A Project report on

Detection of Fake and Clone Accounts in Twitter using Classification and Distance Measure Algorithms

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

G. KIRAN DEEPAK
(19H51A0570)

K.V.S.S.S.R.H SRI HARSHA
(19H51A0573)

G. PRANAY KUMAR
(19H51A05F8)

Under the esteemed guidance of

Dr. P. SENTHIL
(Associate Professor)

Department of Computer Science and Engineering

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC Autonomous)

*Approved by AICTE *Affiliated to JNTUH *NAAC Accredited with A^+ Grade

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

2019- 2023

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Major Project report entitled **"Detection of fake and clone accounts using classification and distance measure algorithms"** being submitted by G.Kiran Deepak (19H51A0570), K.V.S.S.S.R.H Sri Harsha (19H51A0573), G.Pranay Kumar (19H51A05F8) in partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out his/her under my guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

Dr. P. Senthil
Associate Professor
Dept. of CSE

Dr. Siva Skandha Sanagala
Associate Professor and HOD
Dept. of CSE

ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Dr. P. Senthil, Associate Professor** , Department of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. Siva Skandha Sanagala,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr. Vijaya Kumar Koppula**, Dean-Academic, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Computer Science and Engineering for their co-operation

We express our sincere thanks to **Mr. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care.

Finally, We extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

G.Kiran Deepak	19H51A0570
K.V.S.S.S.R.H Sri Harsha	19H51A0573
G.Pranay Kumar	19H51A05F8

TABLE OF CONTENTS

List of Figures

List of Tables

ABSTRACT

Online Social Network (OSN) is a network hub where people with similar interests or real world relationships interact. As the popularity of OSN is increasing, the security and privacy issues related to it are also rising. Fake and Clone profiles are creating dangerous security problems for social network users. Cloning of user profiles is one serious threat, where already existing user's details are stolen to create duplicate profiles and then it is misused for damaging the identity of the original profile owner. They can even launch threats like phishing, stalking, spamming etc. For Profile Cloning detection two methods are used. One using Similarity Measures and the other using C4.5 decision tree algorithm. In Similarity Measures, two types of similarities are considered – Similarity of Attributes and Similarity of Network relationships. C4.5 detects clones by building decision tree by taking information gain into consideration. A comparison is made to check how well these two methods help in detecting clone profiles.

CHAPTER 1
INTRODUCTION

CHAPTER 1
INTRODUCTION

Online Social Networks (OSN) like Face book, Twitter, LinkedIn, Instagram etc are used by billions of users all around the world to build network connections. The ease and accessibility of social networks have created a new era of networking. OSN users share a lot of information in the network like photos, videos, school name, college name, phone numbers, email address, home address, family relations, bank details, career details etc. This information if put into hands of attackers, the after effects are very severe.

Most of the OSN users are unaware of the security threats that exist in the social networks and easily fall prey to these attacks. The risks are more dangerous if the victims are children. In Profile Cloning attack, the profile information of existing users are stolen to create duplicate profiles and these profiles are misused for spoiling the identity of original profile owners. There are two types of Profile Cloning namely - Same Site and Cross Site Profile Cloning. If user credentials are taken from one Network to create a clone profile in same Network then it is called Same Site profile cloning.

In Cross Site profile cloning, attacker takes the user information from one Network to create a duplicate profile in other Network in which the user is not having any account. As the registration process in social networks have become very simple in order to attract more and more users, the creation of fake profiles are also increasing in an alarming rate. An attacker creates a fake profile in order to connect to a victim to cause malicious activities. And also to spread fake news and spam messages.

1.1 Problem Statement

Today, Fake and Clone profiles have become a very serious threat in social networks. So, a detection method is very much necessary to find these frauds who use people's faith to gather private information and create duplicate profiles. Many authors have worked in this area and have proposed methods to identify these type of profiles in social networks.

1.2 Research Objective:

The main objective of this project is to

- Understanding the Prevalence of Fake and Clone Accounts: To understand the prevalence of fake and clone accounts on Twitter. This involves analysing existing literature, examining previous studies, and collecting data on the percentage of fake and clone accounts in the Twitter ecosystem. This research objective provides a baseline for understanding the scope and magnitude of the problem, which helps in developing effective detection methods.
- Identifying Characteristics of Fake and Clone Accounts: To identify the characteristics of fake and clone accounts on Twitter. This involves analysing patterns of behaviour, user profile information, posting frequency, content type, engagement patterns, and other features that may differentiate fake and clone accounts from genuine accounts. This research objective aims to identify key indicators that can be used in developing detection algorithms.
- Evaluating Existing Detection Methods: To evaluate existing methods for detecting fake and clone accounts on Twitter. This involves reviewing the literature and examining previous studies that have proposed and evaluated detection methods. This research objective helps in identifying the strengths and limitations of existing methods and provides insights into areas that need improvement.
- Developing Novel Detection Methods: To develop novel methods for detecting fake and clone accounts on Twitter. This involves leveraging machine learning techniques, natural language processing, network analysis, and other computational approaches to develop algorithms that can effectively identify fake and clone accounts. This research objective aims to contribute to the advancement of the field by proposing new and innovative methods for detection.
- Evaluating the Performance of Detection Methods: To evaluate the performance of the developed detection methods. This involves collecting a dataset of labeled fake, clone, and genuine accounts, and evaluating the accuracy, precision, recall, F1 score, and other performance metrics of the proposed detection methods. This research objective aims to assess the effectiveness and efficiency of the developed methods and compare them with existing methods.

- Understanding the Impacts of Fake and Clone Accounts: To understand the impacts of fake and clone accounts on Twitter. This involves analyzing the spread of misinformation, influence on public opinion, manipulation of social media trends, and other potential negative effects of fake and clone accounts. This research objective provides insights into the societal, political, and economic implications of fake and clone accounts, which can inform policy-making and intervention strategies.
- Proposing Countermeasures against Fake and Clone Accounts: To propose countermeasures against fake and clone accounts on Twitter. This involves developing strategies for detection, prevention, and mitigation of fake and clone accounts, such as improving Twitter's algorithms, implementing user verification mechanisms, promoting digital literacy, and fostering user awareness. This research objective aims to provide practical recommendations for addressing the problem of fake and clone accounts on Twitter.

In summary, the research objective is to investigate and develop effective methods for detecting fake and clone accounts on Twitter, by understanding their prevalence, identifying their characteristics, evaluating existing methods, developing novel methods, evaluating their performance, understanding their impacts, and proposing countermeasures. This research can contribute to the advancement of the field and help in mitigating the negative effects of fake and clone accounts on Twitter and other social media platforms.

1.3 Project Scope and Limitations:

Project Scope

The scope of the project includes investigating and developing methods for detecting fake and clone accounts on Twitter, with a focus on understanding and identifying their characteristics, evaluating existing methods, developing better solutions, evaluating their performance, understanding their impacts, and proposing countermeasures. The project may involve collecting and analyzing data from Twitter, conducting experiments and evaluations of detection methods, and proposing practical recommendations for addressing the problem of fake and clone accounts on Twitter.

- Our Problem falls under classification category.
- Classification is to determine the class to which each data sample of the methods belongs, which methods are used when the outputs of input data are qualitative.
- For the present the project is limited to twitter accounts data set which classifies the data of twitter segregates and does the analysis on twitter data set.

Limitations

There are several limitations to consider in the scope of the project, including:

- Limited access to data: The project may be limited by the availability and access to data from Twitter. Access to Twitter's data may be restricted or limited, and the project may need to rely on publicly available datasets or synthetic data, which may not fully capture the complexity and dynamics of real-world fake and clone accounts.
- Evolving nature of fake and clone accounts: Fake and clone accounts are constantly evolving, with new techniques and strategies being employed to evade detection. The project may face limitations in keeping up with the rapidly changing landscape of fake and clone accounts, and the developed methods may become outdated over time.

- Ethical considerations: The project may face ethical considerations in terms of data privacy, user consent, and potential biases in detection methods. Ensuring the ethical use of data and addressing potential biases in detection methods should be carefully considered and mitigated.
- Generalizability: The project may be limited in terms of generalizability across different regions, languages, and topics on Twitter. Detection methods that are effective in one context may not be directly applicable to other contexts, and the project may need to consider the limitations and challenges of generalizing the findings to different settings.
- Evaluation challenges: Evaluating the performance of detection methods may be challenging, as there may not be a ground truth for identifying fake and clone accounts. Developing appropriate evaluation metrics and benchmark datasets for comparison may be limited by the availability and reliability of labeled data.
- Implementation challenges: Implementing proposed countermeasures against fake and clone accounts may face challenges in terms of technical feasibility, adoption by Twitter or other stakeholders, and potential unintended consequences. The project may need to consider the practicality and feasibility of implementing proposed countermeasures in a real-world setting.
- Dynamic nature of Twitter: Twitter is a dynamic social media platform with constantly changing user behaviors, trends, and features. The project may face limitations in capturing the dynamic nature of Twitter, and the findings and recommendations may need to be updated and adapted to reflect the evolving landscape of Twitter.

It's important to acknowledge these limitations and carefully consider them in the project design, methodology, and interpretation of findings. Transparently communicating the limitations of the project in the research findings and recommendations is crucial for ensuring the validity and reliability of the research outcomes.

CHAPTER 2
BACKGROUND WORK

CHAPTER 2
BACKGROUND WORK

2.1 Fake Account Detection using Naïve Bayes Algorithm

2.1.1 Introduction

Naive Bayes is a simple yet powerful probabilistic algorithm used for classification tasks in machine learning. It is based on the Bayes theorem, which is a statistical formula used to estimate the probability of an event occurring given certain evidence or observations. Naive Bayes is particularly effective for text classification tasks, such as spam detection, sentiment analysis, and fake news detection, but it can also be applied to other types of data.

The basic idea behind Naive Bayes is to calculate the conditional probability of a class label given the input features, and then use this probability to make predictions. The "naive" assumption in Naive Bayes is that all features are independent of each other, meaning that the presence of one feature does not affect the presence of another. This simplifying assumption allows for efficient and fast training of the model, as the model can be trained independently for each feature.

Naive Bayes uses a probabilistic approach to calculate the likelihood of observing the input features given each class label. It assumes that the probability distribution of the features for each class label follows a specific distribution, such as Gaussian (for continuous features), Bernoulli (for binary features), or Multinomial (for discrete features). The algorithm estimates the parameters of these probability distributions from the training data, and then uses them to compute the likelihood of observing the input features for each class label.

Once the likelihoods are calculated, Naive Bayes combines them with the prior probabilities of each class label to obtain the posterior probability of each class given the input features. The class label with the highest posterior probability is predicted as the final output. Naive Bayes is known for its simplicity, speed, and ability to handle high-dimensional data, making it a popular choice for many classification tasks.

Despite its simplicity, Naive Bayes does have some limitations. The "naive" assumption of feature independence may not always hold true in real-world data, and this can result in less accurate predictions. Additionally, Naive Bayes may not perform well when the training data is imbalanced, or when there are missing or noisy data. Nevertheless, Naive Bayes remains a widely used algorithm due to its ease of implementation, interpretability, and good performance in many practical scenarios.

2.1.2 Merits, Demerits and Challenges

Merits of Naive Bayes Algorithm:

- Simplicity and Speed: Naive Bayes is a simple algorithm that is easy to understand and implement. It has low computational overhead, making it fast for training and prediction, making it suitable for large datasets.
- Scalability: Naive Bayes is scalable to high-dimensional data, making it suitable for text classification tasks where the number of features (words) can be very large.
- Interpretable: Naive Bayes provides interpretable results, as it calculates the conditional probabilities of class labels given the input features, allowing for easy interpretation of the model's predictions.
- Robust to Noise: Naive Bayes is robust to noisy and irrelevant features, as it assumes feature independence. This means that even if some features are not informative for the classification task, they do not significantly impact the model's performance.
- Works well with small datasets: Naive Bayes can work well with small amounts of training data, making it suitable for scenarios where obtaining large labeled datasets may be challenging.

Demerits of Naive Bayes Algorithm:

- Assumption of Feature Independence: The "naive" assumption of feature independence may not always hold true in real-world data, and this can lead to less accurate predictions. In cases where features are highly correlated, Naive Bayes may not perform well.

- Simplified Probability Distribution Assumptions: Naive Bayes makes assumptions about the probability distribution of the features, such as Gaussian, Bernoulli, or Multinomial, which may not always accurately represent the true underlying distribution of the data.
- Sensitivity to Imbalanced Data: Naive Bayes can be sensitive to imbalanced datasets, where some classes have significantly fewer examples than others. This can result in biased predictions towards the majority class.

Challenges of Naive Bayes Algorithm:

- Handling Continuous and Categorical Data: Naive Bayes assumes that all features are either continuous or categorical, and the appropriate probability distribution needs to be chosen accordingly. Choosing the correct distribution can be a challenge, especially when dealing with mixed data types.
- Overcoming Feature Independence Assumption: The assumption of feature independence may not always hold true in real-world data, and finding ways to account for feature dependencies can be a challenge.
- Dealing with Missing Data: Naive Bayes may not handle missing data well, as it relies on complete data for probability estimation. Dealing with missing data can be challenging, and may require imputation or other techniques to ensure accurate model performance.
- Handling Class Imbalance: Naive Bayes can be sensitive to imbalanced datasets, where some classes have significantly fewer examples than others. Techniques such as oversampling, undersampling, or using different priors may be needed to address class imbalance.
- Model Selection and Hyperparameter Tuning: Like other machine learning algorithms, Naive Bayes requires careful model selection and hyperparameter tuning to achieve optimal performance. Choosing the appropriate probability distribution, handling missing data, and selecting appropriate priors can be challenging tasks that require careful consideration.

2.1.3. Implementation of Naïve bayes model

- Assume that all features in data set are independent
- Choose the most appropriate dependent variable and split the data set into training, validation, and testing.
- Build a frequency count of the training set and apply Naïve bayes on it

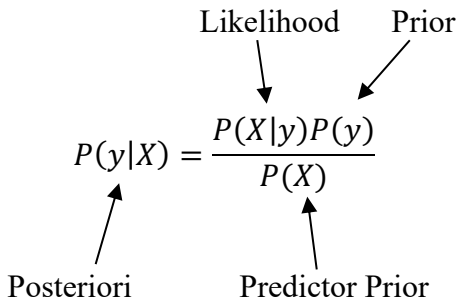


Figure 2.1.3.1: Naïve Bayes

The Bayes' theorem is used to determine the probability of a hypothesis when prior knowledge is available. It depends on conditional probabilities. The formula is given below:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where P(A|B) is posterior probability i.e. the probability of a hypothesis A given the event B occurs. P(B|A) is likelihood probability i.e. the probability of the evidence given that hypothesis A is true. P(A) is prior probability i.e. the probability of the hypothesis before observing the evidence and P(B) is marginal probability i.e. the probability of the evidence. When the Bayes' theorem is applied to classify accounts, the class c of a particular account d is given by :

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d)$$ MAP is "maximum a posteriori" = most likely class

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d \mid c)P(c)}{P(d)}$$ Bayes Rule

$$= \underset{c \in C}{\operatorname{argmax}} P(d \mid c)P(c)$$ Dropping the denominator

$$= \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, \ldots, x_n \mid c)P(c)$$ Document d represented as features x1..xn

General steps to implement Fake Account Detection using Naive Bayes algorithm:

- Load and preprocess the data: Start by loading the dataset that contains the input features (e.g., tweets, account information) and the corresponding class labels (e.g., real or fake accounts). Preprocess the data as needed, such as cleaning and transforming the text data into numerical features that can be used as input to the Naive Bayes algorithm. You may need to perform tasks like text normalization, tokenization, and feature extraction, depending on the nature of your data.
- Split the data: Split the pre-processed data into training and testing sets. The training set will be used to train the Naive Bayes model, while the testing set will be used to evaluate the performance of the trained model.
- Create a Naive Bayes model: Choose the appropriate type of Naive Bayes algorithm based on your data. For example, if you have text data, you can use Multinomial Naive Bayes for discrete features or Gaussian Naive Bayes for continuous features. Create an instance of the Naive Bayes class from the scikit-learn library, such as MultinomialNB or GaussianNB, depending on the type of data.
- Train the model: Use the training set to train the Naive Bayes model using the fit() method of the Naive Bayes class. This step involves estimating the parameters of the probability distribution model used by Naive Bayes, based on the training data.
- Make predictions: Use the trained Naive Bayes model to make predictions on the testing set using the predict() method of the Naive Bayes class. This step involves calculating the conditional probabilities of the input features given the class labels, and selecting the class with the highest probability as the predicted class for each instance in the testing set.
- Evaluate the model: Assess the performance of the Naive Bayes model by comparing the predicted class labels with the true class labels from the testing set. Calculate evaluation metrics such as accuracy, precision, recall, and F1-score to measure the effectiveness of the model in detecting fake accounts.
- Fine-tune the model: If necessary, fine-tune the Naive Bayes model by adjusting its parameters or hyperparameters, or by performing feature engineering to improve its performance. You can also experiment with different types of Naive Bayes algorithms or

explore ensemble methods to further enhance the accuracy and reliability of the fake account detection system.

