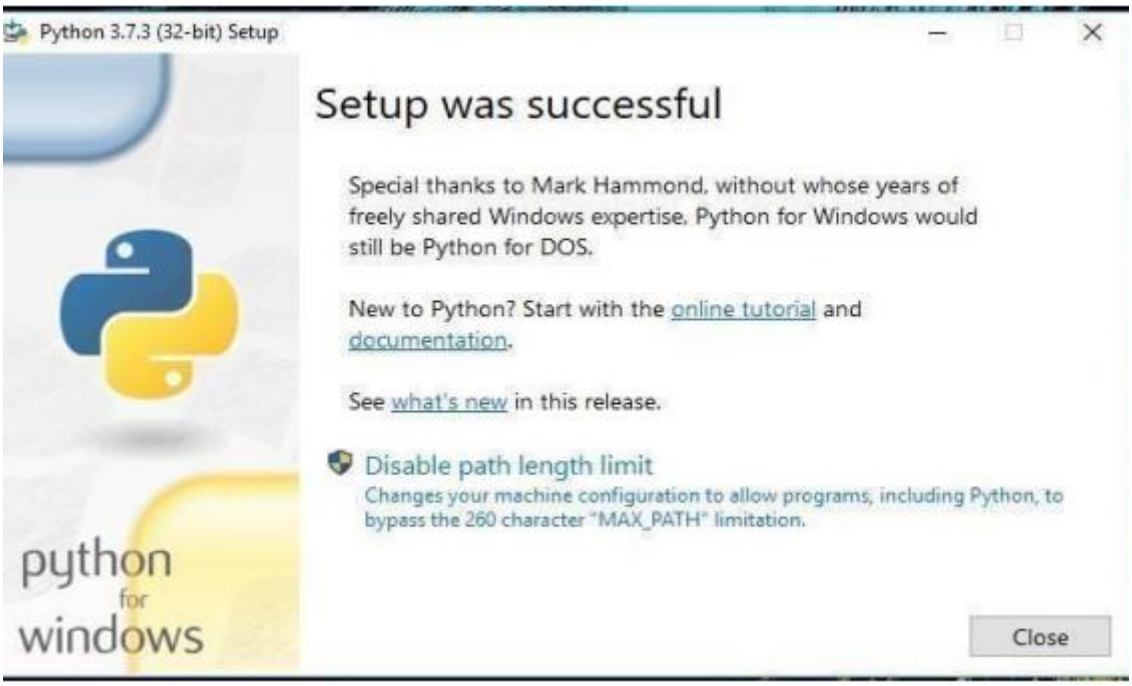


Fig:5.4 Python Setup

The Fig 5.4 describes python setup was successful

Step 3: Add Python to environmental variables

The last (optional) step in the installation process is to add Python Path to the System Environment variables. This step is done to access Python through the command line. In case you have added Python to environment variables while setting the Advanced options during the installation procedure, you can avoid this step. Else, this step is done manually as follows.

In the Start menu, search for “advanced system settings”. Select “View advanced system settings”. In the “System Properties” window, click on the “Advanced” tab and then click on the “Environment Variables” button.

• Locate the Python installation directory on your system. If you followed the steps exactly as above, python will be installed in below locations:

• C:\Program Files (x86)\Python37-32: for 32-bit installation

• C:\Program Files\Python37-32: for 64-bit installation The folder name may be different from “Python37-32” if you installed a different version. Look for a folder whose name starts with Python. Append the following entries to PATH variable as shown below:

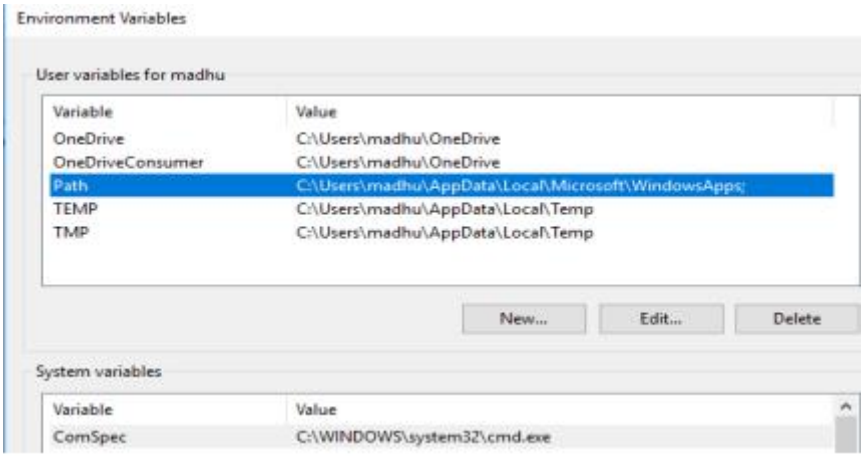


Fig 5.5 Selecting Path

The Fig 5.5 describes how to select a path in python

Fig 5.6 Setting Path

The Fig 5.6 describes how to set a path in python

Step 4: Verify the Python Installation

You have now successfully installed Python 3.7.3 on Windows 10. You can verify if the Python installation is successful either through the command line or through the IDLE app that gets installed along with the installation.

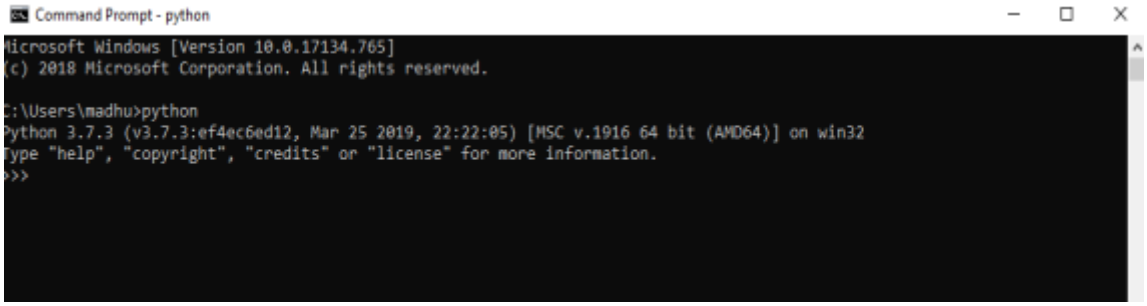


Fig:5.7 Command Prompt

The Fig 5.7 describes about command prompt which is another platform to code

• An alternate way to reach python is to search for “Python” in the start menu and clicking on IDLE (Python 3.7 64-bit). You can start coding in Python using the Integrated Development Environment(IDLE).

