

I. INTRODUCTION

Online social networks have become an integral part of everyone's social life in the contemporary generation. Adding new friends and keeping up with their updates has been easier. Online social networks have an impact on many domains, such as business, education, employment, community activity, and research. Employers search and employ competent candidates who are passionate about their profession using these social networking platforms. Propaganda disseminated through social media is another issue. Conflicts may arise from false accounts that engage in the dissemination of inappropriate and false information. These fictitious accounts are also made to gain followers. False profiles cause more harm than other online crimes to people. Therefore, being able to spot a phony profile is essential.

Most fraudulent profiles are made to increase their following, engaging in phishing and spamming activities. False accounts are equipped with everything needed to commit online crimes. Fake accounts offer serious threats, including identity theft and data breaches. When consumers click on the URLs provided by these fictitious accounts, all user data is sent to distant servers where it can be used against them. In addition to damaging an account's reputation, fraudulent accounts that seem to be from companies or individuals can also cause them to lose followers and likes. Social networking issues include trolling, abuse, and infringement of personal privacy on online platforms, among many other issues.

1. EXISTING SYSTEM

The procedures that need to be done to identify fake profiles are outlined in the suggested framework below. Feedback from the classification algorithm's results, which can also be seen in the system model diagram (Figure 1), drives active learning. The following are the steps in the procedure:

- Data collection and cleaning are the first steps.
- Cross-validation is used to ensure that the model selected is ideal for the data.
- The selected model is then trained using the data set.
- Pipelining is then done to improve accuracy.
- Finally, the model is assessed using a test data set

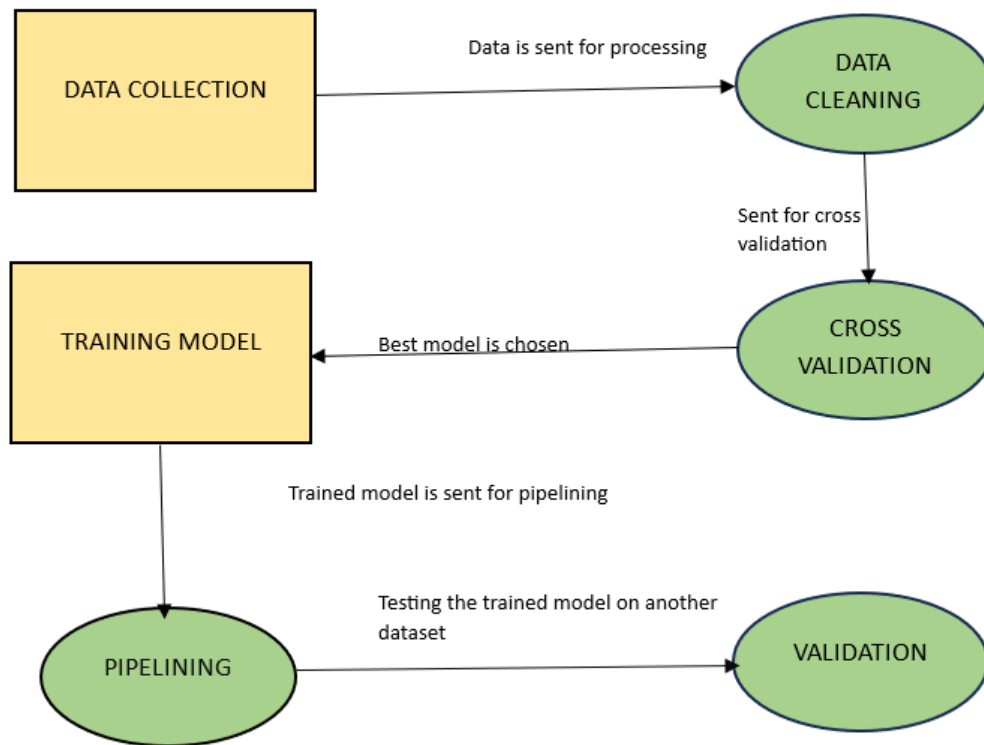


Fig. 1. Existing System Model

1.1 DATA SET

The data set has been extracted from two primary sources. The first one is from Kaggle and the other one is from Git Hub. Both platforms are open-source and save data sets for public use. We have combined similar characteristics from both datasets, such as user_name, following and followers_count, and so forth, that are present on any social media platform. This was done to investigate whether more data would lead to a forecast with a higher accuracy. The 200-row entries in the data set are manually added, and it is then utilized in the procedure. The data sets that were employed were those that were already there and the ones that were manually joined with them. This allows us to make predictions using the data set and determine whether larger amounts are appropriate in this particular situation. We require a data set that includes both fictitious and real profiles. There are two subsets of the dataset: training and testing. The testing data set is used to verify the effectiveness of the model generated by the classification algorithm, which employs the training data for training. To assess the model's accuracy after being trained on a larger volume of data, two data sets are selected. There are 556 entries in the

first data set (data set-1), and 776 entries in the second data set (data set-2). The second data set is created by manually combining 200 entries from one data set with the first data set. Attributes taken from the profiles that are considered in the process of training for the identification of fake profile identification are as follows [Figure 2]:

ATTRIBUTES	DESCRIPTION
Profile Picture	User has a profile picture or not
Full name words	Number of words in tokens
Bio/Description length	Description length in characters
External URL	Has external URL or not
Private	Private account or not
Posts	Number of posts
Followers	Number of followers
Follows	Number of follows

Table. 1. List of attributes and description

1. CROSS-VALIDATION

In the process of cross-validation, numerous training algorithms are applied to the provided data and subsequently tested on the same set of data. Next, the mean scores are shown; greater numbers indicate a stronger ability to produce better results using the data. Because every data set is unique, various optimal techniques for training can be identified by cross-validation. It's a really easy, yet effective, technique to maximize the utilization of the current model.

2. CLASSIFICATION ALGORITHMS

This project involves a wide range of subjects and algorithms. This chapter provides a summary of these subjects so that you may better grasp the steps involved in implementation. In machine learning, there are generally two categories: regression and classification. These are employed based on the data collection and the kind of result the user needs.

When the data set is constrained and predetermined, classification is used. It is employed when a true or false, yes or no format is needed for the output. Regression, on the other hand, is frequently employed in weather forecasts and is utilized when the data is continuous. This project makes use of four machine learning algorithms: Gradient Boosting Classifier, Random Forest Classifier, Logistic Regression, and Gaussian Naive Bayes.

Random Forest: This supervised ensemble learning method builds a large number of decision trees in the training stage, and then selects the best decision trees for prediction using a mean voting mechanism. The data is divided at random into several data samples, which are then divided into training and testing data trees. The voting system in between the trees then determines the prediction score.

Gradient Boosting: This classification approach combines many decision trees to create an additive predictive model. It is comparable to the random forest model, but it is based on the hunch that the model that produces the fewest errors when paired with the preceding one is the optimal one to use next. To construct a powerful prediction model, numerous weak learning models are combined.

Logistic Regression: This method makes use of both probability theory and a predictive analysis tool. This statistical technique is used to analyze data in situations where one or more independent factors control the outcome. The parameters chosen via evaluation-based logistic regression raise the possibility of discovering the case values. It produces the formula coefficients necessary to forecast a justifiable change in the duty of the aspect of interest's actuality.

Gaussian Naïve Bayes: Moreover, Gaussian Naive Bayes takes advantage of supervised machine learning. Additionally, it is a specific application of the Naive Bayes approach in which the attributes have continuous values. This approach makes the assumption that every feature has a Gaussian distribution, commonly known as a normal distribution. Using just the mean and standard deviation, this model fits.

3. CHOSEN MODEL - RANDOM FOREST

In this instance, we used the Random Forest Classifier that we had selected during the cross-validation procedure. It is a supervised learning algorithm that can handle regression as well as classification. Regression is used when there is a need for prediction for continuous data, such as in the stock market, while classification is used when prediction is required for a specific set amount of data. A method called Random Forest creates a forest by using voting to choose the best decision trees. In order to create several decision trees, it splits the data set into subsets. Therefore, the outcome is more accurate the larger the data set. This is how the Random Forest Classifier works:

- Random data samples are selected from the data set.
- Decision trees are constructed for each sample, and the predicted result is then obtained from each tree.
- Voting is conducted for each result.
- The most popular projected result is selected as the final prediction result.

4. PIPELINING PROCESS

Pipelining is iterative since each step is carried out again to enhance the model's accuracy and produce an efficient method. In order to compare and analyze data that have similar characteristics or are in a linear sequence of data transformations that may be chained together to create a modeling process that can be assessed, a pipeline is used to the model. This divides the data into independent, reusable components, which are then merged once more to create a model.

It functions by combining the data into a model that can be tested and assessed to produce a result that raises the model's efficiency. It guarantees the reusable nature of the data utilized in the preparations. It streamlines the process of creating a machine-learning model that is optimally accurate and precise while yet being efficient.

5. EVALUATION MATRICES

This study employed a variety of graphing approaches, including confusion matrices and correlation, to improve understanding of the topic. When it's necessary to verify the attributes' dependability, the correlation graph is utilized.

A plotting matrix that provides an easier-to-understand visual representation of an output is called a confusion matrix.