- Deploy the model: Once you are satisfied with the performance of the Naive Bayes model, you can deploy it in a production environment to detect fake accounts in real-time. This may involve integrating the model into an application, website, or social media platform, and continuously monitoring its performance and updating it as needed.

Implementation of Fake Account Detection using Naive Bayes algorithm in Python using the scikit-learn library:

```
# Import the required libraries
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the dataset
data = pd.read_csv('fake_accounts_dataset.csv')

# Preprocess the data
X = data['text'] # Input features (text data)
y = data['label'] # Class labels (real or fake accounts)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a CountVectorizer object to convert text data into numerical features
vectorizer = CountVectorizer(stop_words='english', max_features=5000)
X_train_vectorized = vectorizer.fit_transform(X_train)
```

```
X_test_vectorized = vectorizer.transform(X_test)

# Create a Multinomial Naive Bayes model
naive_bayes = MultinomialNB()

# Train the model
naive_bayes.fit(X_train_vectorized, y_train)

# Make predictions
y_pred = naive_bayes.predict(X_test_vectorized)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

In this example, we use a CSV dataset with a "text" column containing the text data (e.g., tweets) and a "label" column containing the class labels (e.g., real or fake accounts). We preprocess the text data using a CountVectorizer to convert it into numerical features, and then train a Multinomial Naive Bayes model on the training set. We evaluate the model's performance using accuracy, precision, recall, and F1-score on the testing set. Note that this is a basic example and further customization, fine-tuning, and feature engineering may be needed depending on your specific dataset and requirements.

2.2 Fake Account Detection using SVM

2.2.1 Introduction

Support Vector Machine (SVM) is a popular supervised machine learning algorithm used for classification and regression tasks. It was originally introduced by Vapnik and his colleagues in the 1960s and has since gained widespread use in various fields, including computer vision, natural language processing, bioinformatics, and social media analytics.

At its core, SVM is a binary classification algorithm that aims to find an optimal hyperplane that can separate data points from different classes with the maximum margin. The margin is defined as the distance between the hyperplane and the nearest data points from each class, also known as support vectors. SVM seeks to maximize this margin, as it is believed that a wider margin results in a more robust and accurate classification model.

The key idea behind SVM is to transform the input data into a higher-dimensional space using a kernel function, which allows SVM to capture non-linear relationships between features. The most commonly used kernel functions are linear, polynomial, and radial basis function (RBF). SVM then finds the hyperplane that best separates the transformed data points, aiming to achieve the maximum margin between the classes.

One of the main strengths of SVM is its ability to handle high-dimensional data, making it suitable for tasks where the number of features is large compared to the number of samples. SVM is also robust to outliers, as it focuses on maximizing the margin rather than fitting the data points exactly. Additionally, SVM has a solid theoretical foundation and is known for producing global optimal solutions.

However, SVM also has some limitations. It requires careful selection and tuning of hyperparameters, such as the choice of kernel, regularization parameter (C), and kernel parameters, which can impact the performance of the model. SVM can also be computationally expensive, especially for large datasets, as it involves solving a convex optimization problem.

Interpretability of SVM models can be challenging, as the decision boundaries may be complex and difficult to explain to stakeholders. Additionally, SVM may struggle with imbalanced datasets and may require additional techniques, such as cost-sensitive learning or resampling, to handle class imbalance effectively.

Despite these limitations, SVM has been widely used and has shown promising results in many applications, including fake account detection on social media platforms. Its ability to handle high-dimensional data, robustness to outliers, and capability to capture non-linear relationships make it a popular choice for detecting fake accounts with complex patterns.

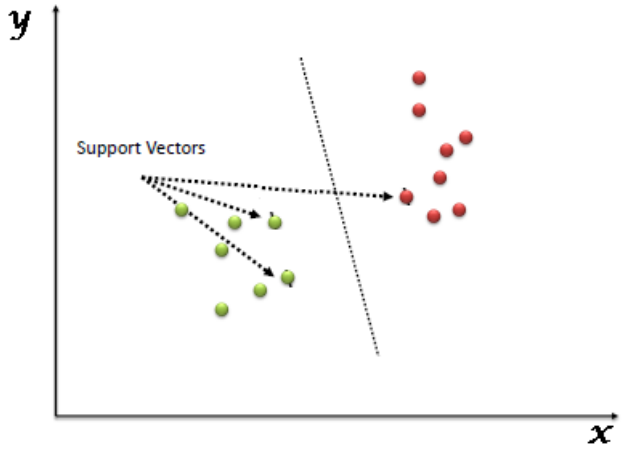


Figure 2.2.1.1: Graph of hyper plane

Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

2.2.2 Merits, Demerits, And Challenges

Merits of using SVM:

- Effective for High-Dimensional Data: SVM performs well in high-dimensional spaces, making it suitable for detecting fake accounts that may have complex patterns in their features.
- Robust to Outliers: SVM is less sensitive to outliers in the data compared to other classification algorithms, as it focuses on maximizing the margin between classes.
- Can Handle Non-Linear Data: SVM can use various kernel functions (such as linear, polynomial, and radial basis function) to transform the data into a higher-dimensional space, enabling it to capture non-linear relationships between features.
- Global Optimal Solution: SVM aims to find the global optimal solution, meaning it tends to produce a more robust and reliable model compared to other algorithms that may get stuck in local optima.

Demerits of using SVM:

- Sensitivity to Hyperparameters: SVM has hyperparameters, such as the choice of kernel, regularization parameter (C), and kernel parameters (e.g., gamma for RBF kernel), which need to be tuned for optimal performance. Poor selection of hyperparameters can lead to suboptimal results.
- Computational Complexity: SVM can be computationally expensive, especially for large datasets, as it involves solving a convex optimization problem that scales with the number of data points. Training SVM on very large datasets may require substantial computational resources.
- Lack of Interpretability: SVM models can be difficult to interpret and explain to stakeholders, as they often produce complex decision boundaries that are not easily interpretable in terms of human-understandable features or rules.
- Imbalanced Data: SVM may struggle with imbalanced datasets where the number of fake accounts is much smaller than the genuine accounts, as it may lead to biased model performance due to class imbalance.

Challenges of using SVM:

- Feature Engineering: The effectiveness of SVM heavily depends on the choice and quality of features used. Extracting relevant and informative features from social media data can be challenging, as fake accounts can employ various techniques to mimic genuine accounts.
- Data Availability: Obtaining labeled data for training SVM models for fake account detection can be challenging, as labeling fake accounts requires manual verification, which can be time-consuming and resource-intensive.
- Evolving Nature of Fake Accounts: Fake accounts constantly evolve and adapt to new detection techniques, making it challenging to develop a robust and accurate SVM model that can effectively detect various types of fake accounts.
- Privacy and Ethical Concerns: Analyzing social media data for fake account detection raises privacy and ethical concerns, such as collecting and using user data without consent, potential bias in the data, and the impact of false positives/negatives on innocent users.

2.2.3 Implementation of SVM Model

Here is a high-level overview of the steps involved in implementing a fake account detection system using SVM:

- Data Collection: Collect a dataset of Twitter accounts, including both authentic and fake accounts. The dataset should be representative and diverse, covering different regions, languages, and topics.
- Feature Extraction: Extract relevant features from the Twitter accounts that can be used as input for the SVM algorithm. Examples of features could include the number of tweets, the frequency of retweets and mentions, the account creation date, the account verification status, and the account's follower and friend counts.
- Data Preprocessing: Clean and preprocess the collected data by removing irrelevant or redundant features, handling missing values, and normalizing numerical features. This step is crucial for ensuring the quality and reliability of the input data.

- Labeling and Training Data Preparation: Label the dataset by categorizing the accounts as authentic or fake based on ground truth information. Split the labeled dataset into training and testing sets, with a majority of the data used for training and a smaller portion used for evaluation.
- Feature Selection: Select the most relevant features that are informative for fake account detection. This can be done using feature selection techniques, such as correlation analysis, mutual information, or regularization methods, to reduce the dimensionality of the data and improve the model's performance.
- SVM Model Training: Train an SVM model on the labeled training dataset using a suitable kernel function (e.g., linear, polynomial, or radial basis function). Tune the hyperparameters of the SVM model, such as the regularization parameter and kernel parameters, using techniques like cross-validation to optimize the model's performance.
- Model Evaluation: Evaluate the trained SVM model on the labeled testing dataset to assess its performance in terms of accuracy, precision, recall, F1-score, and other relevant evaluation metrics. This step helps to gauge the effectiveness and robustness of the model in detecting fake accounts.
- Model Deployment: Once the SVM model has been trained and evaluated, it can be deployed in a production environment to automatically detect fake accounts in real-time or on new data. This can be done using APIs or other integration methods depending on the specific application requirements.

Note: It's important to continuously update and improve the SVM model with new data and feedback to adapt to evolving fake account detection techniques and strategies employed by perpetrators.

Support Vector Machine (SVM) code in Python using the scikit-learn library:

```
# Import required libraries

import pandas as pd

from sklearn.model_selection import train_test_split
```

```
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load and preprocess the dataset

data = pd.read_csv('twitter_accounts.csv')

X = data.drop('label', axis=1)  # Features

y = data['label']  # Labels

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create an SVM model

svm = SVC(kernel='linear', C=1.0)

# Train the SVM model

svm.fit(X_train, y_train)

# Make predictions on the testing set

y_pred = svm.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred)
```

```
print("Accuracy: {:.2f}%".format(accuracy * 100))

print("Precision: {:.2f}%".format(precision * 100))

print("Recall: {:.2f}%".format(recall * 100))

print("F1-score: {:.2f}%".format(f1 * 100))
```

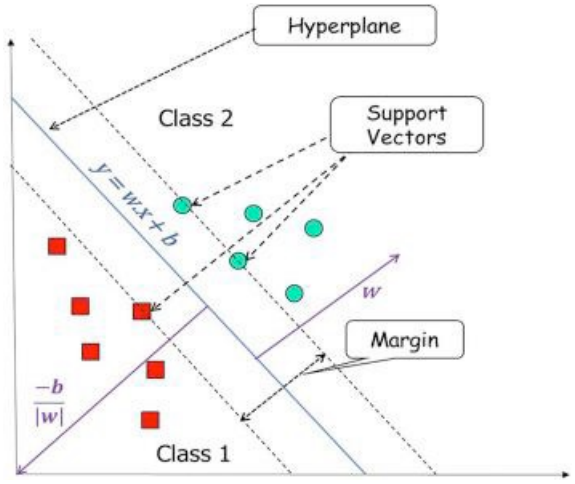


Figure 2.2.3.1: SVM algorithm

In this code, we first import the required libraries, including pandas for data handling, train_test_split for data splitting, SVC for SVM model, and accuracy_score, precision_score, recall_score, and f1_score for evaluation metrics.

Next, we load and preprocess the dataset, where data represents the dataset and X and y represent the features and labels respectively. We then split the dataset into training and testing sets using the train_test_split function.

We create an SVM model with a linear kernel and regularization parameter C set to 1.0. We fit the model to the training data using the fit method.

We make predictions on the testing set using the predict method, and then evaluate the model's performance using the evaluation metrics of accuracy, precision, recall, and F1-score.

2.2 Fake Account detection using Random Forest Algorithm

2.2.1 Introduction

Random Forest is an ensemble learning algorithm that combines the predictions of multiple decision trees to produce a more accurate and robust classifier or regressor. It was introduced by Leo Breiman and Adele Cutler in 2001 and has since become a popular and widely used machine learning technique due to its effectiveness in handling a wide range of tasks, including classification, regression, and feature selection, as well as its ability to handle complex and high-dimensional data.

The basic idea behind the Random Forest algorithm is to build a collection of decision trees, where each tree is trained on a random subset of the data and a random subset of features at each split. This introduces randomness and diversity in the model, as each tree is trained on a different subset of data and features. The predictions of all the trees are then combined to produce the final output.

The Random Forest algorithm consists of the following steps:

- Random sampling of data: A random subset of the original dataset is sampled with replacement, also known as bootstrap sampling. This creates a new dataset of the same size as the original dataset, but with some data points repeated and others omitted.
- Random selection of features: At each split in the decision tree, only a random subset of features (typically a square root of the total number of features) is considered for splitting. This ensures that each tree in the forest is trained on a different set of features, increasing the diversity of the trees.
- Building decision trees: A decision tree is built on the bootstrap sampled data and the randomly selected features. The tree is recursively split based on the selected features, using a split criterion such as Gini impurity or entropy, to create nodes that best separate the data points into different classes or predict the target values.
- Combining predictions: Once all the trees in the forest are built, the predictions of each tree are combined to produce the final output. For classification tasks, the mode of the predicted

class labels is taken as the final prediction, while for regression tasks, the average of the predicted values is taken.

2.3.2 Merits, Demerits and Challenges

Merits of Random Forest algorithm:

- Improved accuracy: Random Forest algorithm typically achieves higher accuracy compared to single decision tree models, as the combination of multiple trees helps to reduce overfitting and improve generalization performance.
- Robustness to noise: Random Forest is robust to noisy data and outliers, as the majority voting or averaging of predictions from multiple trees helps to reduce the impact of individual errors.
- Feature selection: The random selection of features at each split helps in feature selection, as only a subset of features is considered in each tree. This can result in better feature importance estimation and more robust models.
- Ability to handle high-dimensional data: Random Forest can effectively handle high-dimensional data with a large number of features, as the random feature selection helps to capture relevant feature interactions and reduce the curse of dimensionality.
- Ensemble of models: Random Forest is an ensemble learning algorithm, which combines the predictions of multiple decision trees. This results in a more reliable and stable model, as it reduces the risk of overfitting and improves the model's ability to generalize well on unseen data.

Demerits of Random Forest algorithm:

- Computational complexity: Training multiple decision trees can be computationally expensive, especially for large datasets with a large number of trees. However, this can be mitigated by parallelizing the training process and using optimized implementations.
- Interpretability: Random Forest models can be less interpretable compared to single decision trees, as the combination of multiple trees makes it harder to understand the model's

decision-making process. This can be a limitation in scenarios where model interpretability is crucial.

- Hyperparameter tuning: Random Forest has several hyperparameters, such as the number of trees, the maximum depth of trees, and the number of features to consider at each split, which need to be tuned to obtain optimal performance. This can be time-consuming and require careful optimization.

Challenges of Random Forest algorithm:

- Overfitting: Although Random Forest is designed to reduce overfitting compared to single decision trees, there is still a risk of overfitting, especially if the number of trees in the forest is too large or the trees are allowed to grow too deep.
- Interpretability: As mentioned earlier, Random Forest models can be less interpretable due to the combination of multiple trees, which can make it challenging to explain the model's predictions to stakeholders or interpret the important features.
- Scalability: Random Forest can be computationally expensive, especially for large datasets or when a large number of trees are used in the forest. Scalability can be a challenge in scenarios where computational resources are limited.

2.3.3 Implementation of Random Forest Algorithm

While building a Random Forest tree classifier, the main thing is to select the best attribute from the total features list of the dataset for the root nodes as well as for sub-nodes. The selection of best attributes is being achieved with the help of a technique known as the Attribute selection measure (ASM).

- Select the best Features using Attribute Selection Measures(ASM) to split the records.

- Make that attribute/feature a decision node and break the dataset into smaller subsets.

- Start the tree-building process by repeating this process recursively for each child until one of the following condition is being achieved :

  a) All tuples belonging to the same attribute value.

  b) There are no more of the attributes remaining.

  c) There are no more instances remaining.

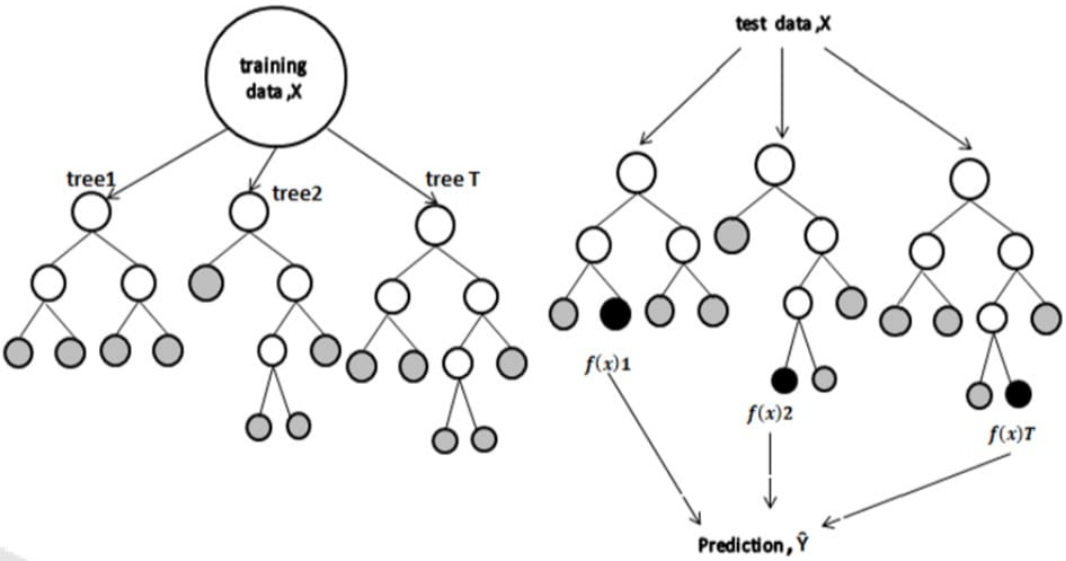


Figure 2.3.3.1 Architecture of Random Forest Classifier

Steps for implementing the Random Forest algorithm for fake account detection:

Step 1: Load and Pre-process the Data

- Load the dataset containing the features and labels for fake account detection. This dataset should be properly pre-processed and formatted.
- Split the dataset into features (X) and labels (y).

Step 2: Split the Data

- Split the data into training and testing sets using the train_test_split function from scikit-learn. This helps in evaluating the performance of the model on unseen data.
- Specify the test size (e.g., 20% or 30%) and a random seed for reproducibility.

Step 3: Create and Train the Random Forest Model

- Create an instance of the Random Forest Classifier using RandomForestClassifier from scikit-learn.
- Specify the hyperparameters for the model, such as the number of estimators (i.e., the number of trees in the forest), maximum depth of the trees, and random state for reproducibility.
- Fit the model to the training data using the fit method, which trains the Random Forest model on the training data.

Step 4: Make Predictions

- Use the trained Random Forest model to make predictions on the test data using the predict method. This predicts the labels for the test data based on the learned patterns from the training data.

Step 5: Evaluate the Model

- Evaluate the performance of the Random Forest model by comparing the predicted labels with the actual labels (ground truth) of the test data.
- Calculate accuracy, precision, recall, F1-score, and other relevant metrics to assess the model's performance.
- You can also visualize the results using confusion matrix, ROC curve, or other relevant plots.

Step 6: Fine-tune the Model (Optional)

- You can further fine-tune the Random Forest model by adjusting its hyperparameters, such as the number of estimators, maximum depth, or other parameters.

- Perform cross-validation, hyperparameter tuning, or other techniques to optimize the model's performance.

Step 7: Deploy the Model (Optional)

- Once the Random Forest model is trained and fine-tuned, you can deploy it in a real-world environment for detecting fake accounts in real-time data.
- Deploy the model on a web server, cloud platform, or other suitable infrastructure for online detection of fake accounts.

Fake Account Detection using the Random Forest algorithm in Python using scikit-learn library:

```
# Import libraries

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load and preprocess the data

# Assume that 'X' is the feature matrix and 'y' is the label vector

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Random Forest model

clf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
```

```
clf.fit(X_train, y_train)

# Make predictions on the test data

y_pred = clf.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

f1_score = f1_score(y_test, y_pred)

print("Accuracy: {:.4f}".format(accuracy))

print("Precision: {:.4f}".format(precision))

print("Recall: {:.4f}".format(recall))

print("F1-Score: {:.4f}".format(f1_score))
```

CHAPTER 3
PROPOSED SYSTEM

CHAPTER 3
PROPOSED SYSTEM

3.1. Objective of Proposed Model

Fake and clone profiles have become a very serious social threat. As information like phone number, email id, school or college name, company name, location etc. are readily exposed in social networks, hackers can easily hack this information to create fake or clone profiles. They then try to cause various attacks like phishing, spamming, cyberbullying etc. They even try to defame the legitimate owner or the organization. So, a detection method has been proposed which can detect both fake and clone profiles to make the social life of the users more secure.

The proposed architecture consists of modules for Fake Profile detection and Clone Profile detection.

A. Fake Profile Detection

This module is used to detect fake Twitter profiles. Here fake profiles are detected based on rules that effectively distinguish fake profiles from genuine ones. Some of the rules that are used to detect fake profiles are - usually fake profiles do not have profile name or image. They do not include any description about the account. The geo-enabled field will be false as they do not want to expose their location in tweets.

They usually make large number of tweets or sometimes the profiles would not have made any tweets etc. The rules are applied on the profile, for each matching rule, a counter is incremented, if the counter value is greater than pre-defined threshold, then the profile is termed as fake.

B. Clone Profile Detection using Similarity Measures