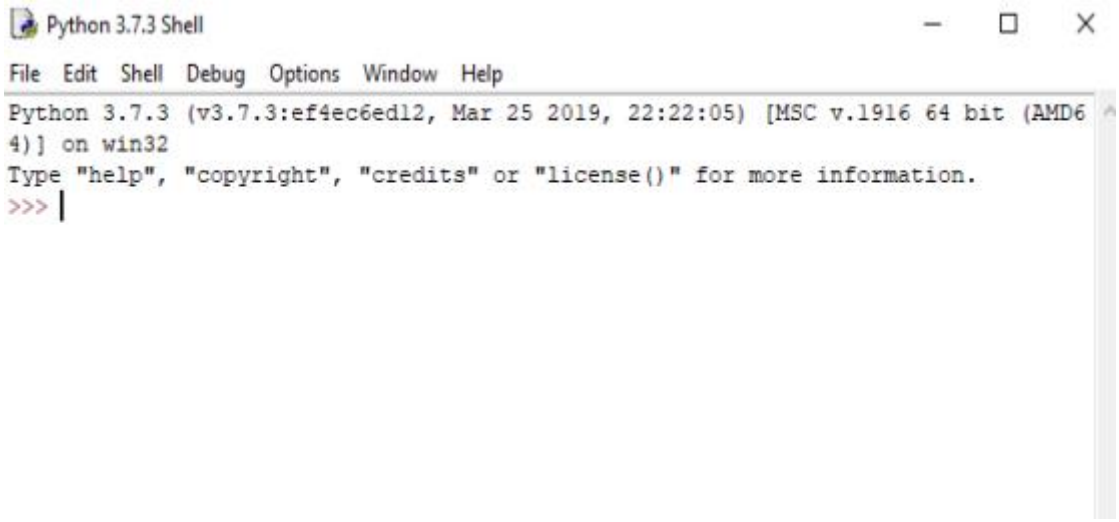


Fig:5.8 Python shell

The Fig 5.8 describe about python shell version 3.7.3 where you can code

Uses:

Since 2003, Python has consistently ranked in the top ten most popular programming languages in the TIOBE Programming Community Index where, as of February 2020, it is the third most popular language (behind Java, and C). It was selected Programming Language of the Year in 2007, 2010, and 2018.

An empirical study found that scripting languages, such as Python, are more productivethan conventional languages, such as C and Java, for programming problems involvingstring manipulation and search in a dictionary, and determined that memory consumption was often "better than Java and not much worse than C or C++".

Large organizations that use Python include Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify and some smaller entities like ILM and ITA. The social news networking site Reddit is written entirely in Python. Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server. With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of complex applications. Pyjs and IronPython can be used to develop the client-side of Ajax-based applications.

SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as Biopython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a notebook interface programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.

Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP, Inkscape, Scribus and Paint Shop Pro, and musical notation programs like scorewriter and capella. GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS. It has also been used in several video games, and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go

Python is commonly used in artificial intelligence projects with the help of libraries likeTensorFlow, Keras, Pytorch and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.Many operating systems include Python as a standard component. It ships with most Linux distributions, AmigaOS 4, FreeBSD (as a package), NetBSD, OpenBSD (as a package) and macOS and can be used from the command line (terminal). Many Linux distributions use installers written in Python: Ubuntu uses the Ubiquity installer, while Red Hat Linux and Fedora use the Anaconda installer. Gentoo Linux uses Python in its package management system, Portage.Python is used extensively in the information security industry, including in exploit development. Most of the Sugar software for the One Laptop per Child XO, now developed at SugarLabs, is written in Python. The Raspberry Pi single-board computer project has adopted Python as its main user-programming language. Due to Python's user-friendly conventions and easy-to-understand language, it is commonly used as an intro language into computing sciences with students. This allows students to easily learn computing theories and concepts and then apply them to other programming languages.

SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as Biopython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a notebook interface programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.

5.8 Modules flow chart

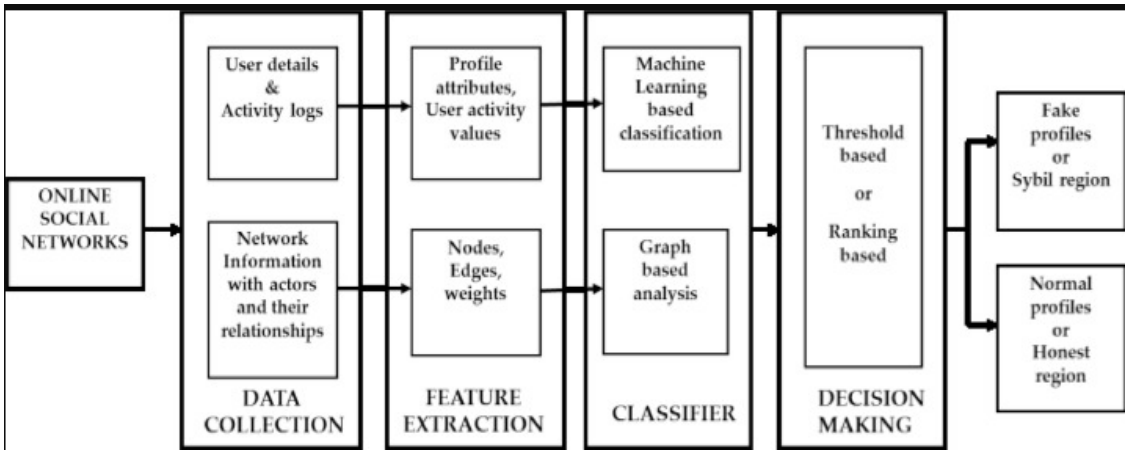


Figure 5.9 flowchart for fake profile detection

Figure 5.1 describes Flowchart of the identifying fake accounts Algorithm: Step 1: Gets the details of the user fake account. Step 2: Identifies the suspicious accounts using the conditional probability model and categorize the account as suspicious.