1.2 PROPOSED SYSTEM

The procedures that need to be followed to identify fraudulent profiles are outlined in the suggested framework below:

1. Data gathering
2. Preparing Data
3. Identification of Outliers
4. Choosing Features
5. Dividing the dataset into test and training sets
6. Use the best machine-learning algorithm available
7. False profile identification

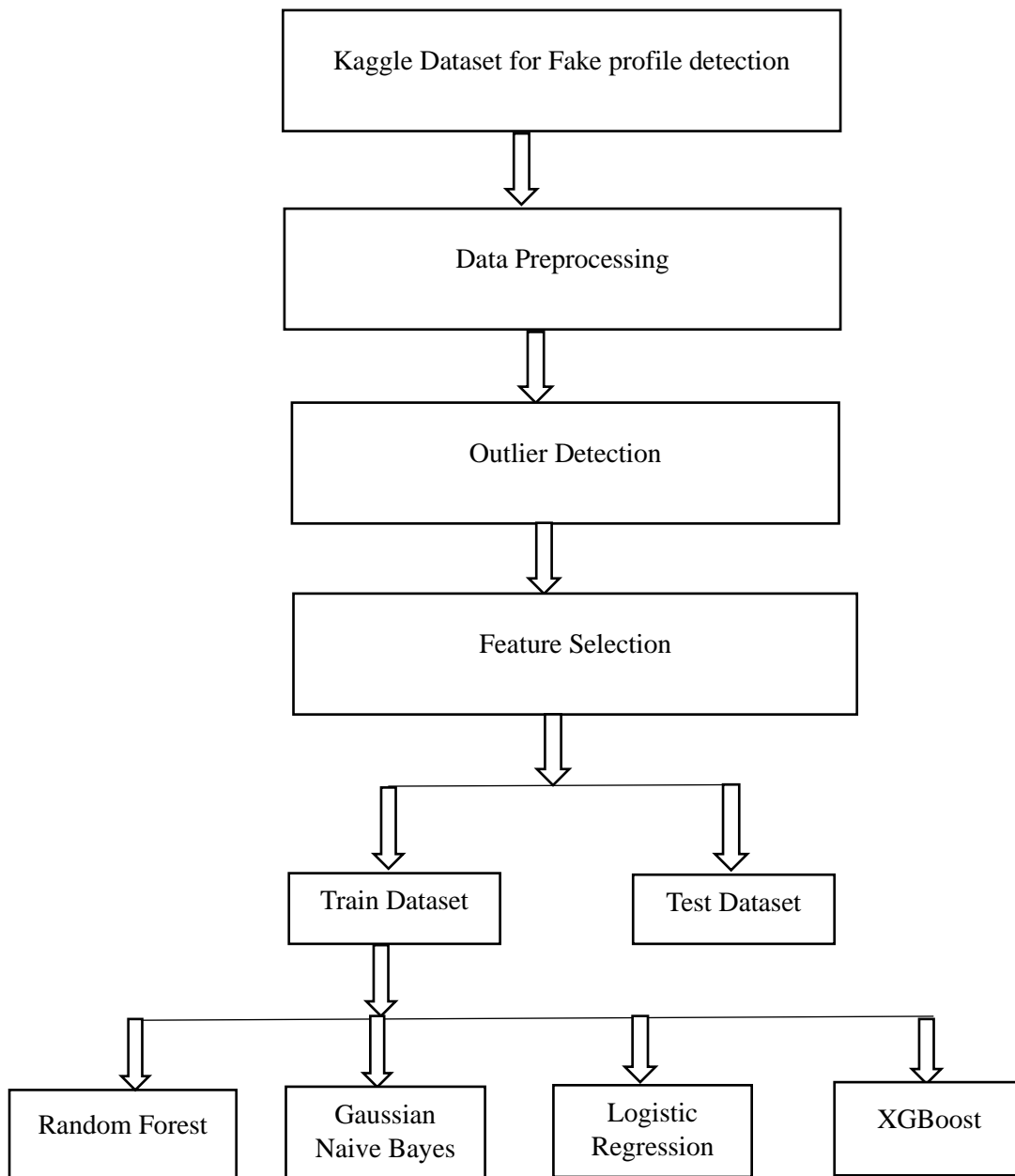
Following data collection, the data must be cleaned to improve accuracy; this includes removing superfluous columns and missing values, among other things. We call this procedure "data cleaning." Data cleansing is done as part of the preparation stage of the process. In order to clean up our dataset, we eliminated any unnecessary columns and replaced any empty values with zeros. The target column "isFake" was also added.

Once the preprocessing of the data is finished, the IQR (Interquartile Range) is used to identify and eliminate outliers.

The next step is feature selection, which essentially entails deleting any unnecessary and undesirable attributes from the dataset.

Following the dataset's division into training and testing sets, we must select the machine learning algorithms that yield the best accuracy and are most appropriate for our dataset.

Ultimately, based on the supplied user profile URL, the trained machine learning model returns a response indicating if the data is bogus or not.



SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

- Processor : intel core i7-7500UCPU@2.70gh
- Cache memory : 4 megabyte(MB)
- RAM : 8 gigabyte(GB)

SOFTWARE REQUIREMENTS:

- Operating System : Windows 10 64-bit Operating System
- Coding Language : Python
- Software Installations : Jupyter Notebook

II. LITERATURE SURVEY

MACHINE LEARNING

Machine learning is one use of artificial intelligence (AI) that enables computers to learn on their own and improve via experience instead of requiring explicit programming. Making computer programs that can access data and use self-learning techniques is its primary objective. The main objective is to enable computers to learn on their own without human guidance and modify their operations accordingly.

SOME MACHINE LEARNING METHODS

Supervised and unsupervised machine learning techniques are frequently used classification schemes.

When the system has received enough training, it may provide targets for any fresh input. Using labeled examples, supervised machine learning algorithms may predict future events by applying historical data to fresh data. To forecast the output values, the learning algorithm first analyzes a known training dataset and generates an inferred function. Additionally, it may detect faults and make necessary adjustments to the model by comparing its output with the intended, accurate result.

On the other hand, in situations when the training data is neither categorized nor labeled, unsupervised machine learning techniques are employed. Unsupervised learning investigates

how unlabeled data can be used by systems to infer a function that describes a hidden structure. Although the system is unable to determine the correct output, it is capable of analyzing the data and deriving conclusions from datasets in order to characterize hidden structures in the absence of labels.

Because semi-supervised machine learning algorithms use both labelled and unlabelled data for training—typically a small quantity of labelled data and a big amount of unlabelled data—they fall somewhere in between supervised and unsupervised learning. Systems that employ this technique can significantly increase the accuracy of their learning. Semi-supervised learning is typically used when training or learning from the labelled data requires specialized and pertinent resources. Otherwise, more resources are typically not needed to obtain unlabeled data.

Reinforcement machine learning algorithms are a type of learning that responds to its surroundings by generating actions and identifying mistakes or rewards. Reward delay and trial-and-error search are the most essential aspects of reinforcement learning. With the use of this technique, machines and software agents may automatically ascertain the best course of action to take in a given situation to optimize performance. To determine the optimal course of action, the agent must just receive basic reward feedback. This signal is referred to as reinforcement.

APPLICATIONS OF MACHINE LEARNING

1. Virtual Personal Assistants
2. Predictions while Commuting
3. Videos Surveillance
4. Social Media Services
5. Email Spam and Malware Filtering
6. Online Customer Support
7. Search Engine Result Refining
8. Product Recommendations
9. Online Fraud Detection

PREVALENCE OF FAKE PROFILE FRAUDS ON SOCIAL MEDIA

Since social media first emerged, there has been a noticeable rise in the number of fraudulent operations involving fictitious personas. Let's look at some shocking figures:

Social Media as a Haven for Scammers:

More than one in four individuals who detailed losing cash to extortion in 2021 showed that it has begun on social media. These tricks frequently start with an advertisement, a post, or a message.

In 2021, over 40% of those who reported falling victim to financial fraud said the scam started on social media. These frauds frequently started with a message, post, or advertisement

More than 95,000 people reported losing around \$770 million in 2021 as a result of fraud committed via social media. These losses represented a startling eighteen-fold rise over 2017 and made up over 25% of all recorded fraud losses in that year.

The Reasons Social Media Is So Popular With Scammers:

1.Low-expense Reach: Scammers may reach billions of individuals worldwide at a minimal expense by using social media.

2.False Personas: To trick users, scammers can quickly fabricate new profiles or take over already-existing ones.

3.Personalized Targeting: Scammers can adjust their strategy and methodically target people depending on attributes like age, interests, or previous purchases by examining personal information posted on social media.

4.Investment Scams: Investment scams, particularly those involving fictitious cryptocurrency investments, are very common on social media. In 2021, almost 50% of victims of investment scams stated that the fraud originated on social media.

5.Romance scams: Romance scams are the second most lucrative type of fraud on social media, after financial scams. These frauds frequently start out innocently with a friend request, then progress to polite conversation before demanding money.

Online shopping scams:

In terms of money lost, romance and investment scams rank highest, but the majority of reports originate from victims who were duped into purchasing goods they saw advertised on

social media. Online shopping accounted for 45% of losses due to social media scams that were recorded in 2021.

EXECUTION OF MACHINE LEARNING UTILIZING PYTHON

Python could be a well-known programming dialect. It was made in 1991 by Guido van Rossum. It is utilized for:

1. web improvement (server-side)
2. program advancement
3. arithmetic
4. framework scripting.