This module detects clones based on Attribute and Network similarity. User profile is taken as input. User identifying information are extracted from the profile. Profiles which are having attributes matching to that of user's profile are searched. Similarity index is calculated and if the similarity index is greater than the threshold, then the profile is termed as clone, else normal.

Attribute Similarity:

Attribute similarity is calculated based on the similarity of attribute values between the profiles. The attributes that are considered for similarity measurement are Name, ScreenName, Language, Location and Time_zone. Two similarity measures are used to measure the similarity between the attributes – Cosine similarity and Levenshtein distance. Cosine similarity is used to find similarity between words and Levenshtein distance is used to find similarity between two sequences.

System requirements:

H/W System Configuration: -

- Processor - Pentium –IV (Entry level processor)
- RAM - 4 GB (min)
- Hard Disk - 20 GB
- Keyboard - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - Any basic monitor

Software Requirements:

- Operating system : Windows 7 Ultimate.
- Coding Language : Python.
- Front-End : Python.
- Back-End : Django
- Designing : Html, css, javascript.
- DataBase : MySQL (WAMP Server).

3.2. Algorithms Used for Proposed Model

Decision Tree algorithm

Decision tree algorithm is a popular machine learning algorithm used for classification and regression problems. It is a tree-based model that recursively partitions the feature space into smaller regions, where each region corresponds to a leaf node or a decision made by the algorithm. The algorithm is called a decision tree because it is represented as a tree-like structure where each node represents a test on a feature, and the branches represent the possible outcomes of the test.

The decision tree algorithm builds the tree by selecting the best feature to split the data at each node, based on a criterion such as entropy or Gini index, which measures the impurity of the data. The goal of the algorithm is to minimize the impurity of the data at each node, and to maximize the purity of the data at the leaf nodes.