CHAPTER-6

IMPLEMENTATION

SOURCE CODE

```
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,accuracy_score,roc_curve,confusion_matrix
from import_libs import
train_data_path = 'datasets/Fake-Instagram-Profile-Detection-main/insta_train.csv'
test_data_path = 'datasets/Fake-Instagram-Profile-Detection-main/insta_test.csv'
pd.read_csv(test_data_path)
train_data_path = 'datasets/Insta_Fake_Profile_Detection/train.csv'
test_data_path = 'datasets/Insta_Fake_Profile_Detection/test.csv'
pd.read_csv(train_data_path)
instagram_df_train=pd.read_csv(train_data_path)
instagram_df_train
instagram_df_test=pd.read_csv(test_data_path)
instagram_df_test
from dataset_load import instagram_df_train,instagram_df_test
from import_libs import
instagram_df_train.head()
instagram_df_train.tail()
```

```
instagram_df_test.head()
instagram_df_test.tail()
instagram_df_train.info()
instagram_df_train.describe()
instagram_df_train.isnull().sum()
instagram_df_train['profile pic'].value_counts()
instagram_df_train['fake'].value_counts()
instagram_df_test.info()
instagram_df_test.describe()
instagram_df_test.isnull().sum()
instagram_df_test['fake'].value_counts()
sns.countplot(instagram_df_train['fake'])
plt.show()
sns.countplot(instagram_df_train['private'])
plt.show()
sns.countplot(instagram_df_train['profile pic'])
plt.show()
plt.figure(figsize = (20, 10))
sns.distplot(instagram_df_train['nums/length username'])
plt.show()
plt.figure(figsize=(20, 20))
cm = instagram_df_train.corr()
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax)
plt.show()
sns.countplot(instagram_df_test['fake'])
sns.countplot(instagram_df_test['private'])

sns.countplot(instagram_df_test['profile pic'])
from dataset_load import instagram_df_test,instagram_df_train
from import_libs import
X_train = instagram_df_train.drop(columns = ['fake'])
X_test = instagram_df_test.drop(columns = ['fake'])
print(X_train,X_test)
```

```
y_train = instagram_df_train['fake']
y_test = instagram_df_test['fake']
print(y_train,y_test)
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)
y_train = tf.keras.utils.to_categorical(y_train, num_classes = 2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes = 2)
print(y_train,y_test)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
Training_data = len(X_train)/( len(X_test) + len(X_train) )  100
Testing_data = len(X_test)/( len(X_test) + len(X_train) )  100
print(Training_data, Testing_data)
from import_libs import
from dataset_load import instagram_df_train,instagram_df_test
from data_preprocess_model import X_train,y_train,X_test,y_test
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
model = Sequential()
model.add(Dense(50, input_dim=11, activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(150, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(25, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2,activation='softmax'))
model.summary()
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
epochs_hist = model.fit(X_train, y_train, epochs = 50,  verbose = 1, validation_split = 0.1)
from main_model import model, epochs_hist
from data_preprocess_model import X_test,y_test
```

```
from import_libs import
predicted = model.predict(X_test)
predicted_value = []
test = []
for i in predicted:
    predicted_value.append(np.argmax(i))
for i in y_test:
    test.append(np.argmax(i))
print(classification_report(test, predicted_value))
plt.figure(figsize=(10, 10))
cm=confusion_matrix(test, predicted_value)
sns.heatmap(cm, annot=True)
plt.show()
print(epochs_hist.history.keys())
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])
plt.title('Model Loss Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()
plt.plot(epochs_hist.history['accuracy'])
plt.plot(epochs_hist.history['val_accuracy'])
plt.title('Model Accuracy Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Acc', 'Validation Acc'])
plt.show()
dicts = {
    'Accuracy' : epochs_hist.history['accuracy'],
    'Validation_Accuracy' : epochs_hist.history['val_accuracy'],
    'Loss' : epochs_hist.history['loss'],
    'Validation Loss' : epochs_hist.history['val_loss']
}
```

```
model_training_progress = pd.DataFrame(dicts)
model_training_progress
print(model_training_progress)
def get_avg(lst):
    return sum(lst) / len(lst)
print("Accuracy : ", get_avg(model_training_progress['Accuracy'])  100)
print("Validation Accuracy : ", get_avg(model_training_progress['Validation_Accuracy'])  100)
print("Loss : ", get_avg(model_training_progress['Loss'])  100)
print("Validation Loss : ", get_avg(model_training_progress['Validation Loss'])  100)
```

OUTPUT

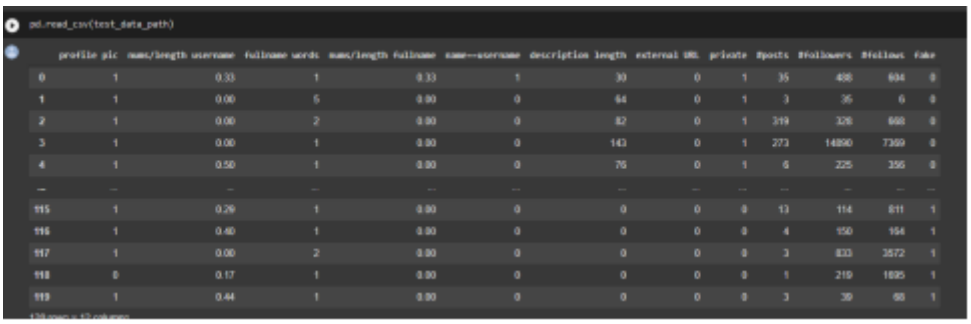


Figure 6.1: Load Data (Pre-processing)

Figure 6.1 describes The data is getting loaded and after loading the data, the data will go under processing.

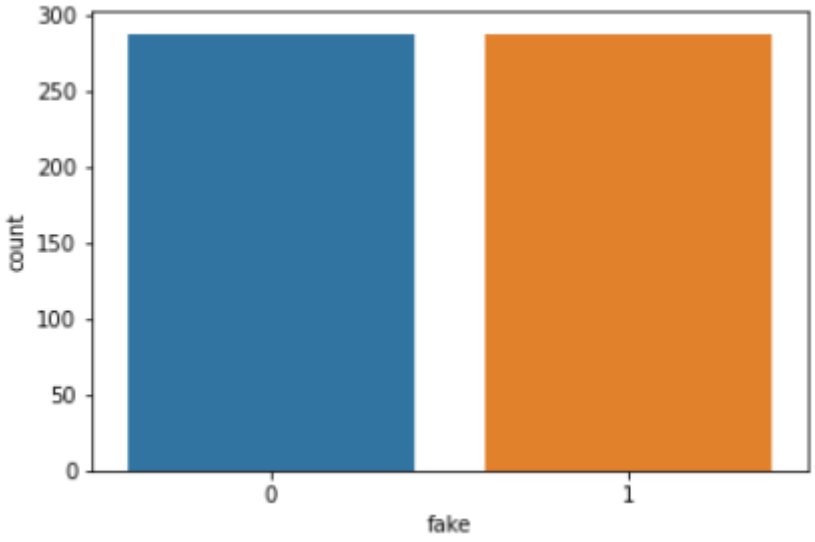


Figure 6.2: Bar Plot – Visualization (Data Insights)