Python can be written in an IDE (Integrated Development Environment), such as Thonny, Pycharm, Netbeans, Eclipse, or Anaconda. These tools are especially helpful for organizing and maintaining bigger sets of Python files. Python's readability was a design feature. Python finishes a command with a new line, unlike other programming languages that frequently utilize parentheses or semicolons. Python defines scope—such as the scope of loops, functions, and classes—by indentation with whitespace. Curly brackets are frequently used in other computer languages for this reason. In the past, machine learning jobs were manually completed by coding all algorithms and statistical and mathematical formulas. This resulted in a laborious, time-consuming, and ineffective approach. However, because to numerous Python libraries, frameworks, and modules, it is now lot easier and more efficient than it was in the past. Because of its extensive library, Python is currently one of the most widely used programming languages for this kind of work, replacing a number of other languages used in the business. The following Python libraries are used in machine learning:

1. Numpy
2. Scikit-learn
3. Pandas
4. Matplotlib

NumPy is a popular Python toolkit for working with large multi-dimensional arrays and matrices by employing a number of complex mathematical operations. It is quite beneficial for elementary scientific computations in machine learning. It is particularly useful for

functions involving the Fourier transform, random numbers, and linear algebra. Better frameworks like TensorFlow internally use NumPy to work with Tensors.

Fans of machine learning love SciPy because it has a ton of modules for statistics, linear algebra, integration, and optimization. There is a distinction between the SciPy library and the SciPy stack. SciPy is one of the core packages that make up the SciPy stack. SciPy is an excellent tool for picture manipulation as well. Scikit-learn is one of the most popular machine-learning libraries for conventional machine-learning methods. It is based on two essential Python libraries, NumPy and SciPy. Scikit-learn supports almost all supervised and unsupervised learning algorithms. Since Scikit-learn can be used for data mining and analysis, it's a great tool for anyone who is just starting out with machine learning. Pandas is a popular Python package.

Pandas is a well-known Python data analysis toolkit. It has nothing to do with machine learning specifically. Since the dataset needs to be ready before training, we already know this. Pandas are useful in this situation because it was created especially for extracting and preparing data. It offers a large range of data analysis capabilities as well as high-level data structures. Numerous built-in techniques for sifting, merging, and groping data are available.

A well-liked Python data visualization package is called Matplotlib. It has nothing to do with machine learning, much like pandas. It is very useful for programmers who wish to see the patterns in the data. This is a library for 2D charting that is used to create 2D graphs and plots. Programmers can easily plot data thanks to a module called Pyplot, which offers capabilities like formatting axes, font attributes, and line styles to manipulate. It offers a variety of plots and graphs for data visualization, error charts, bar charts, histograms, and other purposes.

III. SYSTEM ANALYSIS

3.1 SCOPE OF THE PROJECT

This project mainly focuses on detecting fake profiles, thereby, helping to prevent fraud by fake users and other user privacy intrusion breaches. The scope of this project is that the user can find whether a particular user profile is fake or not while using social media platforms and interacting with unknown users online.

3.2 ANALYSIS

The datasets used in this project are the Kaggle Fake profile detection datasets. Each of these datasets contains 35 attributes that are used to predict whether a given user profile is fake or not, such as name, followers_count, etc.

0	id	2818	non-null	int64
1	name	2818	non-null	object
2	screen_name	2818	non-null	object
3	fav_number	2818	non-null	int64
4	statuses_count	2818	non-null	int64
5	followers_count	2818	non-null	int64
6	friends_count	2818	non-null	int64
7	favourites_count	2818	non-null	int64
8	listed_count	2818	non-null	int64
9	created_at	2818	non-null	object
10	url	463	non-null	object
11	lang	2818	non-null	object
12	time_zone	1069	non-null	object
13	location	2271	non-null	object
14	default_profile	1728	non-null	float64
15	default_profile_image	8	non-null	float64
16	geo_enabled	721	non-null	float64
17	profile_image_url	2818	non-null	object
18	profile_banner_url	987	non-null	object
19	profile_use_background_image	2760	non-null	float64
20	profile_background_image_url_https	2818	non-null	object
21	profile_text_color	2818	non-null	object
22	profile_image_url_https	2818	non-null	object
23	profile_sidebar_border_color	2818	non-null	object
24	profile_background_tile	489	non-null	float64
25	profile_sidebar_fill_color	2818	non-null	object
26	profile_background_image_url	2818	non-null	object
27	profile_background_color	2818	non-null	object
28	profile_link_color	2818	non-null	object
29	utc_offset	1069	non-null	float64
30	protected	0	non-null	float64
31	verified	0	non-null	float64
32	description	2547	non-null	object
33	updated	2818	non-null	object
34	dataset	2818	non-null	object
35	isFake	2818	non-null	float64

Fig.4. Dataset

3.3 DATA PRE-PROCESSING

The act of making adjustments to our data prior to providing it to an algorithm is known as pre-processing. Pre-processing turns unclean, unprocessed data into something unfit for analysis. Better results from machine learning models depend on the data being in the correct format. Different algorithms require the data to be in a specific format. For example, the Random Forest algorithm does not support null values. Raw data must be used to manage those null values. Pre-processing is the set of adjustments we make to our data before sending it to the algorithm. Details Pre-processing is one technique used to turn raw data into clean data.