Once the decision tree is built, it can be used for making predictions on new data by traversing the tree from the root to the leaf node that corresponds to the prediction. The prediction is based on the majority class of the training samples that reach the leaf node.

Decision tree algorithm has several advantages, such as its interpretability, as the tree structure can be easily visualized and understood by humans. It is also computationally efficient, as it can handle large datasets and high-dimensional feature spaces. Additionally, decision trees can handle both numerical and categorical features, and they are robust to outliers and missing values.

Similarity measure algorithm

Similarity measure algorithms are used to determine the similarity or dissimilarity between two objects or sets of objects. These algorithms are commonly used in machine learning and data mining applications to compare and cluster data points based on their similarities or differences. Two popular similarity measure algorithms are the Levenshtein distance and cosine similarity.

The Levenshtein distance algorithm measures the minimum number of operations required to transform one string into another. The operations can be insertions, deletions, or substitutions of a single character. This algorithm is commonly used for text analysis tasks such as spell checking and language recognition. In the context of detecting fake and clone accounts on Twitter, Levenshtein distance can be used to compare the content of tweets or user profile information and identify accounts with similar or identical content.

Cosine similarity measures the cosine of the angle between two vectors in a high-dimensional space. It is commonly used in natural language processing and information retrieval applications to compare the similarity of two documents or texts. In the context of detecting fake and clone accounts on Twitter, cosine similarity can be used to compare the vectors of user profile features such as the number of followers, the number of tweets, and the frequency of tweets. Accounts with similar feature vectors are likely to be clones.