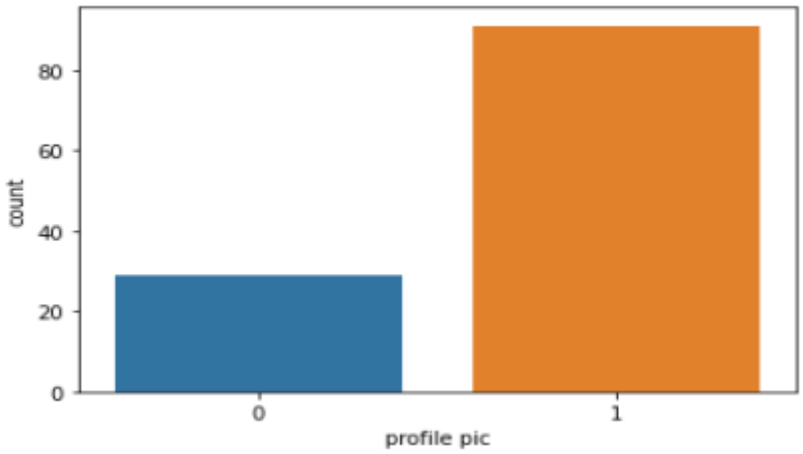


Figure 6.3: Bar Plot – Visualization (Data Insights)

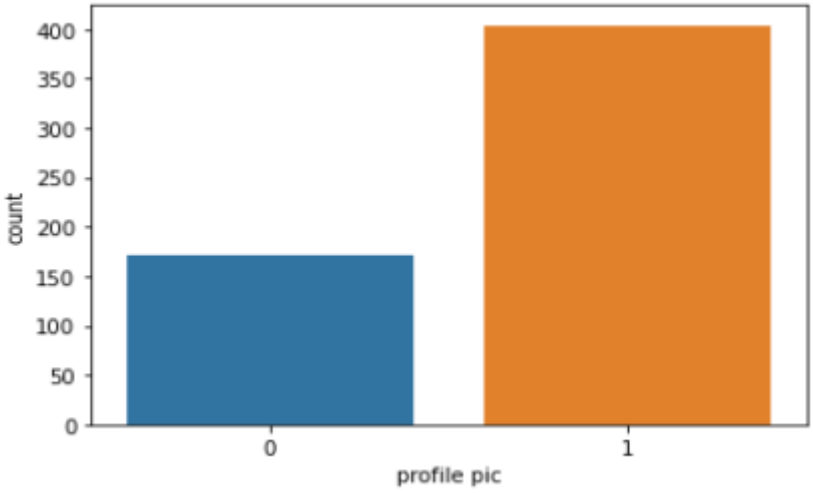


Figure 6.4: Bar Plot – Visualization (Data Insights)

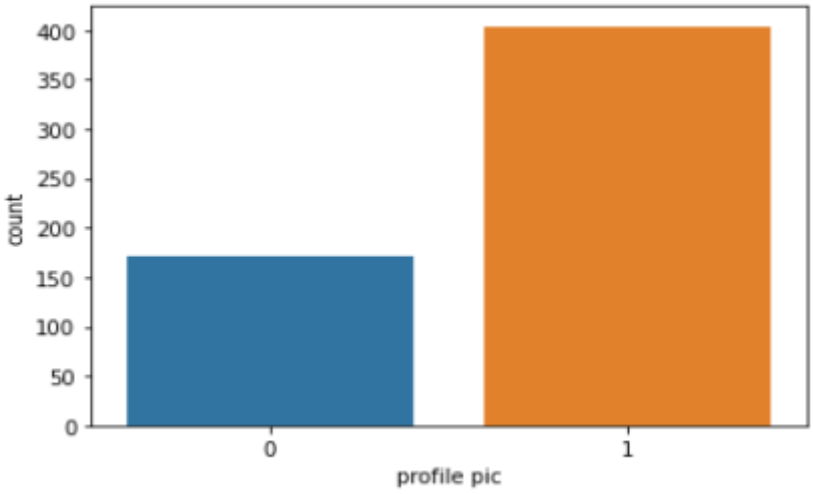
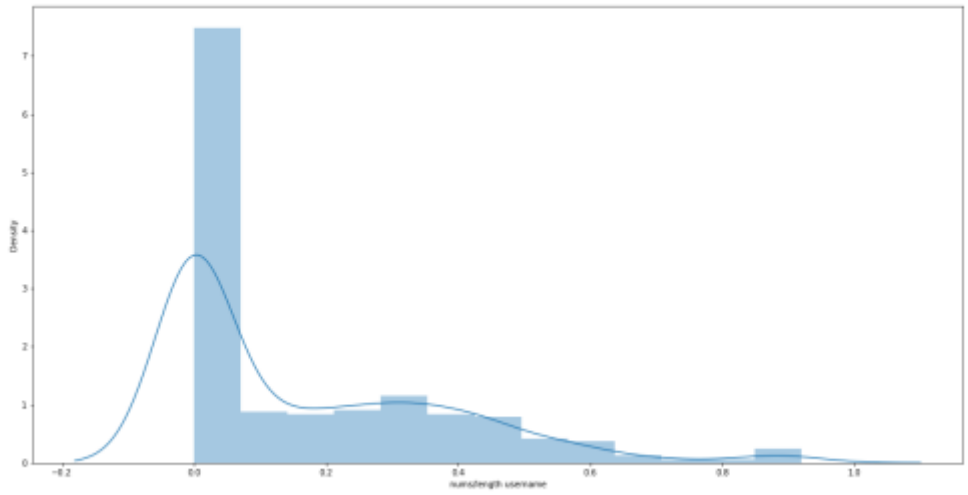


Figure 6.5: Bar Plot – Visualization (Data Insights)

KDE Plot (Data Insights)



Heat Map – Correlation Check (Data Insights)

Figure 6.6: KDE plot(Data Insights)

Heat Map – Correlation Check (Data Insights)