3.3.1 MISSING VALUES

Completing the missing values is one of the pre-processing techniques. For pandas to recognize the absence values, they must be transformed to a standard missing value, NaN. We have filled in the blanks using the feature median. After the missing numbers are filled in, our heat map looks like this in the figure. Handling missing values is important since machine learning algorithms do not tolerate missing values in the data.

Among the strategies for managing lost values are:

1. Swap out for a cruel, middle, or mode
2. Back-fill or forward-fill
3. Erasing push

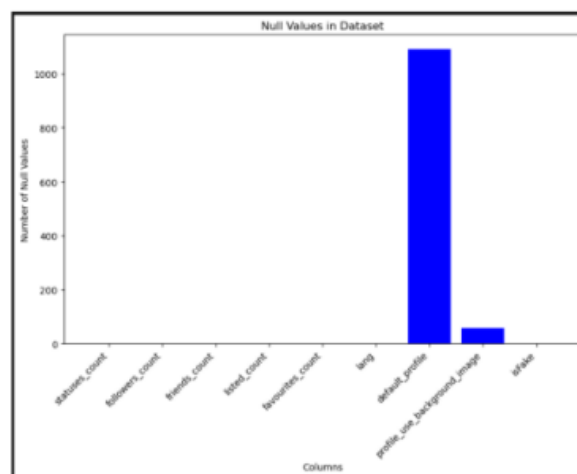


Fig.5. Dataset before filling missing values

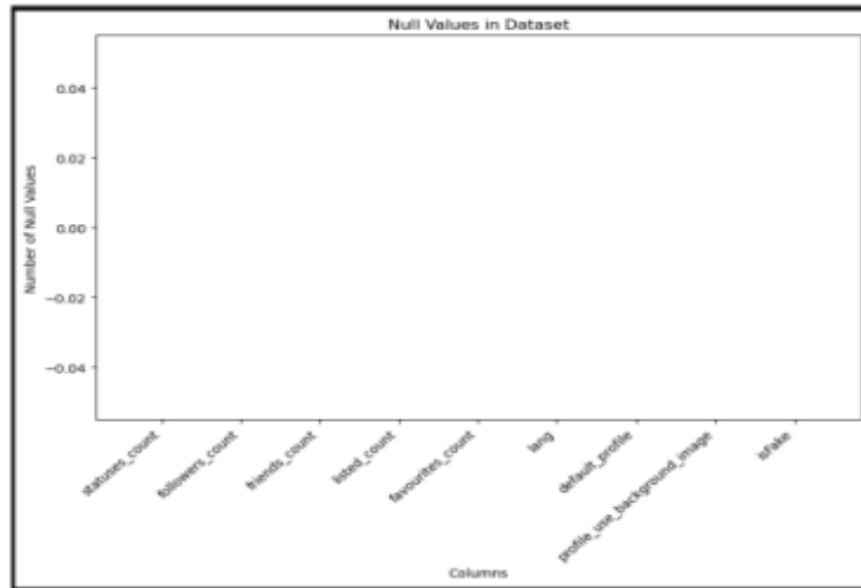


Fig.6. Dataset after filling missing values

3.5 FEATURE SELECTION

When creating a predictive model, feature selection is the process of minimizing the number of input variables. Reducing the number of input variables is desirable in order to lower the computational cost of modeling and, in certain situations, to increase or improve the model's performance. Correlation is the feature selection strategy employed in this project.

3.5.1 CORRELATION

Correlation describes the linear relationship between the two continuous variables. Correlation is utilized when a response variable cannot be determined. It's a statistical measure that illustrates the simultaneous fluctuations of two or more variables. The amount that those variables rise or fall at the same time is indicated by a positive correlation. A negative correlation shows how much one variable rises while the other lowers.