Similarity measure algorithms have several advantages, such as their ability to handle high-dimensional data and their robustness to noise and outliers. They are also relatively simple and computationally efficient. However, the performance of these algorithms depends on the choice of similarity measure and the representation of the data. Different similarity measures may be more appropriate for different types of data or applications, and the choice of representation can significantly impact the results.

3.3. Designing

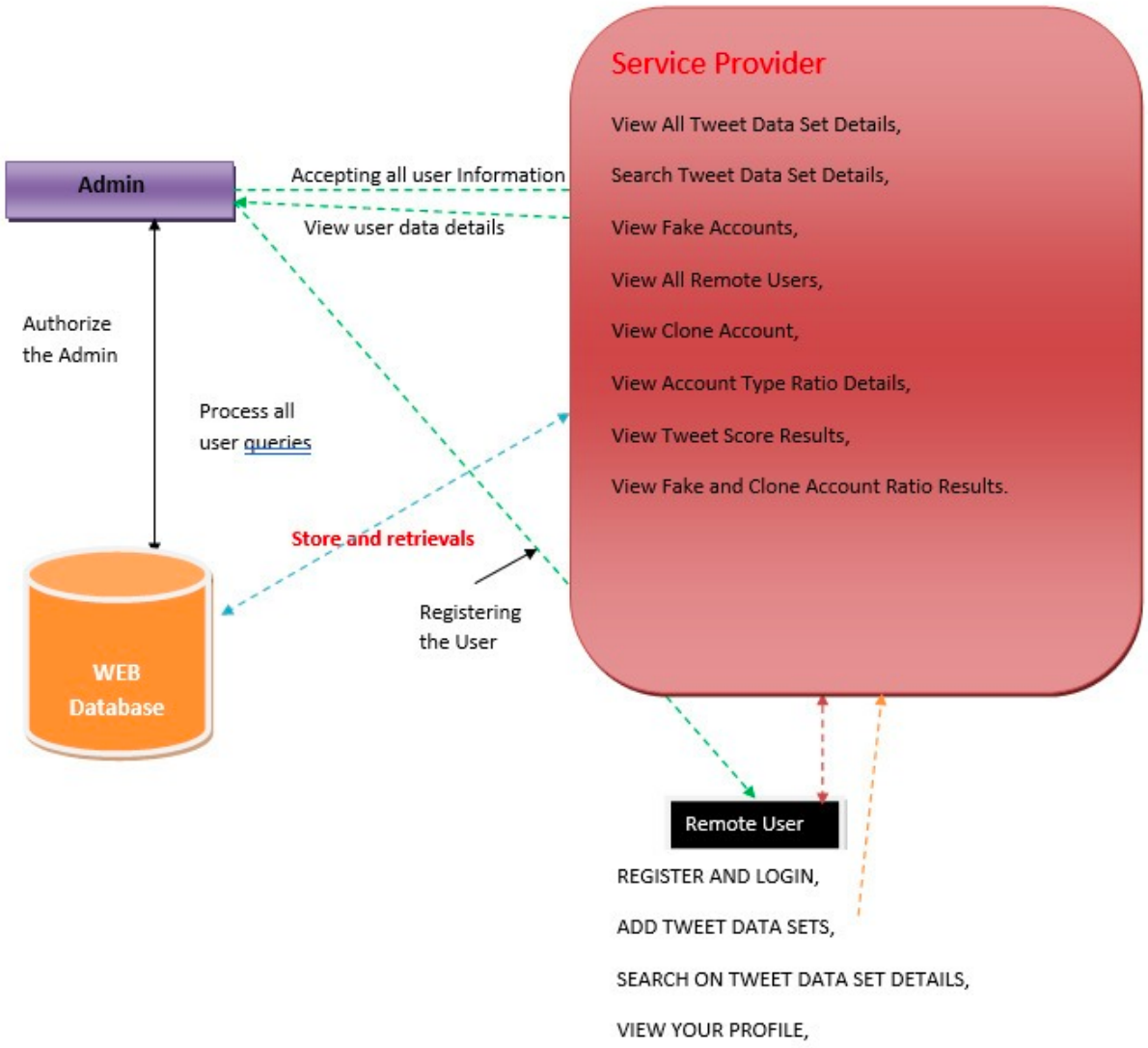


Figure 3.3.1 ARCHITECTURE DIAGRAM

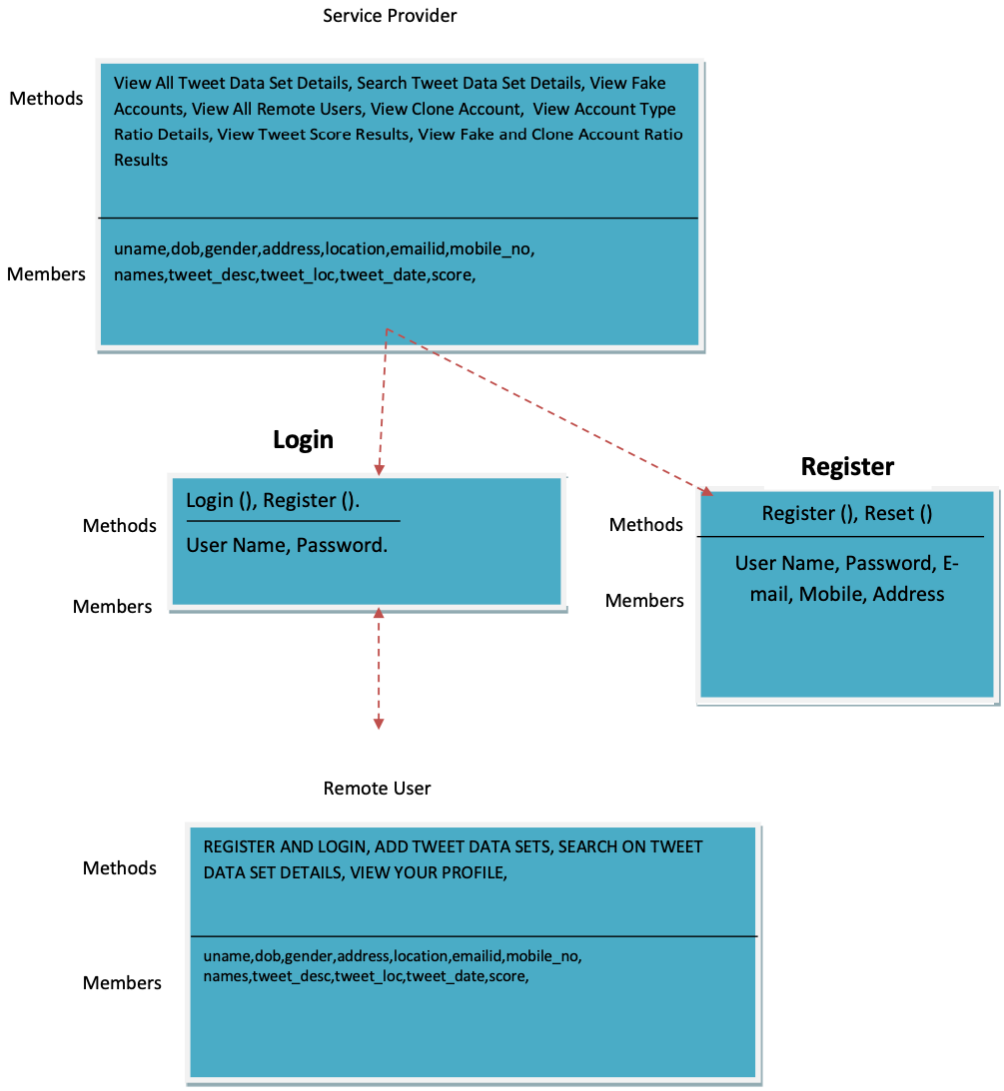


Figure 3.3.2 Class Diagram

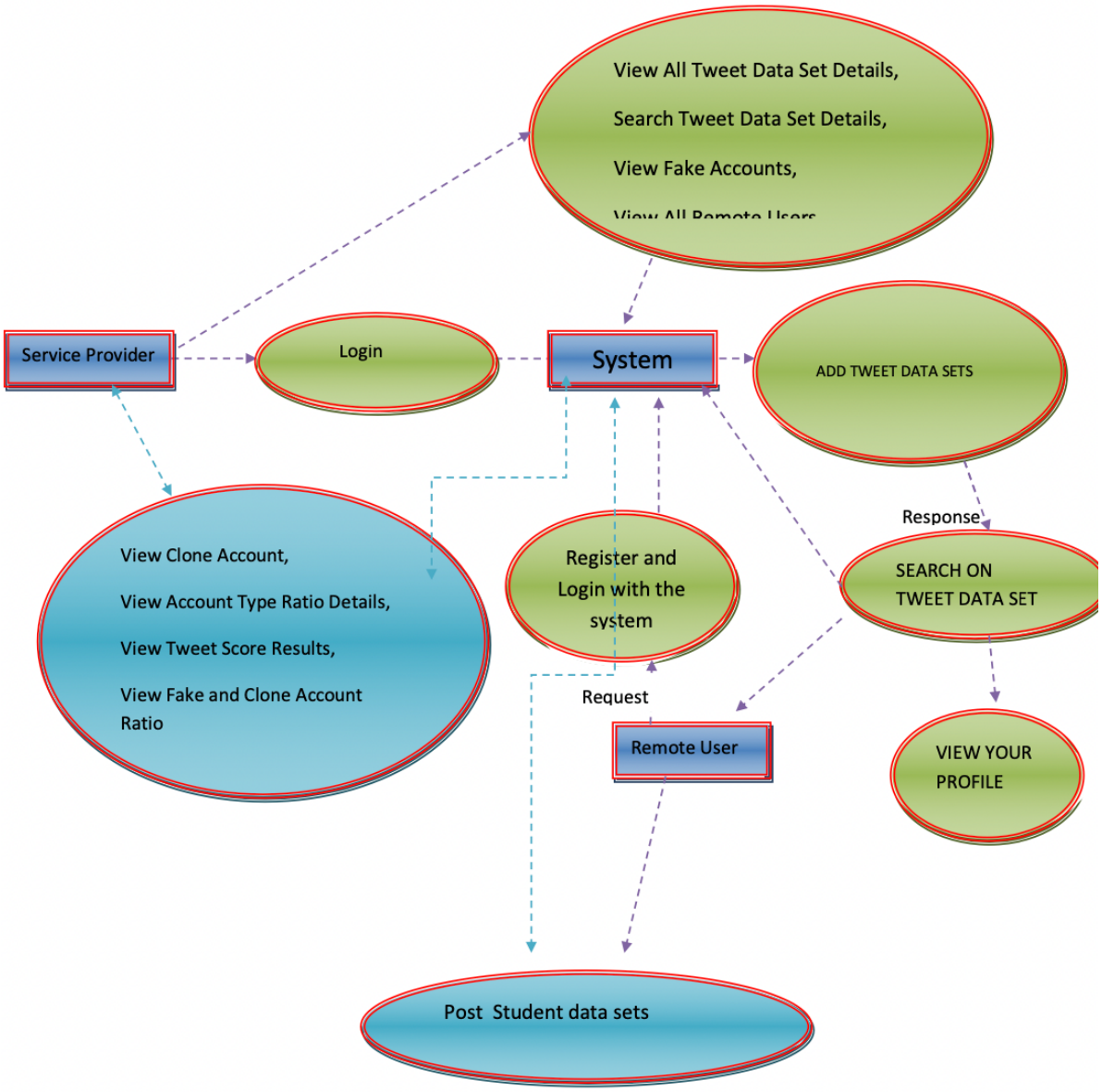
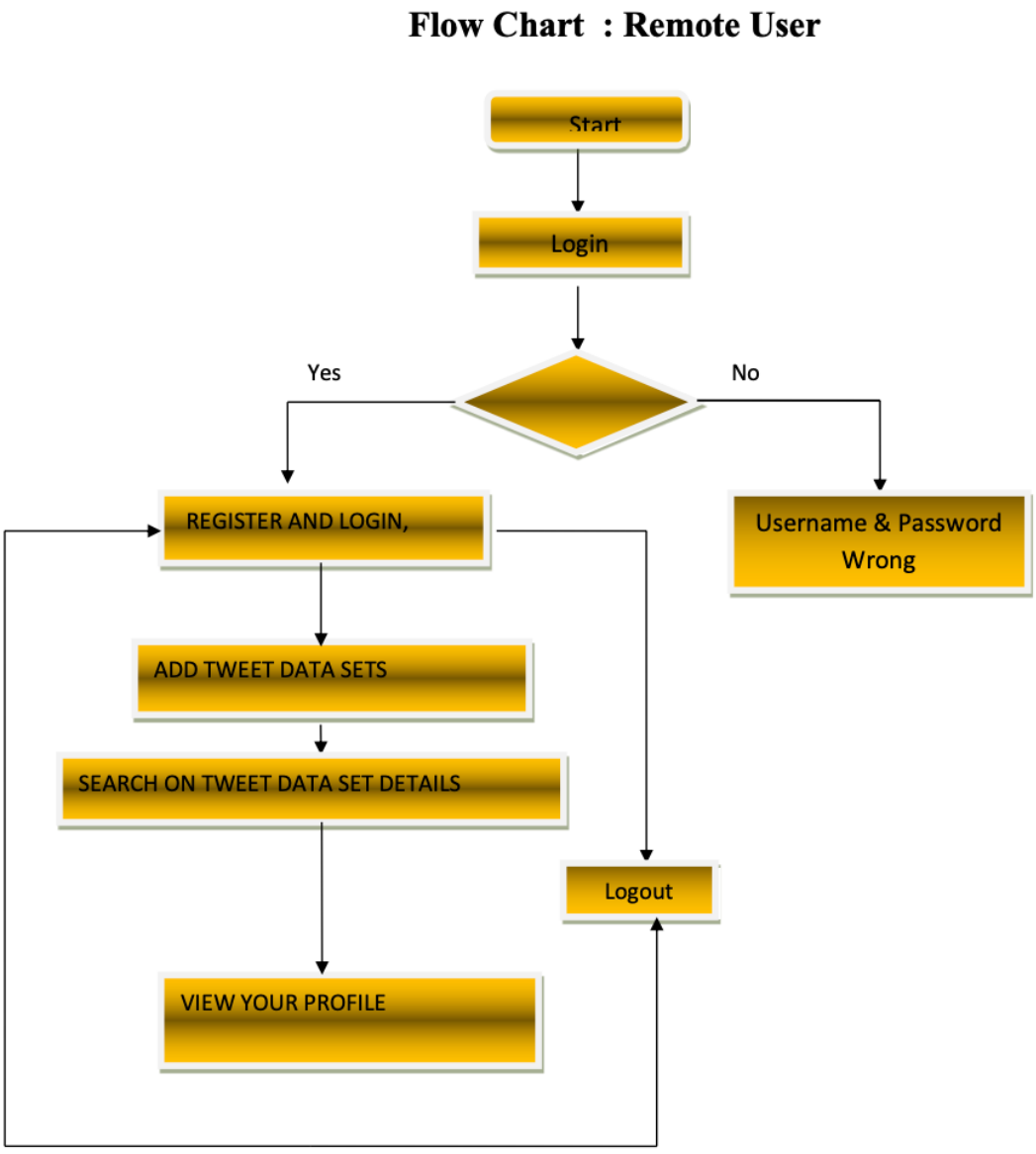
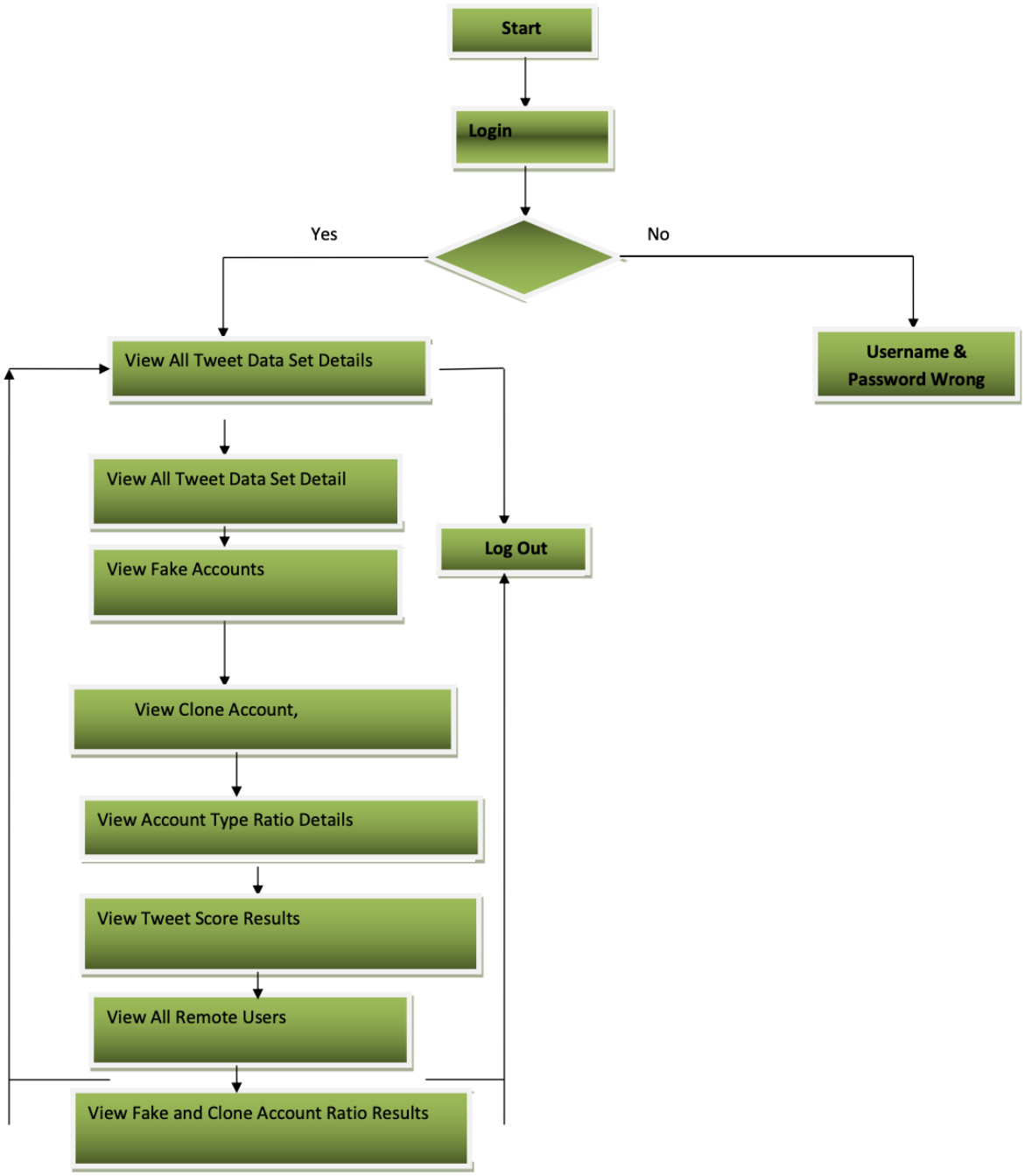


Figure 3.3.3 Data Flow Diagram

Figure 3.3.4 Flow Chart Diagram



Flow Chart : Service Provider



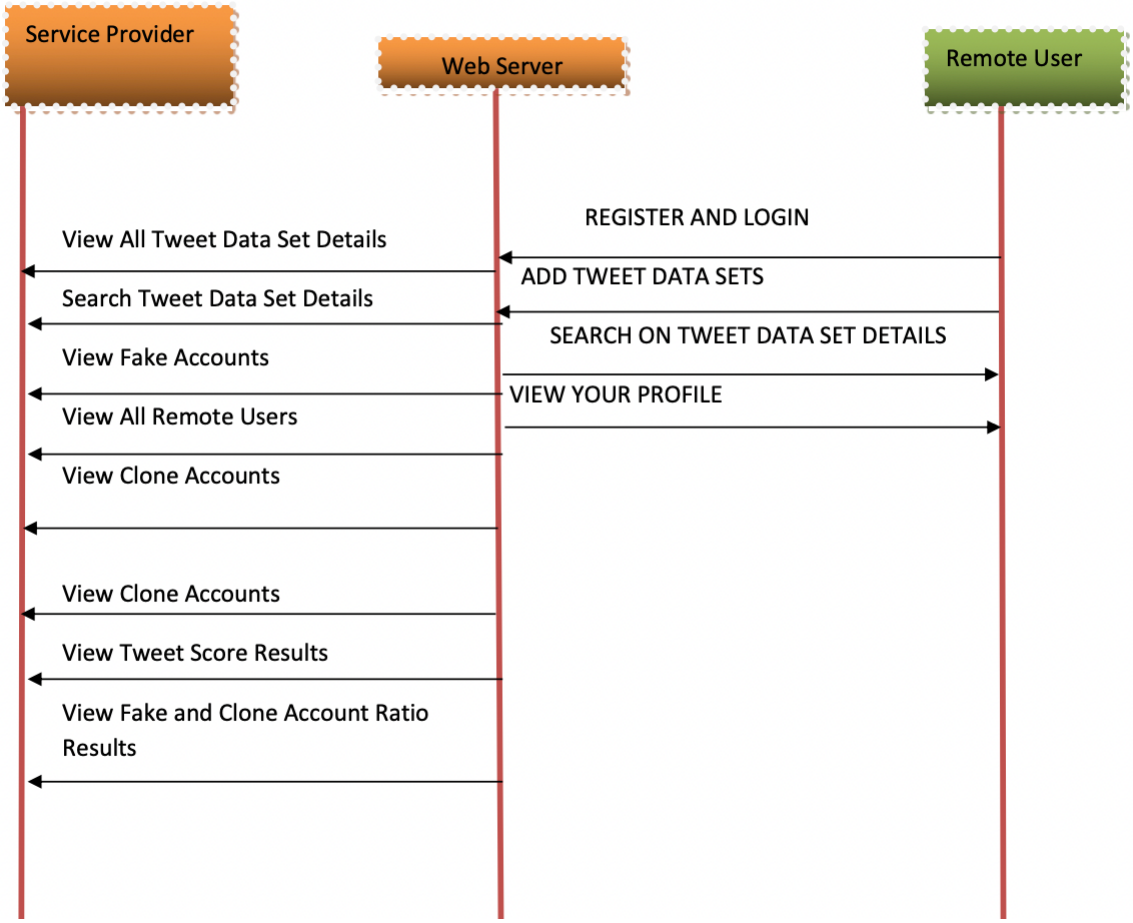


Figure 3.3.5 Sequence Diagram

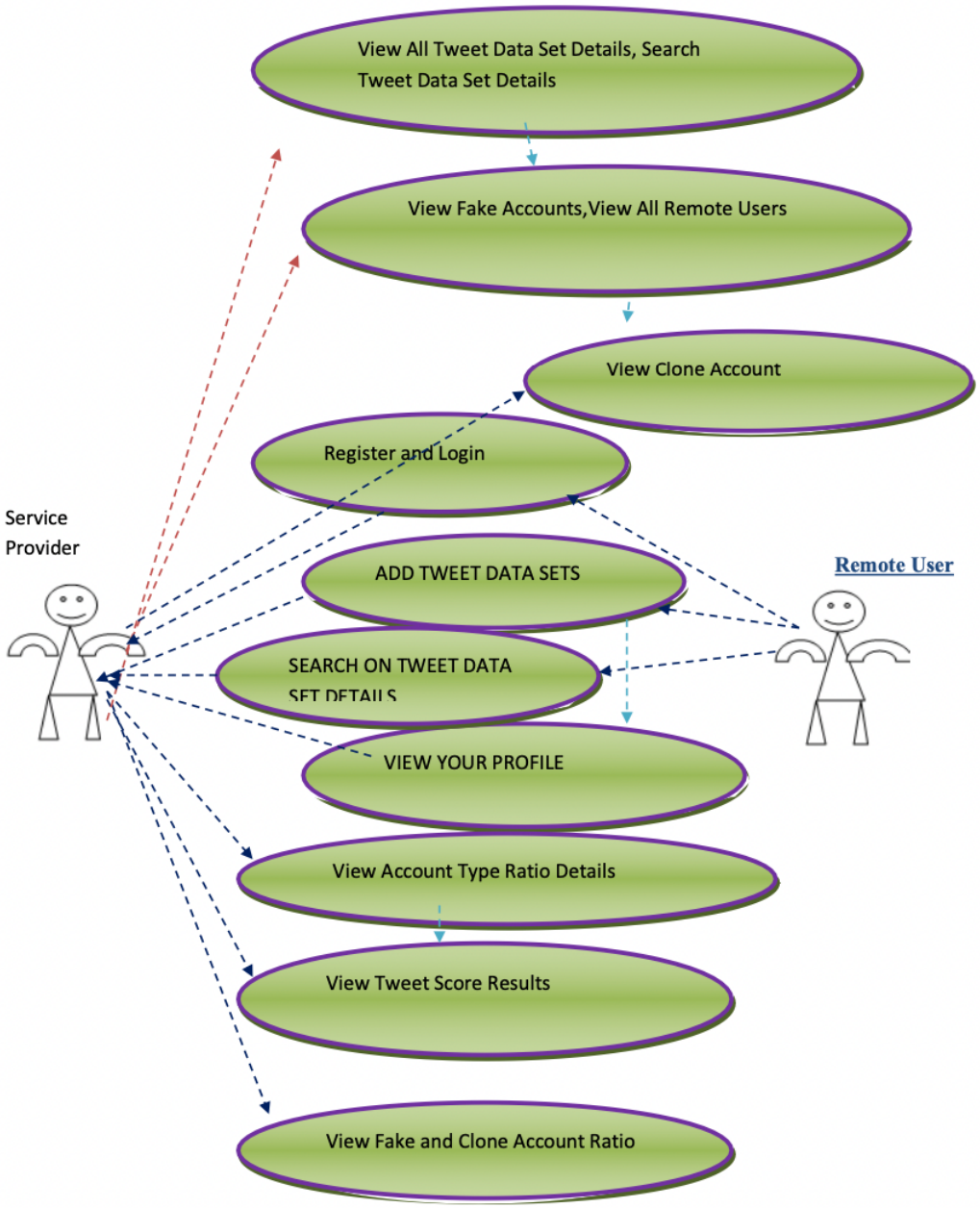


Figure 3.3.6 Use Case Diagram

3.4. Stepwise Implementation and Code

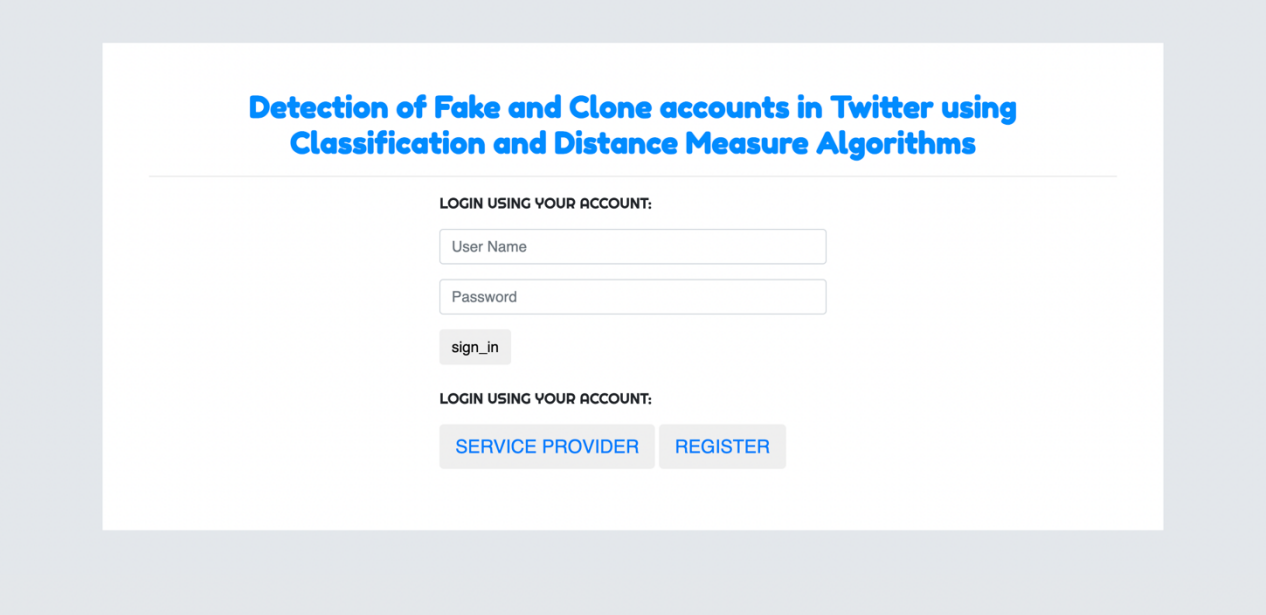


Figure 3.4.1 Remote User Login Page

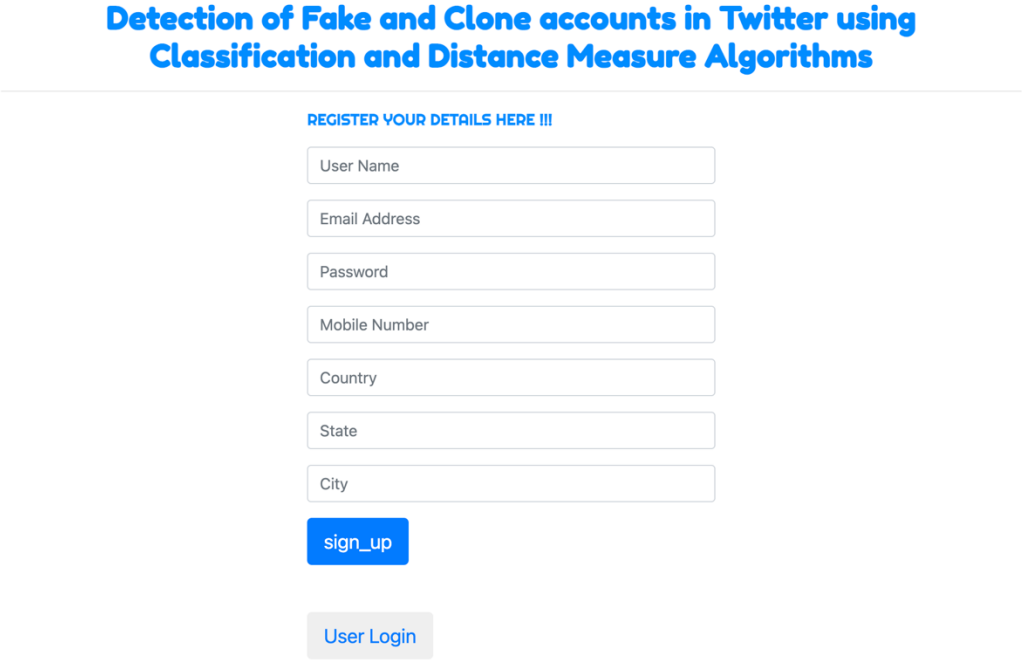


Figure 3.4.2 Remote User Register Page

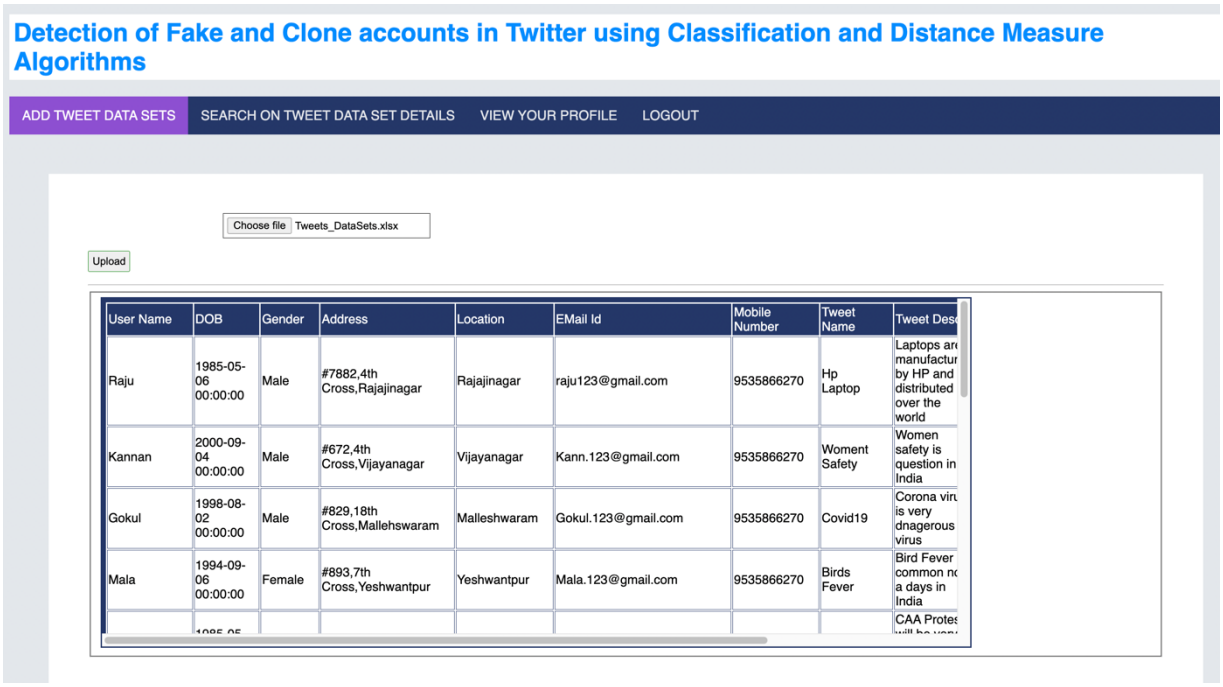


Figure 3.4.3 Remote User Data upload page

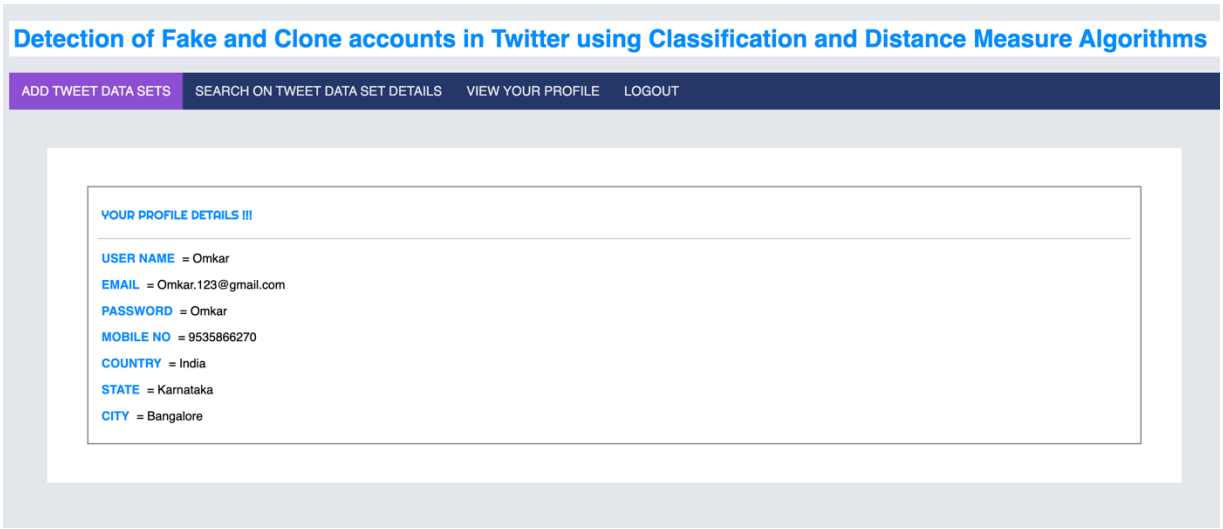


Figure 3.4.4 Remote user profile details page

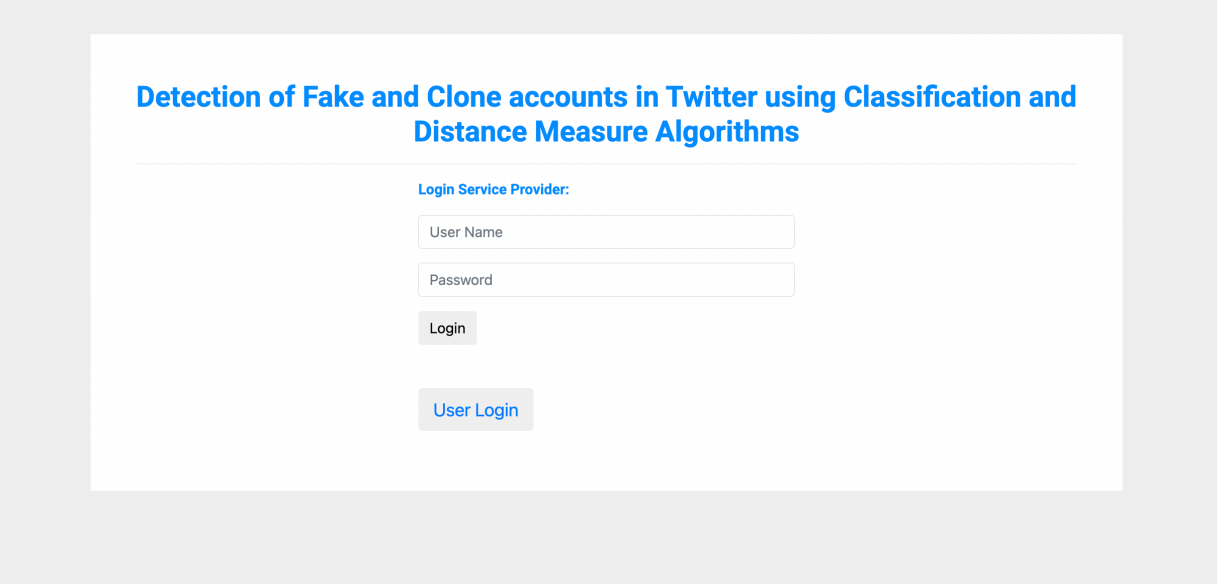


Figure 3.4.5 Admin (Service Provider) Login Page

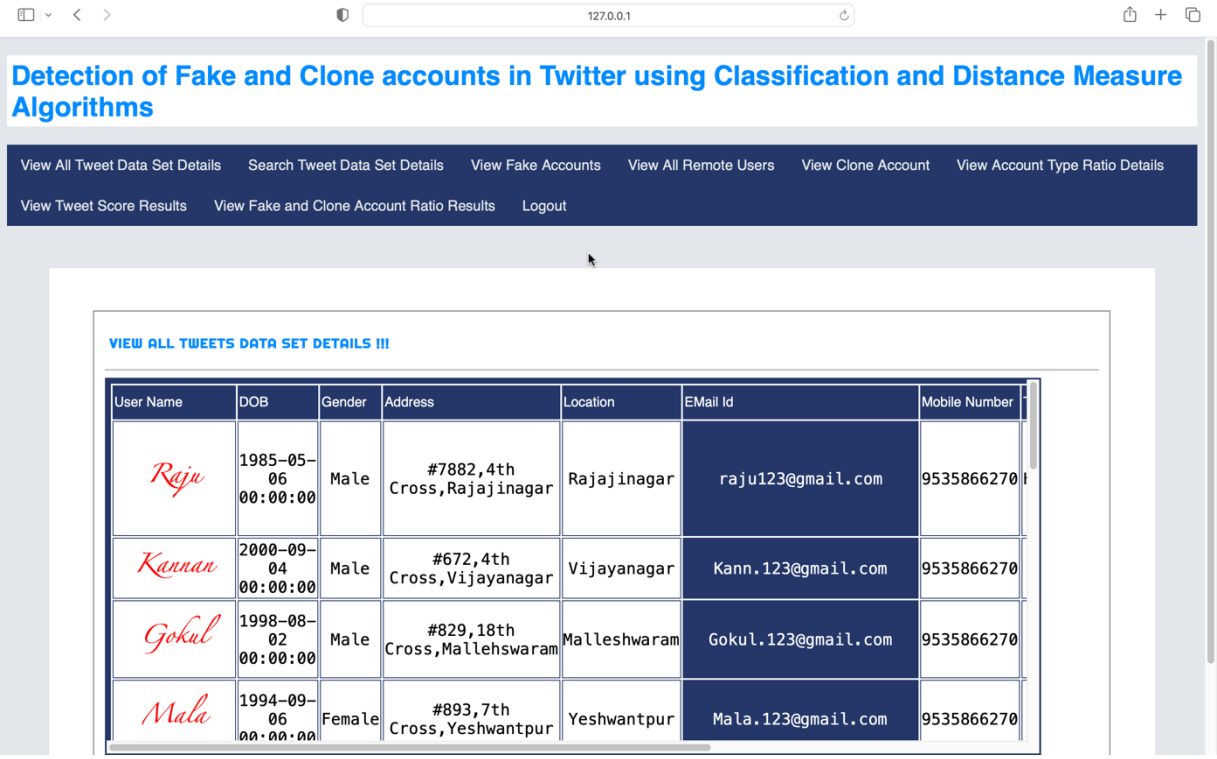


Figure 3.4.6 View All Tweet Data Set Details Page

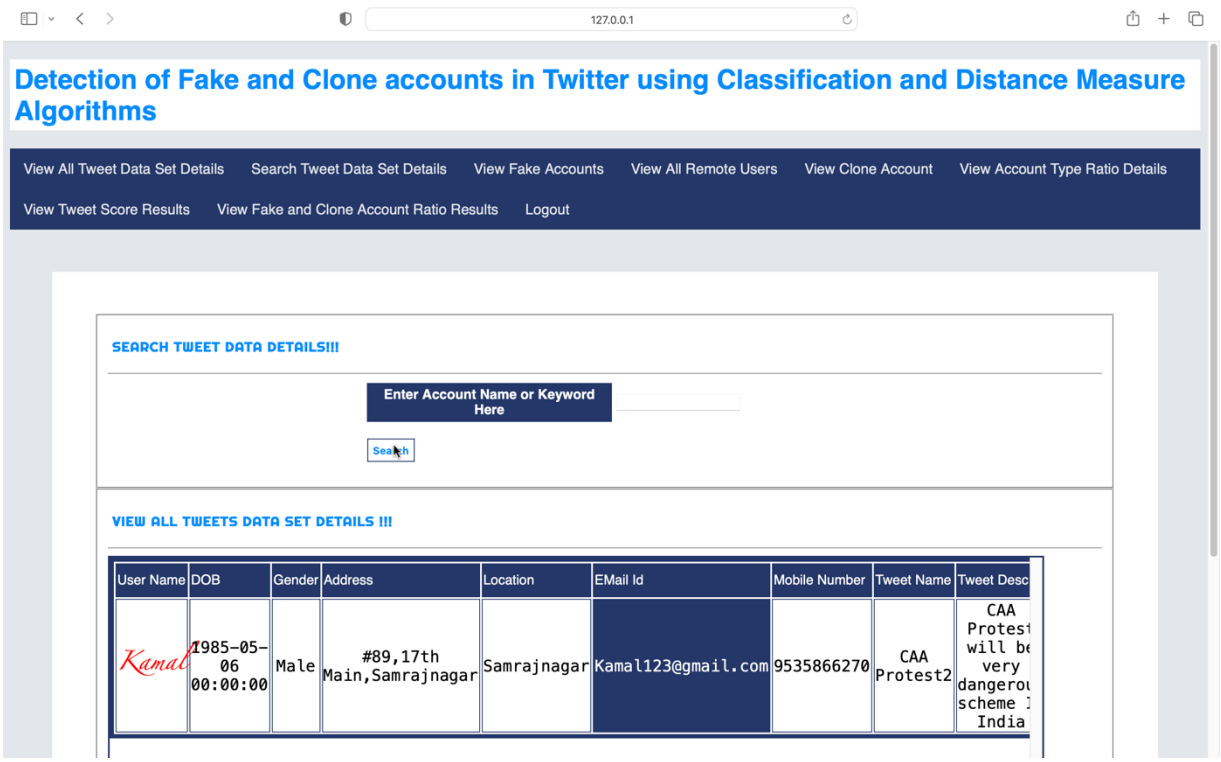


Figure 3.4.7 Search Tweet Data Details

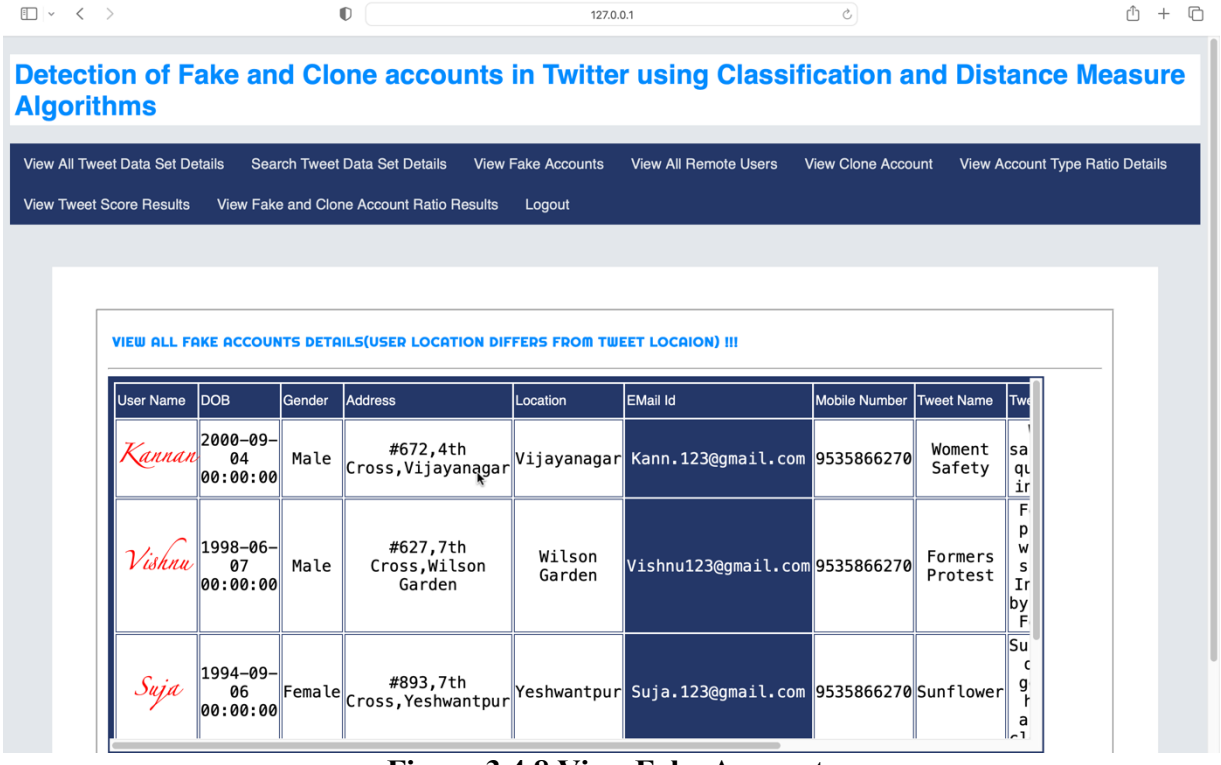


Figure 3.4.8 View Fake Accounts

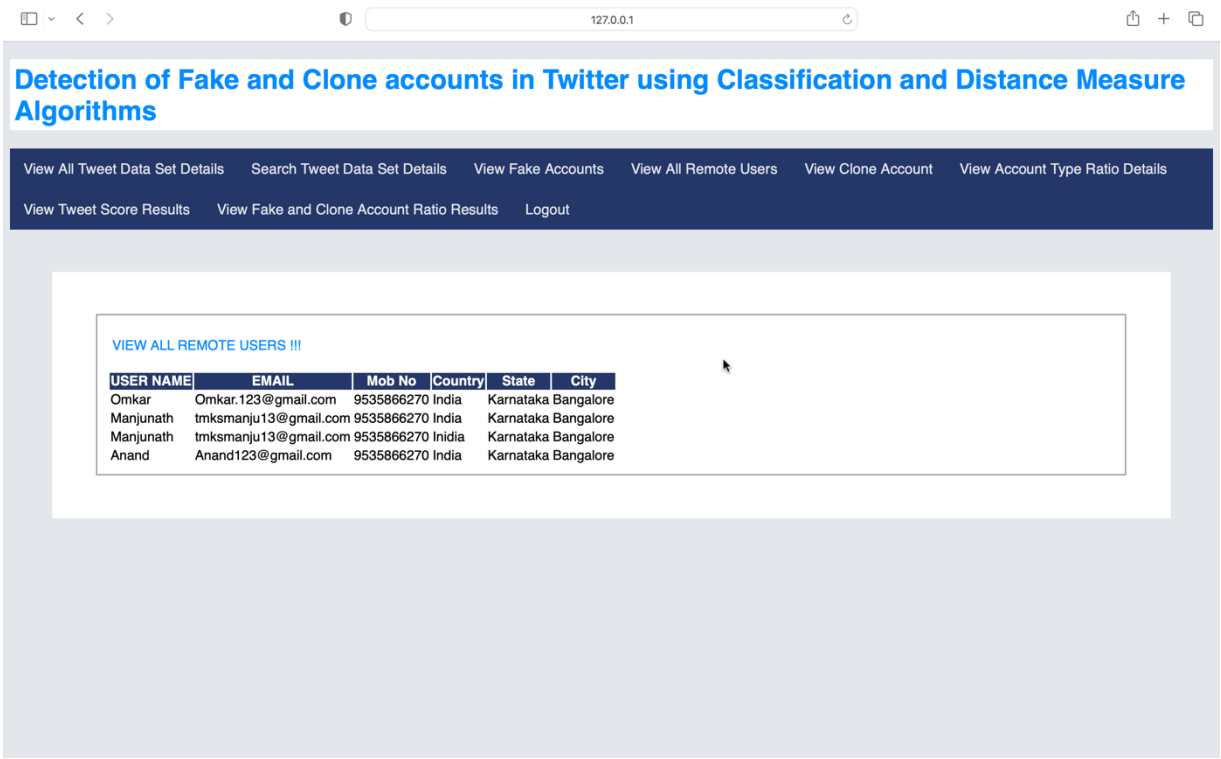


Figure 3.4.9 View Remote Users

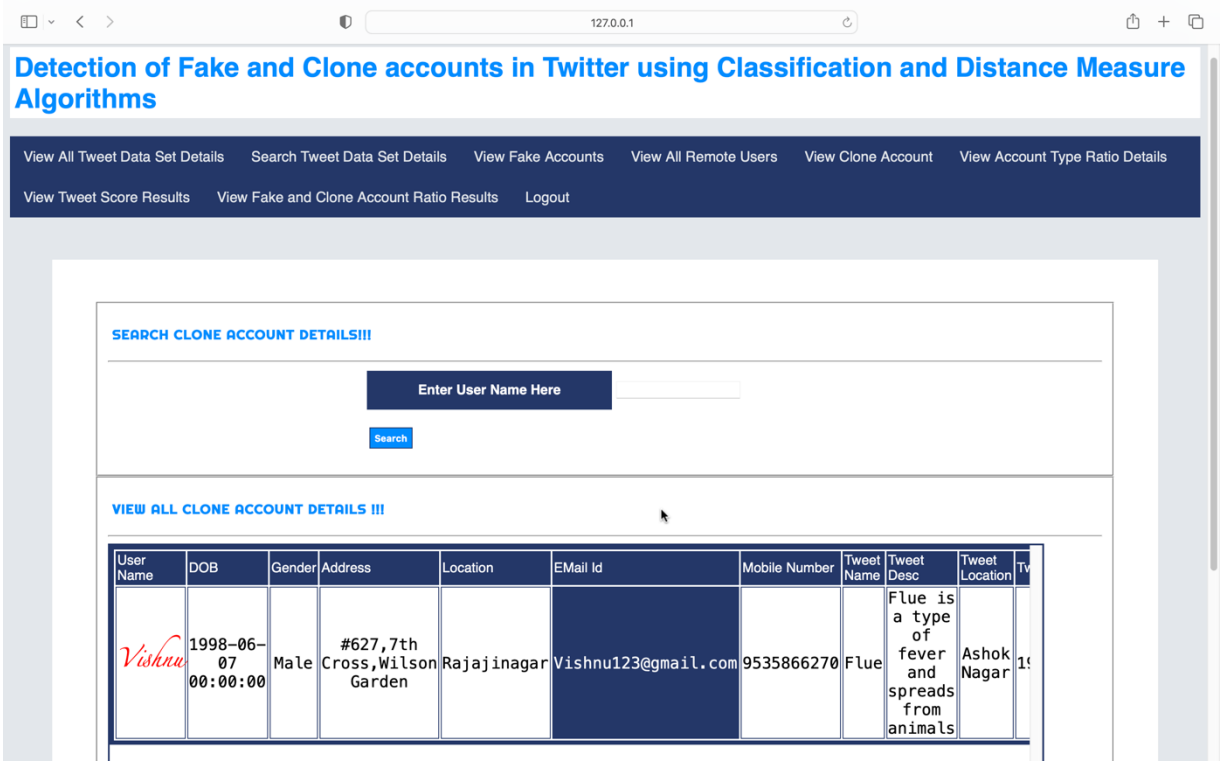


Figure 3.4.10 View Clone Account Details

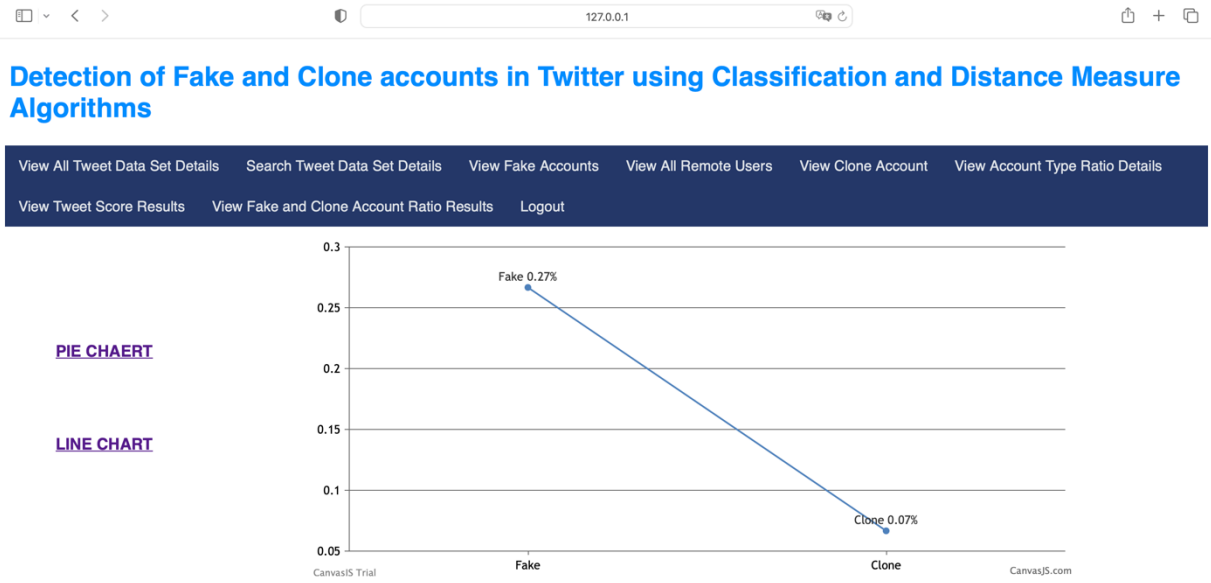


Figure 3.4.11 Account Type Ratio Details

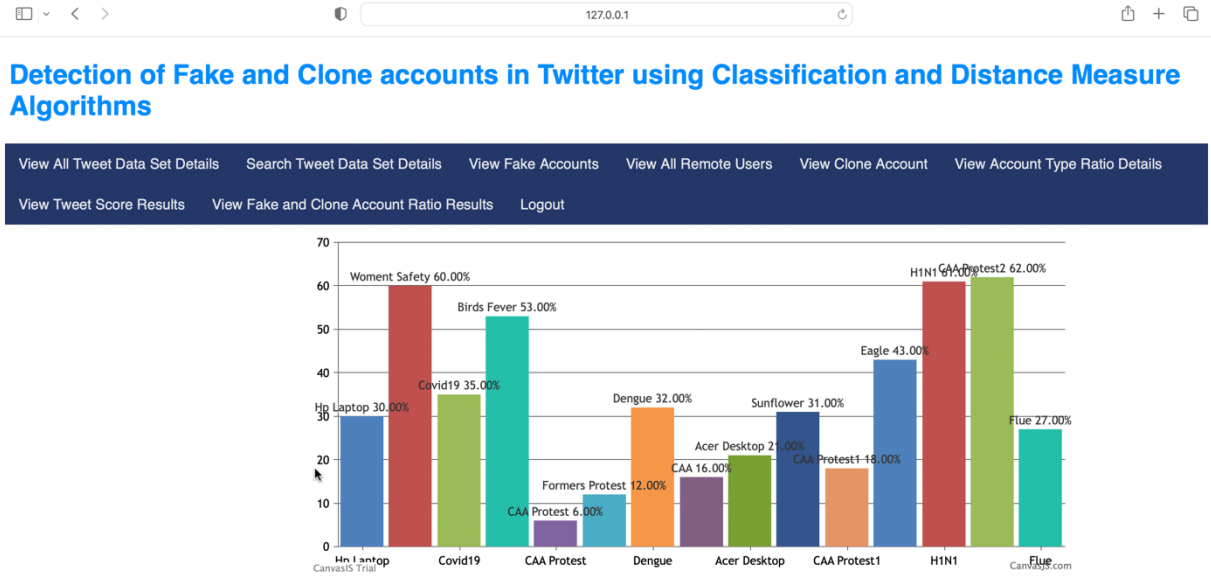


Figure 3.4.12 Tweet Score Results

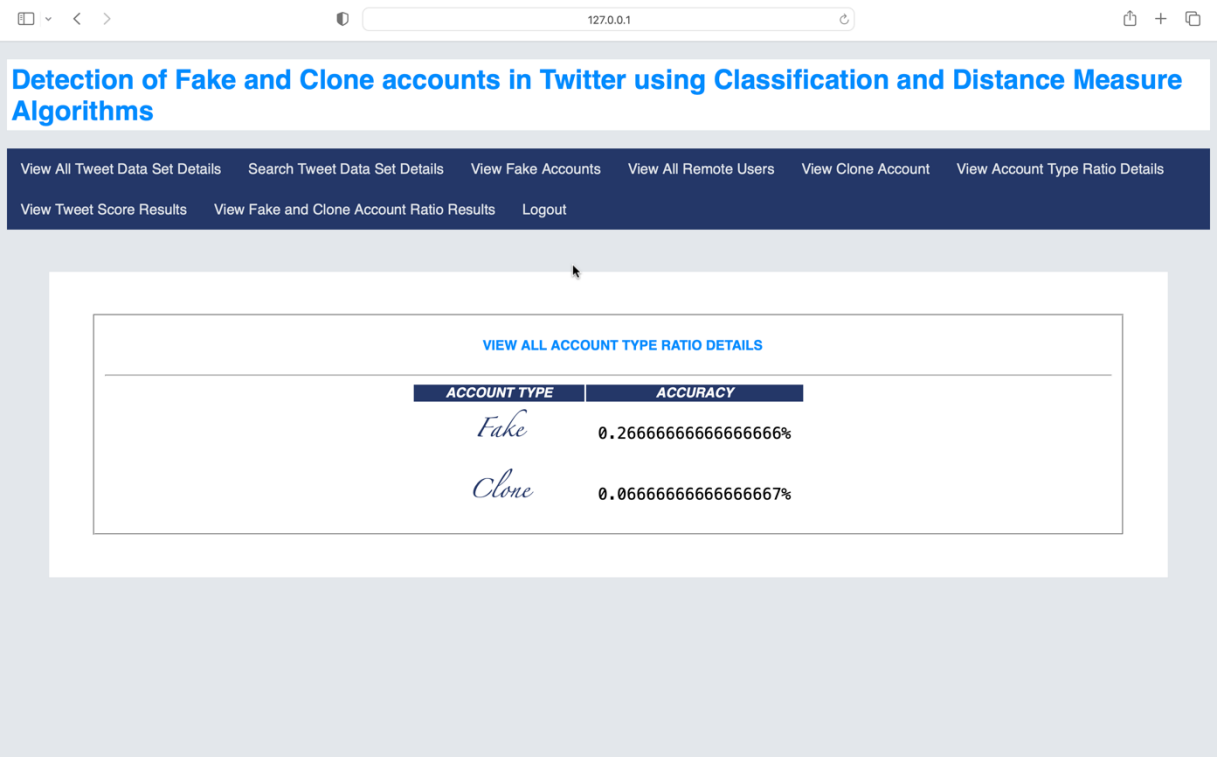


Figure 3.4.13 View Fake and Clone Account Ratio Results

Code:

Remote User – views.py