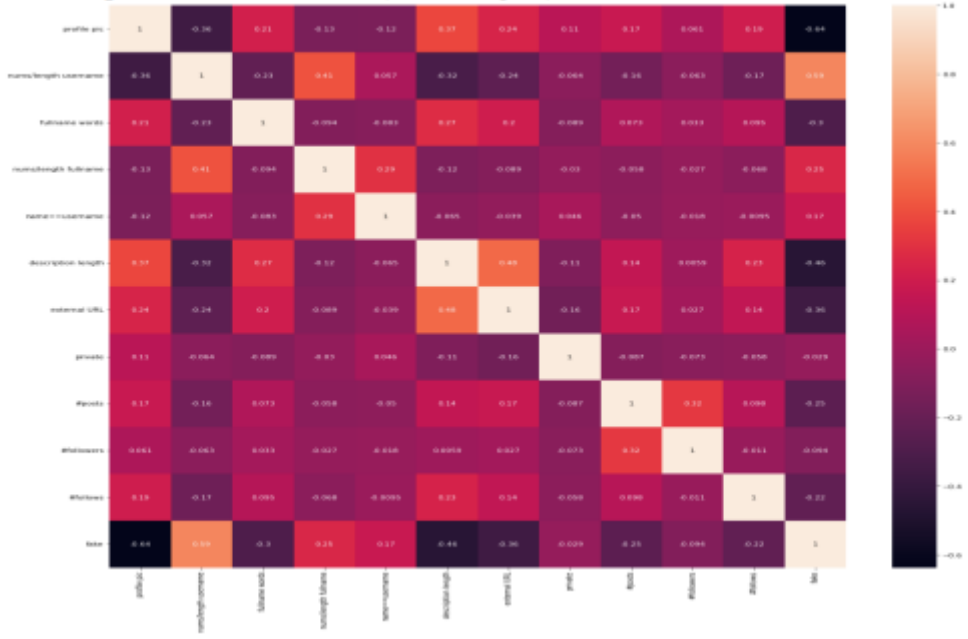


Figure 6.7: Heat map- Correlation check(Data Insights)

Model Training- (Sequential Training)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                600

 dense_1 (Dense)             (None, 150)               7650

 dropout (Dropout)           (None, 150)               0

 dense_2 (Dense)             (None, 150)               22650

 dropout_1 (Dropout)         (None, 150)               0

 dense_3 (Dense)             (None, 25)                3775

 dropout_2 (Dropout)         (None, 25)                0

 dense_4 (Dense)             (None, 2)                 52

=================================================================
Total params: 34,727
Trainable params: 34,727
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
17/17 [==============================] - 1s 24ms/step - loss: 0.6356 - accuracy: 0.6583 - val_loss: 0.4993 - val_accuracy: 0.8103
Epoch 2/50
17/17 [==============================] - 0s 7ms/step - loss: 0.4191 - accuracy: 0.8707 - val_loss: 0.3006 - val_accuracy: 0.8276
Epoch 3/50
17/17 [==============================] - 0s 6ms/step - loss: 0.3170 - accuracy: 0.8919 - val_loss: 0.3410 - val_accuracy: 0.8276
Epoch 4/50
17/17 [==============================] - 0s 7ms/step - loss: 0.3015 - accuracy: 0.8958 - val_loss: 0.1594 - val_accuracy: 0.9138
Epoch 5/50
17/17 [==============================] - 0s 6ms/step - loss: 0.2653 - accuracy: 0.9054 - val_loss: 0.1720 - val_accuracy: 0.8966
Epoch 6/50
17/17 [==============================] - 0s 6ms/step - loss: 0.2506 - accuracy: 0.9131 - val_loss: 0.1611 - val_accuracy: 0.9138
Epoch 7/50
17/17 [==============================] - 0s 6ms/step - loss: 0.2604 - accuracy: 0.9093 - val_loss: 0.1841 - val_accuracy: 0.8966
Epoch 8/50
17/17 [==============================] - 0s 7ms/step - loss: 0.2420 - accuracy: 0.9151 - val_loss: 0.2184 - val_accuracy: 0.8966
Epoch 9/50
17/17 [==============================] - 0s 7ms/step - loss: 0.2153 - accuracy: 0.9266 - val_loss: 0.1787 - val_accuracy: 0.8966
Epoch 10/50
17/17 [==============================] - 0s 7ms/step - loss: 0.2151 - accuracy: 0.9286 - val_loss: 0.2093 - val_accuracy: 0.8966
Epoch 11/50
17/17 [==============================] - 0s 6ms/step - loss: 0.2052 - accuracy: 0.9189 - val_loss: 0.1791 - val_accuracy: 0.8966
Epoch 12/50
17/17 [==============================] - 0s 6ms/step - loss: 0.2081 - accuracy: 0.9266 - val_loss: 0.1888 - val_accuracy: 0.9138
Epoch 13/50
17/17 [==============================] - 0s 6ms/step - loss: 0.1933 - accuracy: 0.9189 - val_loss: 0.2070 - val_accuracy: 0.9138
Epoch 14/50
17/17 [==============================] - 0s 6ms/step - loss: 0.1995 - accuracy: 0.9286 - val_loss: 0.2029 - val_accuracy: 0.9138
Epoch 15/50
17/17 [==============================] - 0s 6ms/step - loss: 0.1913 - accuracy: 0.9170 - val_loss: 0.1981 - val_accuracy: 0.9138
```

Figure 6.8: Model training-(sequential training)

Training Progress - Loss (Training)

Figure 6.9: Training Progress-Loss training

Training Progress - Accuracy(Training)

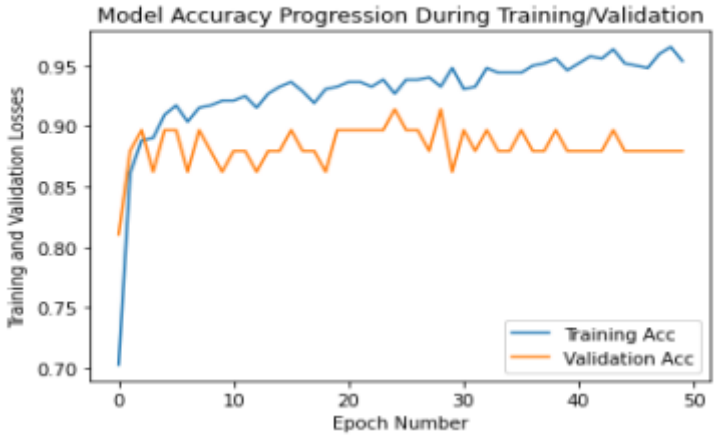


Figure 6.10: Training Progress- Accuracy Training

Classification Report (Evaluation)

```
print("Accuracy : ", get_avg(model_training_progress['Accuracy']) * 100)

print("Validation Accuracy : ", get_avg(model_training_progress['Validation_Accuracy']) * 100)


print("Loss : ", get_avg(model_training_progress['Loss']) * 100)


print("Validation Loss : ", get_avg(model_training_progress['Validation Loss']) * 100)
```

```
Accuracy :  92.91891884803772
Validation Accuracy :  88.31034588813782
Loss :  17.7891463637352
Validation Loss :  26.413013011217117
```

Figure 6.11: Classification Report Evaluation