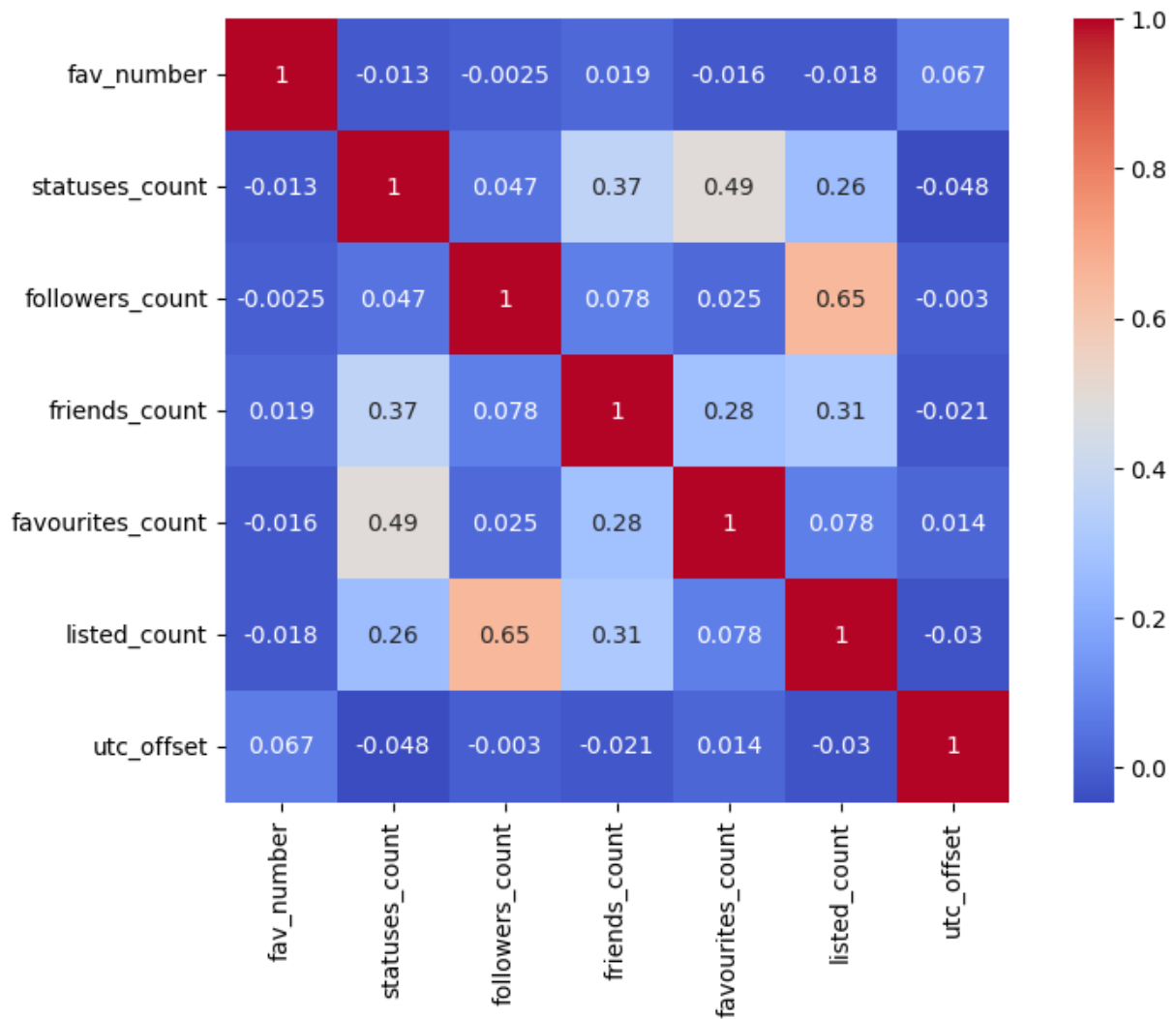


Fig.7. Correlation for fake profile detection dataset

3.6 CLASSIFICATION

The goal of the supervised machine learning method for classification is for the model to predict an input set of data's proper label. In classification, the model is completely trained with the training set and evaluated with test data before being used to generate predictions on new, unseen data. It guesses the class of supplied data points and divides a batch of data into classes. Classes are also referred to as labels, goals, or categories. Mapping function (f) from input variables (X) to discrete output variables (y) is the task of classification predictive modeling.

REGRESSION VS. MACHINE LEARNING CLASSIFICATION

Algorithms for machine learning fall into four primary categories: semi-supervised, supervised, unsupervised, and reinforcement learning. While not the same, classification and

regression are both included in the category of supervised learning. The prediction job is a classification when the target variable is discrete. An application is the process of identifying the underlying sentiment of a text. Regression is the prediction issue if the target variable is continuous. One example would be to project an individual's salary depending on their seniority, location, previous employment history, and educational background.

VARIOUS MACHINE LEARNING CLASSIFICATION TASK TYPES

Machine learning involves four primary classification tasks: binary, multi-class, multi-label, and unbalanced classifications.

BINARY CLASSIFICATION

A binary classification task is to sort the input data into two groups that are mutually exclusive. Here, the problem being solved determines the binary format in which the training data is labeled: true or false, positive or negative, 0 or 1, spam or not spam, etc.

Vector machine support and logistic regression algorithms are designed with binary classifications in mind.

MULTIPLE CLASSIFICATION

Unlike binary classification, which has at least two mutually distinct class labels, multi-class classification aims to predict which class a given input example belongs to. Most binary classification methods can be used to achieve multi-class classification

Among these algorithms are, but are not restricted to:

1. K-Nearest Neighbours
2. Random Forest
3. Naive Bayes
4. Gradient Boosting
5. SVM
6. Logistic Regression.

It is to be noted that multi-class classification is not supported by default by SVM and Logistic Regression.