```
from django.db.models import Count
from django.db.models import Q
from django.shortcuts import render, redirect, get_object_or_404
import datetime
import openpyxl

from Remote_User.models import
review_Model,ClientRegister_Model,clone_accounts_model,recommend_Model,tweet_accuracy_m
odel,fake_accounts_model


def login(request):


    if request.method == "POST" and 'submit1' in request.POST:

        username = request.POST.get('username')
        password = request.POST.get('password')
        try:

            enter = ClientRegister_Model.objects.get(username=username, password=password)
            request.session["userid"] = enter.id
            return redirect('Add_DataSet_Details')
        except:
            pass

    return render(request,'RUser/login.html')

def Add_DataSet_Details(request):
    if "GET" == request.method:
        return render(request, 'RUser/Add_DataSet_Details.html', {})
    else:
        excel_file = request.FILES["excel_file"]

        # you may put validations here to check extension or file size

        wb = openpyxl.load_workbook(excel_file)

        # getting all sheets
        sheets = wb.sheetnames
        print(sheets)

        # getting a particular sheet
        worksheet = wb["Sheet1"]
        print(worksheet)

        # getting active sheet
```

```
        active_sheet = wb.active
        print(active_sheet)

        # reading a cell
        print(worksheet["A1"].value)

        excel_data = list()
        # iterating over the rows and
        # getting value from each cell in row
        for row in worksheet.iter_rows():
            row_data = list()
            for cell in row:
                row_data.append(str(cell.value))
                print(cell.value)
            excel_data.append(row_data)

            clone_accounts_model.objects.all().delete()
            tweet_accuracy_model.objects.all().delete()
    for r in range(1, active_sheet.max_row+1):

        clone_accounts_model.objects.create(
        uname=active_sheet.cell(r, 1).value,
        dob=active_sheet.cell(r, 2).value,
        gender=active_sheet.cell(r, 3).value,
        address=active_sheet.cell(r, 4).value,
        location=active_sheet.cell(r, 5).value,
        mailid=active_sheet.cell(r, 6).value,
        mobile_no=active_sheet.cell(r, 7).value,
        names=active_sheet.cell(r, 8).value,
        tweet_desc=active_sheet.cell(r, 9).value,
        tweet_loc=active_sheet.cell(r, 10).value,
        tweet_date=active_sheet.cell(r, 11).value,
        score=active_sheet.cell(r, 12).value
        )

    return render(request, 'RUser/Add_DataSet_Details.html', {"excel_data": excel_data})


def Register1(request):

    if request.method == "POST":
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        phoneno = request.POST.get('phoneno')
        country = request.POST.get('country')
        state = request.POST.get('state')
        city = request.POST.get('city')
        ClientRegister_Model.objects.create(username=username, email=email, password=password,
phoneno=phoneno,
                                            country=country, state=state, city=city)
```

```
        return render(request, 'RUser/Register1.html')
    else:

        return render(request,'RUser/Register1.html')


def ViewYourProfile(request):
    userid = request.session['userid']
    obj = ClientRegister_Model.objects.get(id= userid)
    return render(request,'RUser/ViewYourProfile.html',{'object':obj})


def Search_Account_Details(request):

    if request.method == "POST":
        kword = request.POST.get('keyword')
        obj = clone_accounts_model.objects.all().filter(Q(uname__contains=kword) |
Q(names__contains=kword))
        return render(request, 'RUser/Search_Account_Details.html',{'objs': obj})
    return render(request, 'RUser/Search_Account_Details.html')

def ratings(request,pk):
    vott1, vott, neg = 0, 0, 0
    objs = clone_accounts_model.objects.get(id=pk)
    unid = objs.id
    vot_count = clone_accounts_model.objects.all().filter(id=unid)
    for t in vot_count:
        vott = t.ratings
        vott1 = vott + 1
        obj = get_object_or_404(clone_accounts_model, id=unid)
        obj.ratings = vott1
        obj.save(update_fields=["ratings"])
        return redirect('Add_DataSet_Details')

    return render(request,'RUser/ratings.html',{'objs':vott1})
```

Service Provider – views.py

```
from django.db.models import  Count, Avg
from django.shortcuts import render, redirect,get_object_or_404
from django.db.models import Count,Subquery
from django.db.models import Q
import datetime
from Remote_User.models import
clone_accounts_model,clone1_accounts_model,ClientRegister_Model,review_Model,recommend_
Model,tweet_accuracy_model,fake_accounts_model


def serviceproviderlogin(request):
    if request.method  == "POST":
```

```
        admin = request.POST.get('username')
        password = request.POST.get('password')
        if admin == "admin" and password =="admin":
            tweet_accuracy_model.objects.all().delete()
            return redirect('View_Remote_Users')

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


def viewtreandingquestions(request,chart_type):
    dd = {}
    pos,neu,neg =0,0,0
    poss=None
    topic =
clone_accounts_model.objects.values('ratings').annotate(dcount=Count('ratings')).order_by('-
dcount')
    for t in topic:
        topics=t['ratings']
        pos_count=clone_accounts_model.objects.filter(topics=topics).values('names').annotate(topicco
unt=Count('ratings'))
        poss=pos_count
        for pp in pos_count:
            senti= pp['names']
            if senti == 'positive':
                pos= pp['topiccount']
            elif senti == 'negative':
                neg = pp['topiccount']
            elif senti == 'nutral':
                neu = pp['topiccount']
        dd[topics]=[pos,neg,neu]
    return
render(request,'SProvider/viewtreandingquestions.html',{'object':topic,'dd':dd,'chart_type':chart_type
})

def Search_Account(request): # Search
    if request.method == "POST":
        kword = request.POST.get('keyword')
        obj = clone_accounts_model.objects.all().filter(Q(uname__contains=kword) |
Q(names__contains=kword))
        return render(request, 'SProvider/Search_Account.html', {'objs': obj})
    return render(request, 'SProvider/Search_Account.html')

def View_Fake_Account(request): # Using SVM
    atype='Fake'

    obj1 = clone_accounts_model.objects.values('uname',
'dob',
'gender',
'address',
'location',
'mailid',
```

```
'mobile_no',
'names',
'tweet_desc',
'tweet_loc',
'tweet_date',
'score'
)

    fake_accounts_model.objects.all().delete();
    for t in obj1:
        uname = t['uname']
        dob = t['dob']
        gender= t['gender']
        address= t['address']
        location= t['location']
        emailid= t['mailid']
        mobile_no= t['mobile_no']
        names= t['names']
        tweet_desc= t['tweet_desc']
        tweet_loc= t['tweet_loc']
        tweet_date= t['tweet_date']
        score= t['score']

        if location!=tweet_loc:
          fake_accounts_model.objects.create(uname=uname,dob = dob,gender = gender,address =
address,location = location,mailid=emailid,mobile_no=mobile_no,names =names,tweet_desc
=tweet_desc,tweet_loc =tweet_loc,tweet_date =tweet_date,score=score)

    obj2 = fake_accounts_model.objects.all()

    count=obj2.count()
    count1=obj1.count()
    accuracy=count/count1

    if accuracy !=0:
        tweet_accuracy_model.objects.create(names=atype,accuracy=accuracy)


    return render(request, 'SProvider/View_Fake_Account.html', {'objs': obj2,'count':accuracy})


def View_Clone_Account(request): # Positive # Using SVM
    atype = 'Clone'
    obj1=''
    kword=''
    obj2 = ''
    ratio=''
    if request.method == "POST":
        kword = request.POST.get('keyword')
        obj1 = clone_accounts_model.objects.values('uname',
                                    'dob',
```

```
                                    'gender',
                                    'address',
                                    'location',
                                    'mailid',
                                    'mobile_no',
                                    'names',
                                    'tweet_desc',
                                    'tweet_loc',
                                    'tweet_date',
                                    'score'
                                    )
        count=0
        clone1_accounts_model.objects.all().delete();
        for t in obj1:
            uname = t['uname']
            dob = t['dob']
            gender = t['gender']
            address = t['address']
            location = t['location']
            emailid = t['mailid']
            mobile_no = t['mobile_no']
            names = t['names']
            tweet_desc = t['tweet_desc']
            tweet_loc = t['tweet_loc']
            tweet_date = t['tweet_date']
            score = t['score']

            if uname == kword:
                count=count+1
                if(count>1):
                  clone1_accounts_model.objects.create(uname=uname, dob=dob, gender=gender,
address=address, location=location,mailid=emailid, mobile_no=mobile_no, names=names,
tweet_desc=tweet_desc,tweet_loc=tweet_loc, tweet_date=tweet_date, score=score)

        obj2 = clone1_accounts_model.objects.all()
        obj1 = clone_accounts_model.objects.all()
        count = obj2.count()
        count1 = obj1.count()
        ratio = count / count1
        if ratio != 0:
            tweet_accuracy_model.objects.create(names=atype, accuracy=ratio)

    return render(request, 'SProvider/View_Clone_Account.html', {'objs': obj2,'count':ratio})


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

def ViewTrendings(request):
```

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

def negativechart(request,chart_type):
    dd = {}
    pos, neu, neg = 0, 0, 0
    poss = None
    topic =
clone_accounts_model.objects.values('ratings').annotate(dcount=Count('ratings')).order_by('-
dcount')
    for t in topic:
        topics = t['ratings']
        pos_count =
clone_accounts_model.objects.filter(topics=topics).values('names').annotate(topiccount=Count('ratin
gs'))
        poss = pos_count
        for pp in pos_count:
            senti = pp['names']
            if senti == 'positive':
                pos = pp['topiccount']
            elif senti == 'negative':
                neg = pp['topiccount']
            elif senti == 'nutral':
                neu = pp['topiccount']
        dd[topics] = [pos, neg, neu]
    return
render(request,'SProvider/negativechart.html',{'object':topic,'dd':dd,'chart_type':chart_type})

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

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

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

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

Service Provider – ViewFakeAccount.html

```
{% extends 'SProvider/design1.html' %} {% block researchblock %}

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

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" />
<link
  href="https://fonts.googleapis.com/css?family=Righteous"
  rel="stylesheet"
/>
<link
  href="https://fonts.googleapis.com/css?family=Fredoka+One"
  rel="stylesheet"
/>
<body>
  <div class="container-fluid">
    <div class="container">
      <div class="row">
        <div class="col-md-15">
          <form role="form" method="POST">
            {% csrf_token %}
            <fieldset>
              <p class="text-uppercase pull-center style1">
                VIEW ALL FAKE ACCOUNTS DETAILS(USER LOCATION DIFFERS FROM
TWEET
                LOCAION) !!!
              </p>
              <hr />
              <div class="tweettext">
                <table border="5" bordercolor="#243768">
                  <tr>
                    <td bgcolor="#243768">
                      <span class="style5"> User Name</span>
                    </td>
                    <td bgcolor="#243768"><span class="style5">DOB</span></td>
                    <td bgcolor="#243768">
                      <span class="style5">Gender</span>
                    </td>
                    <td bgcolor="#243768">
                      <span class="style5">Address</span>
                    </td>
                    <td bgcolor="#243768">
                      <span class="style5">Location</span>
                    </td>
                    <td bgcolor="#243768">
                      <span class="style5">EMail Id</span>
                    </td>
                    <td bgcolor="#243768">
                      <span class="style5">Mobile Number</span>
                    </td>
```