```
predicted = model.predict(X_test)

predicted_value = []
test = []
for i in predicted:
    predicted_value.append(np.argmax(i))

for i in y_test:
    test.append(np.argmax(i))

print(classification_report(test, predicted_value))
```

```
              precision    recall  f1-score   support

           0       0.91      0.87      0.89        60
           1       0.87      0.92      0.89        60

    accuracy                           0.89       120
   macro avg       0.89      0.89      0.89       120
weighted avg       0.89      0.89      0.89       120
```

Figure 6.12: Classification Report Evaluatio

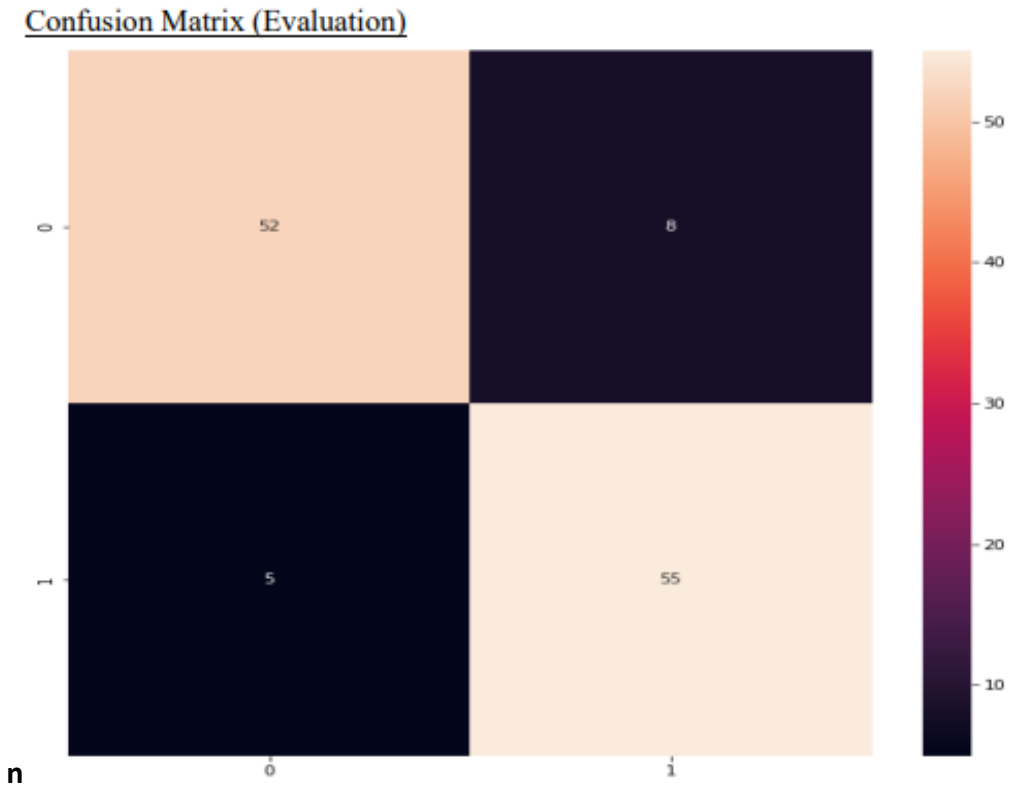


Figure 6.13: Classification Report Evaluation

CHAPTER- 8

SYSTEMTESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement

8.1 TYPES OF TESTS

Unit testing:

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results

Integration testing:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional testing:

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions :identified functions must be exercised.

Output : identified classes of application outputs must exercised

Procedures : interfacing systems or procedures must be invoked

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Testing:

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points

White Box Testing:

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing:

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

8.2 TEST STRATEGY AND APPROACH

Field testing will be performed manually and functional tests will be written in detail.

Test objectives:

● All field entries must work properly.

● Pages must be activated from the identified link.

● The entry screen, messages and responses must not be delayed.

Features to be tested:

● Verify that the entries are of the correct format

● No duplicate entries should be allowed

● All links should take the user to the correct page

Integration Testing:

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at thecompany level – interact without error .

Test Results: All the test cases mentioned above passed successfully. No defects encountered. Acceptance

Testing: User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

8.3 VALIDATION:

The process of ensuring that the software or system meets the requirements and expectations of the stakeholders and users. It involves assessing whether the product being developed is what the stakeholders actually need and want. Validation typically occurs after the completion of development but before the product is released to the end-users.

Requirement Validation:

This involves verifying that the software meets the specified requirements. It ensures that the implemented features and functionalities align with the needs of the stakeholders as outlined in the requirements

documentation. Requirement validation may involve reviews, walkthroughs, and inspections of the requirements documents to confirm their completeness, correctness, and relevance.

User Acceptance Testing (UAT):

UAT is a type of validation testing where endusers or representatives of the end-users test the software in a real-world environment to determine whether it satisfies their needs and expectations. UAT typically involves creating test scenarios based on real-life use cases and having users execute these scenarios to validate the software's usability, functionality, and overall suitability for their needs

Functional Validation:

This involves validating that the software functions correctly according to the specified requirements. It verifies that each feature and functionality performs as intended and meets the desired outcomes. Functional validation may involve various testing techniques such as functional testing, integration testing, and system testing to ensure that the software behaves as expected under different conditions.

Performance Validation :

Performance validation focuses on assessing the performance characteristics of the software, such as response time, throughput, scalability, and reliability. It ensures that the software can handle the expected workload and perform efficiently under normal and peak usage conditions. Performance validation may involve performance testing techniques like load testing,stress testing, and scalability testing to identify and address any performancerelated issues.

Security Validation:

Security validation involves evaluating the security features and measures implemented in the software to protect it from unauthorized access, data breaches, and other security threats. It ensures that the software complies with security standards, regulations, and best practices. Security validation may involve security testing techniques such as penetration testing, vulnerability scanning, and security code reviews to identify and mitigate security vulnerabilities and weakness.

8.4 TYPES OF TEST CASES

When designing test cases for fake profile detection, it's important to cover various aspects of the detection process to ensure its effectiveness. Here are some test cases you might consider:

- Basic Profile Information Validation:

  - Test a profile with incomplete or obviously fake basic information (e.g., empty name, nonsensical date of birth).

  - Test a profile with suspiciously similar basic information across multiple profiles.

- Profile Activity Patterns:
  - Test profiles with abnormal activity patterns (e.g., excessive friend requests, rapid posting frequency).
  - Test profiles with no activity or very minimal activity over an extended period.
- Profile Picture Analysis
  - Test profiles with images that have been flagged as commonly used in fake profiles (using image recognition).

  - Test profiles with inconsistent or low-quality profile pictures.

- Text Analysis:
  - Test profiles with generic or nonsensical "About Me" sections or descriptions.
  - Test profiles with text that includes suspicious keywords or phrases often associated with fake profiles (e.g., "get rich quick," "free giveaways," etc.).
- Friend Network Analysis:
  - Test profiles with an unusually large number of friends added in a short period.
  - Test profiles with a friend network that shows patterns of reciprocal friendships with other fake profiles.
- Behavioral Analysis:
  - Test profiles with abnormal login patterns (e.g., logging in from multiple locations within a short span of time).
  - Test profiles that exhibit bot-like behavior, such as consistently responding at odd hours or with scripted responses.
- Cross-Platform Verification:
  - Test profiles that are linked to other social media accounts with inconsistent or suspicious information.
  - Test profiles that are linked to email addresses or phone numbers associated with known fake profiles.

- Manual Review Test Cases:

  - Test cases where human reviewers manually examine profiles flagged as suspicious by the automated system to check for false positives/negatives.