```
  <td bgcolor="#243768">
   <span class="style5">Tweet Name</span>
  </td>
  <td bgcolor="#243768">
   <span class="style5">Tweet Desc</span>
  </td>
  <td bgcolor="#243768">
   <span class="style5">Tweet Location</span>
  </td>
  <td bgcolor="#243768">
   <span class="style5">Tweet Date</span>
  </td>
  <td bgcolor="#243768">
   <span class="style5">Tweet Score</span>
  </td>
 </tr>

 {% for object in objs %}
 <tr>
  <td
   style="color: red; font-size: 20px; font-family: fantasy"
  >
   <div align="center">{{object.uname}}</div>
  </td>
  <td style="font-family: monospace; font-size: 19px">
   <div align="center">{{object.dob}}</div>
  </td>
  <td style="font-family: monospace; font-size: 19px">
   <div align="center">{{object.gender}}</div>
  </td>
  <td style="font-family: monospace; font-size: 19px">
   <div align="center">{{object.address}}</div>
  </td>
  <td style="font-family: monospace; font-size: 19px">
   <div align="center">{{object.location}}</div>
  </td>
  <td
   bgcolor="#243768"
   style="
    font-family: monospace;
    font-size: 19px;
    color: white;
   "
  >
   <div align="center">{{object.mailid}}</div>
  </td>
  <td style="font-family: monospace; font-size: 19px">
   <div align="center">{{object.mobile_no}}</div>
  </td>
  <td style="font-family: monospace; font-size: 19px">
   <div align="center">{{object.names}}</div>
```

```
                </td>
                <td style="font-family: monospace; font-size: 19px">
                  <div align="center">{{object.tweet_desc}}</div>
                </td>
                <td style="font-family: monospace; font-size: 19px">
                  <div align="center">{{object.tweet_loc}}</div>
                </td>
                <td style="font-family: monospace; font-size: 19px">
                  <div align="center">{{object.tweet_date}}</div>
                </td>
                <td style="font-family: monospace; font-size: 19px">
                  <div align="center">{{object.score}}</div>
                </td>
              </tr>
              {% endfor %}
            </table>
          </div>
        </fieldset>
      </form>
    </div>
   </div>
  </div>
 </div>
 {% endblock %}
 <tr></tr>
</body>
```

Service Provider – ViewCloneAccount.html

```
{% extends 'SProvider/design1.html' %} {% block researchblock %}

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

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" />
<link
 href="https://fonts.googleapis.com/css?family=Righteous"
 rel="stylesheet"
/>
<link
 href="https://fonts.googleapis.com/css?family=Fredoka+One"
 rel="stylesheet"
/>
<link rel="stylesheet" href="styles-gpk.css" />
<body>
 <div class="container-fluid">
  <div class="container">
   <div class="row">
    <div class="col-md-15">
     <form role="form" method="POST">
```

```
        {% csrf_token %}
        <fieldset>
          <p class="text-uppercase pull-center style1">
            SEARCH CLONE ACCOUNT DETAILS!!!
          </p>
          <hr />

          {% csrf_token %}
          <table width="568" align="center">
            <tr>
              <td width="287" height="44" bgcolor="#243768">
                <div align="center">
                  <span class="style4 style2">Enter User Name Here </span>
                </div>
              </td>
              <td width="269"><input type="text" name="keyword" /></td>
            </tr>
            <tr>
              <td>
                <p>
                  <input
                    name="submit"
                    type="submit"
                    class="style1 btn-gpk"
                    value="Search"
                  />
                </p>
              </td>
            </tr>
          </table>
        </fieldset>
      </form>

      <form role="form" method="POST">
        {% csrf_token %}
        <fieldset>
          <p class="text-uppercase pull-center style1">
            VIEW ALL CLONE ACCOUNT DETAILS !!!
          </p>
          <hr />
          <div class="tweettext">
            <table border="5" bordercolor="#243768">
              <tr>
                <td bgcolor="#243768">
                  <span class="style5"> User Name</span>
                </td>
                <td bgcolor="#243768"><span class="style5">DOB</span></td>
                <td bgcolor="#243768">
                  <span class="style5">Gender</span>
                </td>
                <td bgcolor="#243768">
```

```
            <span class="style5">Address</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">Location</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">EMail Id</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">Mobile Number</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">Tweet Name</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">Tweet Desc</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">Tweet Location</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">Tweet Date</span>
           </td>
           <td bgcolor="#243768">
            <span class="style5">Tweet Score</span>
           </td>
          </tr>

          {% for object in objs %}
          <tr>
           <td
            style="color: red; font-size: 20px; font-family: fantasy"
           >
            <div align="center">{{object.uname}}</div>
           </td>
           <td style="font-family: monospace; font-size: 19px">
            <div align="center">{{object.dob}}</div>
           </td>
           <td style="font-family: monospace; font-size: 19px">
            <div align="center">{{object.gender}}</div>
           </td>
           <td style="font-family: monospace; font-size: 19px">
            <div align="center">{{object.address}}</div>
           </td>
           <td style="font-family: monospace; font-size: 19px">
            <div align="center">{{object.location}}</div>
           </td>
           <td
            bgcolor="#243768"
            style="
             font-family: monospace;
```

```
                font-size: 19px;
                color: white;
              "
            >
              <div align="center">{{object.mailid}}</div>
            </td>
            <td style="font-family: monospace; font-size: 19px">
              <div align="center">{{object.mobile_no}}</div>
            </td>
            <td style="font-family: monospace; font-size: 19px">
              <div align="center">{{object.names}}</div>
            </td>
            <td style="font-family: monospace; font-size: 19px">
              <div align="center">{{object.tweet_desc}}</div>
            </td>
            <td style="font-family: monospace; font-size: 19px">
              <div align="center">{{object.tweet_loc}}</div>
            </td>
            <td style="font-family: monospace; font-size: 19px">
              <div align="center">{{object.tweet_date}}</div>
            </td>
            <td style="font-family: monospace; font-size: 19px">
              <div align="center">{{object.score}}</div>
            </td>
          </tr>
          {% endfor %}
        </table>
      </div>
    </fieldset>
  </form>
 </div>

 <div class="col-md-2">
  <!-------null------>
 </div>
</div>
</div>
</div>
{% endblock %}
<tr></tr>
</body>
```

CHAPTER 4
RESULTS AND DISCUSSION

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Comparison of existing Solution

From the existing solution we can see that random forest has a higher accuracy among all the existing solutions but the tree constructed is larger which results in delay in the processing and more consumption of memory. Next for the Naïve bayes algorithm is that it supposes that all features are separated or not connected. So, we cannot know the relationship between features. This algorithm faces a challenge 'zero-frequency trouble', (i.e. if the categorical variable has a category in the testing dataset, but not observed in the training dataset, then the pattern assigns a 0 probability and will not able to make a prediction). Next for Neural networks there is a neural tree constructed which sets itself for each input and changes each time. It doesn't perform well when we have large data set because the required training time is higher. It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping. SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of the Python scikit-learn library.

4.2 Data Collection and Performance Metrics

4.2.1 Data Collection

The datasets used in the experiment are collected from MIB projects. It consists of Genuine and Fake Twitter datasets. The Genuine accounts dataset contains accounts of people who came forward to be part of academic study for detecting fake accounts on Twitter and it is mostly a mixture of accounts of researchers, social experts and journalists from Italy, US and other European countries. The fake accounts were purchased from three different Twitter online markets namely fastfollowerz.com, intertwitter.com and twittertechnology.com

4.2.2 Evaluation Metrics

In order to evaluate the performance of the system, various evaluation metrics are used based on following four standard indicators

- True Positive (TP): True positives are records that are correctly detected with expected vectors.
- True Negative (TN): True negatives are records correctly detected expected as Neutral.
- False Positive (FP): False positives are records that were detected by the system as expected but are listed in the other vectors.
- False Negative (FN): False negatives are records not detected by the system.

The evaluation metrics considered are

- Accuracy which gives the ratio of number of correct results to the total number of inputs
- Precision which gives the proportion of positive detection that was actually correct
- Recall which gives the proportion of actual positives that was detected correctly
- F1 Score which takes into account both precision and recall to compute the score. F1-score is given by harmonic mean of precision and recall. If F1-score is 1, then it is best value and worst is 0.

Machine Learning Algorithm	TN %	TP %	FN %	FP %	Precision %	Recall %	F-Measure %
Random forest	94.69	94.20	17.45	3.76	96.16	71.04	81.71
Decision Tree	82.57	88.90	33.64	5.71	93.96	67.04	78.25
Naïve Bayes	79.24	81.01	57.95	7.19	91.85	61.09	73.38
Neural Network	78.17	89.33	32.57	7.54	92.21	67.36	77.85
SVM	55.64	96.58	10.45	14.92	86.62	72.83	79.13

Table 3.2.2.1 Evaluation Metrics

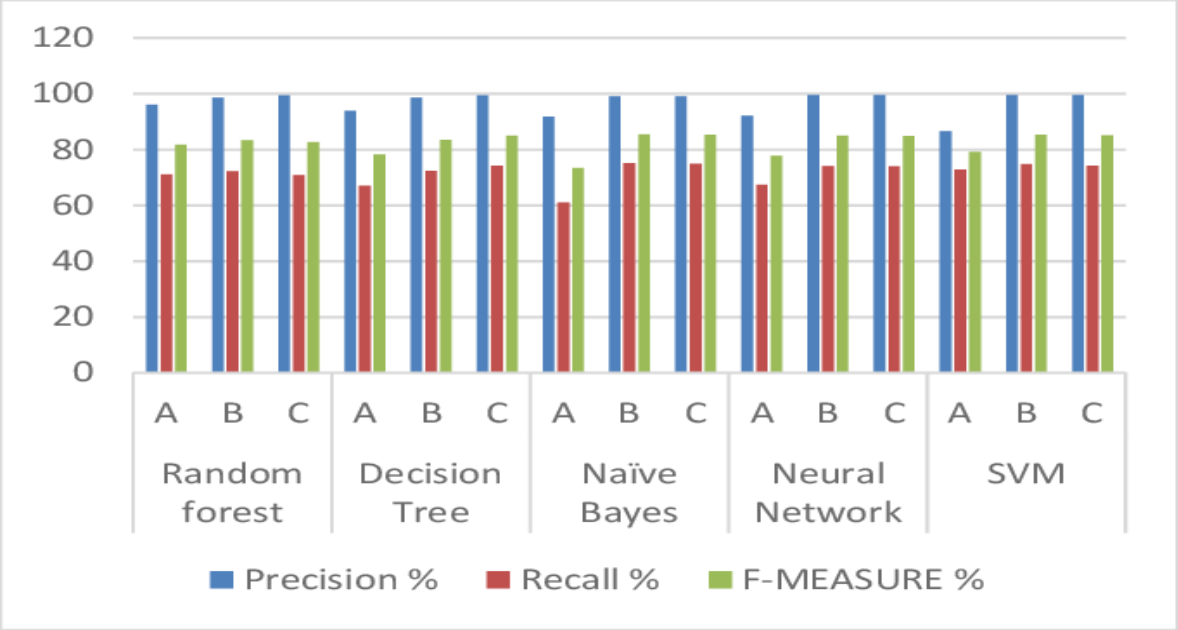


Figure 3.2.2.2 Graph for precision recall f-measure of different algorithms

CHAPTER 5
CONCLUSION

CHAPTER 5

5.1 CONCLUSION

On comparison with the existing solutions, first the bayes probability the performance metrics indicates the performance as low, also missing of a particular attribute indicates its absence which affects the whole classification. Next for SVM(Support Vector Machine) the training time is higher and there arises a problem with large data sets the accuracy is better in comparison to the bayes probability and also there is a problem with noise values overlapping this resulting in failure. Next if we include the random forest this is the most accurate classification of all but the problem is with the training it takes more time to train and consumes a lot of toll on the memory as the classification tree is larger. A huge loss on memory storage and a lot of preparation time. We propose the use of decision tree-based classification algorithms and similarity measure algorithms such as levenshtein distance and cosine similarity to address this problem. These algorithms can be effective in identifying patterns and features that distinguish between genuine and fake accounts. Ultimately, the development of effective detection techniques can help to mitigate the impact of fake and clone accounts on social media and improve the credibility of online information.

5.2 FUTURE ENHANCEMENT

Though our project is successful in detecting the fake and clone accounts on twitter, we would like to enhance it in the future in order to detect the fake and clone accounts on other social media platforms and developing real-time monitoring and detection systems that can continuously analyze Twitter data and detect fake and clone accounts in real-time can be a valuable enhancement. This can involve streaming data processing, real-time feature extraction, and adaptive model updating to keep up with the evolving nature of fake and clone accounts.

CHAPTER 6
REFERENCES

CHAPTER 6
REFERENCES

[1] Sowmya P and Madhumita Chatterjee ," Detection of Fake and Cloned Profiles in Online Social Networks", Proceedings 2019: Conference on Technologies for Future Cities (CTFC)

[2] Georgios Kontaxis, Iasonas Polakis, Sotiris Ioannidis and Evangelos P. Markatos, "Detecting Social Network Profile Cloning", 2013

[3] Piotr Brodka, Mateusz Sobas and Henric Johnson, "Profile Cloning Detection in Social Networks", 2014 European Network Intelligence Conference

[4] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angello Spognardi, Maurizio Tesconi, "Fame for sale: Efficient detection of fake Twitter followers", 2015 Elsevier's journal Decision Support Systems,Volume 80

[5] Ahmed El Azab, Amira M Idrees, Mahmoud A Mahmoud, Hesham Hefny, "Fake Account Detection in Twitter Based on Minimum Weighted Feature set", World Academy of Science, Engineering and Technology, International Journal of Computer and Information Engineering Vol:10, 2016

[6] M.A.Devmane and N.K.Rana, "Detection and Prevention of Profile Cloning in Online Social Networks", 2014 IEEE International Conference on Recent Advances and Innovations in Engineering

[7] Kiruthiga. S, Kola Sujatha. P and Kannan. A, "Detecting Cloning Attack in Social Networks Using Classification and Clustering Techniques" 2014 International Conference on Recent Trends in Information Technology

[8] Buket Erşahin, Ozlem Aktaş, Deniz Kilinc, Ceyhun Akyol, "Twitter fake account detection", 2017 International Conference on Computer Science and Engineering (UBMK)

[9] Arpitha D, Shrilakshmi Prasad, Prakruthi S, Raghuram A.S, "Python based Machine Learning for Profile Matching", International Research Journal of Engineering and Technology (IRJET), 2018

[10] Olga Peled, Michael Fire, Lior Rokach, Yuval Elovici, "Entity Matching in Online Social Networks", 2013 International Conference on Social Computing

[11] Aditi Gupta and Rishabh Kaushal, "Towards Detecting Fake User Accounts in Facebook", 2017 ISEA Asia Security and Privacy (ISEASP)

[12] Michael Fire, Roy Goldschmidt, Yuval Elovici, "Online Social Networks: Threats and Solutions", JOURNAL OF LATEX CLASS FILES, VOL. 11, NO. 4, DECEMBER 2012, IEEE Communications Surveys & Tutorials

[13] Ashraf Khalil, Hassan Hajjdiab and Nabeel Al-Qirim, "Detecting Fake Followers in Twitter: A Machine Learning Approach" 2017 International Journal of Machine Learning and Computing

[14] Mohammad Reza Khayyambashi and Fatemeh Salehi Rizi, "An approach for detecting profile cloning in online social networks" 2013 International Conference on e-Commerce in Developing Countries: with focus on e-Security

[15] Mauro Conti, Radha Poovendran and Marco Secchiero, "FakeBook: Detecting Fake Profiles in On-line Social Networks", 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

GitHub Link

https://github.com/gpranaykumar/B56_Major_Project