- Historical Data Analysis:

  - Test cases involving historical data analysis to identify patterns or correlations between profiles flagged as fake and certain behaviors or attributes.

- Geolocation Analysis:

  - Test profiles with inconsistent geolocation data (e.g., a profile claiming to be in one country but showing activity consistent with another).

- Consistency Checks:

  - Test profiles for consistency across different attributes (e.g., age, location, interests) to identify discrepancies that may indicate fakeness

- Machine Learning Model Evaluation:

  - Test the accuracy of the machine learning models used for fake profile detection against a labeled dataset of known fake and genuine profiles.

- Scalability Tests:

  - Test the system's performance when processing a large number of profiles simultaneously to ensure it can handle peak loads without compromising accuracy.

- False Positive/Negative Testing:

  - Test cases where known fake profiles are deliberately created to check if they are correctly flagged, and vice versa, to ensure the system's precision and recall rates.

- Edge Cases:

  - Test cases involving unusual scenarios or edge cases that may not be covered by standard detection methods (e.g., profiles with mixed characteristics of fake and genuine).

- Regulatory Compliance:

  - Test cases to ensure compliance with relevant data protection regulations, especially regarding the handling and processing of user data for profile detection purposes.

By incorporating these test cases into your fake profile detection system, you can assess its robustness, accuracy, and reliability in identifying and mitigating fake profiles effectively.

8.3 TEST CASES

Test Case	Description
Basic Profile Information Validation	Test profiles with incomplete or obviously fake basic information. Test profiles with suspiciously similar basic information across multiple profiles.
Profile Activity Patterns	Test profiles with abnormal activity patterns such as excessive friend requests or rapid posting frequency. Test profiles with no activity or minimal activity over an extended period.
Profile Picture Analysis	Test profiles with images flagged as commonly used in fake profiles. Test profiles with inconsistent or low-quality profile pictures.
Text Analysis	Test profiles with generic or nonsensical "About Me" sections or descriptions. Test profiles with text containing suspicious keywords or phrases associated with fake profiles.
Friend Network Analysis	Test profiles with an unusually large number of friends added in a short period. Test profiles with a friend network displaying patterns of reciprocal friendships with other fake profiles.
Behavioral Analysis	Test profiles with abnormal login patterns or bot-like behavior such as responding at odd hours or with scripted responses.
Cross-Platform Verification	Test profiles linked to other social media accounts with inconsistent or suspicious information. Test profiles linked to email addresses or phone numbers associated with known fake profiles.
Manual Review Test Cases	Test cases where human reviewers manually examine profiles flagged as suspicious by the automated system to check for false positives/negatives.
Historical Data Analysis	Test cases involving historical data analysis to identify patterns or correlations between profiles flagged as fake and certain behaviors or attributes.
Geolocation Analysis	Test profiles with inconsistent geolocation data.
Consistency Checks	Test profiles for consistency across different attributes to identify discrepancies that may indicate fakeness.
Machine Learning Model Evaluation	Test the accuracy of machine learning models against a labeled dataset of known fake and genuine profiles.
Scalability Tests	Test the system's performance when processing a large number of profiles simultaneously.

Test Case	Description
False Positive/Negative Testing	Test known fake profiles to check if they are correctly flagged, and vice versa.
Edge Cases	Test cases involving unusual scenarios or edge cases that may not be covered by standard detection methods.
Regulatory Compliance	Test cases to ensure compliance with relevant data protection regulations.

Figure 8.1:Test cases table

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENTS

9.1 Conclusion

In conclusion, designing comprehensive test cases for fake profile detection is essential to ensure the effectiveness and reliability of the detection system. By covering various aspects such as basic profile information validation, activity patterns, image and text analysis, friend network examination, behavioral analysis, cross-platform verification, and more, we can thoroughly evaluate the system's capability to identify and mitigate fake profiles.Furthermore, incorporating manual review test cases, historical data analysis, machine learning model evaluation, scalability tests, and compliance checks enhances the robustness and accuracy of the detection process. It's crucial to address false positives and false negatives, as well as consider edge cases and regulatory compliance to ensure the system's integrity and trustworthiness.By systematically testing the fake profile detection system across these dimensions, organizations can gain confidence in its ability to protect users from fraudulent activities and maintain a safe and authentic online environment. Continuous refinement and validation of these test cases are essential to adapt to evolving threats and maintain the efficacy of the detection mechanisms over time.Effective fake profile detection is vital for maintaining the integrity and security of online platforms, social networks, and digital communities. In addition to the test cases outlined earlier, ongoing monitoring and adaptation are crucial components of a robust detection system.Continuous refinement of detection algorithms and methodologies is necessary to stay ahead of evolving tactics used by malicious actors creating fake profiles. This may involve incorporating new data sources, refining machine learning models, or adopting advanced techniques such as natural language processing and deep learning.

9.2 FUTURE ENHANCEMENTS

Looking ahead, there are several potential future enhancements that could further strengthen fake profile detection systems:

- Advanced Machine Learning Techniques: Continuously improving machine learning models by leveraging advancements in techniques such as deep learning, reinforcement learning, and ensemble methods can enhance the accuracy and efficiency of fake profile detection.
- Natural Language Understanding (NLU): Integrating NLU capabilities can enable more nuanced analysis of profile descriptions, comments, and messages, helping to identify subtle cues indicative of fake profiles.
- Behavioral Biometrics: Incorporating behavioral biometrics, such as typing patterns, mouse movements, and browsing behavior, can add an extra layer of authentication to verify the identity of users and detect anomalies associated with fake profiles.
- Blockchain Technology: Leveraging blockchain technology for identity verification and credential management can enhance trust and transparency in user identities, making it more difficult for malicious actors to create fake profiles.
- Collaborative Filtering and Network Analysis: Utilizing collaborative filtering algorithms and network analysis techniques can help identify clusters of fake profiles and their connections, enabling more effective detection of coordinated fake profile campaigns.
- Real-Time Monitoring: Implementing real-time monitoring capabilities can enable immediate detection and response to emerging threats and suspicious activities, reducing the potential impact of fake profiles on users and the platform.
- User Engagement Analysis: Analyzing user engagement metrics, such as likes, shares, and comments, in conjunction with profile attributes can provide valuable insights into the authenticity of profiles and help identify anomalies indicative of fake profiles.
- Cross-Platform Integration: Integrating fake profile detection systems with other online platforms and social networks can enable the sharing of data and insights, facilitating a more comprehensive approach to combating fake profiles across the digital landscape.
- Privacy-Preserving Techniques: Developing privacy-preserving techniques, such as federated learning and differential privacy, can enable collaborative model training while protecting sensitive user data, ensuring compliance with privacy regulations.

- Continuous Learning and Adaptation: Implementing mechanisms for continuous learning and adaptation based on feedback from user reports, manual reviews, and emerging threats can help fake profile detection systems stay effective in dynamic and evolving environments.

By incorporating these future enhancements into fake profile detection systems, organizations can stay ahead of emerging threats and provide users with a safer and more authentic online experience.

CHAPTER-10

REFERENCES

[1] Yuan, D., Miao, Y., Gong, N. Z., Yang, Z., Li, Q., Song, D., Wang, D. and Liang, X. (2019) Detecting Fake Accounts in Online Social Networks at the Time of Registrations. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, 11-15 November1423-1438.

https://doi.org/10.1145/3319535.3363198

[2] Kudugunta, S. and Ferrara, E. (2018) Deep Neural Networks for Bot Detection. Information Sciences, 467, 312-322.

https://doi.org/10.1016/j.ins.2018.08.019

[3] Ramalingam, D. and Chinnaiah, V. (2018) Fake Profile Detection Techniques in Large-Scale Online Social Networks: A Comprehensive Review. Computers & Electrical Engineering, 65, 165-177.

https://doi.org/10.1016/j.compeleceng.2017.05.020

[4] Hajdu, G., Minoso, Y., Lopez, R., Acosta, M. and Elleithy, A. (2019) Use of Artificial Neural Networks to Identify Fake Profiles. 2019 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, 3 May 2019, 1-4.

https://doi.org/10.1109/LISAT.2019.8817330

[5] Swe, M.M. and Myo, N.N. (2018) Fake Accounts Detection on Twitter Using Blacklist. 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS), Singapore, 6-8 June 2018, 562-566.

https://doi.org/10.1109/ICIS.2018.8466499

[6] Wanda, P. and Jie, H.J. (2020) DeepProfile: Finding Fake Profile in Online Social Network Using Dynamic CNN. Journal of Information Security and Applications, 52, Article ID: 102465.

https://doi.org/10.1016/j.jisa.2020.102465

[7] Kodati, S., Reddy, K.P., Mekala, S., Murthy, P.S. and Reddy, P.C.S. (2021) Detection of Fake Profiles on Twitter Using Hybrid SVM Algorithm. E3S Web of Conferences, 309, Article No. 01046.

https://doi.org/10.1051/e3sconf/202130901046

[8] Meshram, E.P., Bhambulkar, R., Pokale, P., Kharbikar, K. and Awachat, A. (2021) Automatic Detection of Fake Profile Using Machine Learning on Instagram. International Journal of Scientific Research in Science and Technology, 8, 117-127.

https://doi.org/10.32628/IJSRST218330

[9] Chakraborty, P., Muzammel, C.S., Khatun, M., Islam, S.F. and Rahman, S. (2020) Automatic Student Attendance System Using Face Recognition. International Journal of Engineering and Advanced Technology (IJEAT), 9, 93-99.

https://doi.org/10.35940/ijeat.B4207.029320

[10] Sayeed, S., Sultana, F., Chakraborty, P. and Yousuf, M.A. (2021) Assessment of Eyeball Movement and Head Movement Detection Based on Reading. In: Bhattacharyya, S., Mršić, L., Brkljačić, M., Kureethara, J.V. and Koeppen, M., Eds., Recent Trends in Signal and Image Processing, Springer, Singapore, 95-103.

https://doi.org/10.1007/978-981-33-6966-5_10

[11] Chakraborty, P., Yousuf, M.A. and Rahman, S. (2021) Predicting Level of Visual Focus of Human's Attention Using Machine Learning Approaches. In: Shamim Kaiser, M., Bandyopadhyay, A., Mahmud, M. and Raym K., Eds., Proceedings of International Conference on Trends in Computational and Cognitive Engineering, Springer, Singapore, 683-694.

https://doi.org/10.1007/978-981-33-4673-4_56

[12] Muzammel, C.S., Chakraborty, P., Akram, M.N., Ahammad, K. and Mohibullah, M. (2020) Zero-Shot Learning to Detect Object Instances from Unknown Image Sources. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 9, 988-991.

https://doi.org/10.35940/ijitee.C8893.029420

[13] Sultana, M., Ahmed, T., Chakraborty, P., Khatun, M., Hasan, M.R. and Uddin, M.S. (2020) Object Detection Using Template and Hog Feature Matching. International Journal of Advanced Computer Science and Applications, 11, 233-238.

https://doi.org/10.14569/IJACSA.2020.0110730

[14] Faruque, M.A., Rahman, S., Chakraborty, P., Choudhury, T., Um, J.S. and Singh, T.P. (2021) Ascertaining Polarity of Public Opinions on Bangladesh Cricket Using Machine Learning Techniques. Spatial Information Research, 30, 1-8.

https://doi.org/10.1007/s41324-021-00403-8

[15] Sarker, A., Chakraborty, P., Sha, S.S., Khatun, M., Hasan, M.R. and Banerjee, K. (2020) Improvised Technique for Analyzing Data and Detecting Terrorist Attack Using Machine Learning Approach Based on Twitter Data. Journal of Computer and Communications, 8, 50-62.

https://doi.org/10.4236/jcc.2020.87005

[16] Ahammad, K., Shawon, J.A.B., Chakraborty, P., Islam, M.J. and Islam, S. (2021) Recognizing Bengali Sign Language Gestures for Digits in Real Time using Convolutional Neural Network. International Journal of Computer Science and Information Security (IJCSIS), 19, 11-19.

[17] Sultana, M., Chakraborty, P. and Choudhury, T. (2022) Bengali Abstractive News Summarization Using Seq2Seq Learning with Attention. In: Tavares, J.M.R.S., Dutta, P., Dutta, S. and Samanta, D., Eds., Cyber Intelligence and Information Retrieval, Springer, Singapore, 279-289.

https://doi.org/10.1007/978-981-16-4284-5_24