LOGISTIC REGRESSION

The right kind of regression analysis to perform when the dependant variable is dichotomous (binary) is logistic regression. Logistic regression is a type of regression analysis, much like any other. It is employed to characterize data and clarify the connection between one or more nominal, ordinal, interval, or ratio-level independent variables and one dependent binary variable.

It simply refers to a variable with only two possible outcomes, such as whether or not a person will survive this accident and whether or not a student will pass this exam. There are two possible outcomes: yes and no.

TYPES OF LOGISTIC REGRESSION

The following are the three main types of logistic regression:

BINARY LOGISTIC REGRESSION

To predict the likelihood of a binary outcome, such as true or false, yes or no, or 0 or 1, binary logistic regression is utilized. It could be used to forecast, for instance, if a patient has a sickness or not, whether a loan will be repaid, or whether a client will churn.

LOGISTIC MULTINOMIAL REGRESSION

Logistic regression is the appropriate type of regression analysis to use when the dependent variable is binary or dichotomous. Like every other kind of regression analysis, logistic regression is one. It is used to describe data and make clear how one dependent binary variable and one or more independent variables at the nominal, ordinal, interval, or ratio level relate to each other. Yes or no are the two possible results.

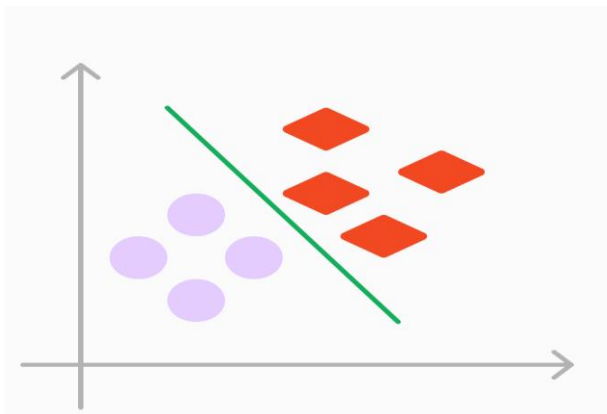


Fig. 8. Binary classification

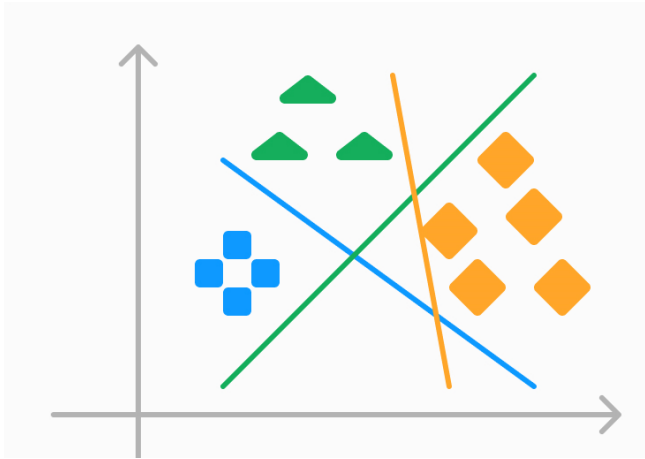


Fig. 9. Multiple regression

THE LOGISTIC FUNCTION

Formula for Logistic Regression:

We all know the equation of the best fit line in linear regression is:

$$y = \beta_0 + \beta_1 x$$

Let's say instead of y we are taking probabilities (P). But there is an issue here, the value of (P) will exceed 1 or go below 0 and we know that range of Probability is (0-1). To overcome this issue we take "odds" of P :

$$p = \beta_0 + \beta_1 x$$

$$p/(1-p) = \beta_0 + \beta_1 x$$

The range restriction in this case is a concern since it would reduce our correlation, which is why we don't want a restricted range. We are essentially reducing the number of data points by narrowing the range, and naturally, as the number of data points drops, so does the correlation. Restricting the range of a variable makes it challenging to model. We use the log of odds, which ranges from $(-\infty, +\infty)$ to manage this.

$$\log([p/(1-p)]) = \beta_0 + \beta_1 x$$

We want to predict probability, thus all we need is a function of P, not a log of the odds. In order to accomplish this, we shall multiply both sides by the exponent and then find P.

$$\exp(\log([p/(1-p)])) = \exp(\beta_0 + \beta_1 x)$$

$$e^{\ln[p/(1-p)]} = e^{(\beta_0 + \beta_1 x)}$$

$$p = e^{(\beta_0 + \beta_1 x)} - p e^{(\beta_0 + \beta_1 x)}$$

$$p = p[(e^{(\beta_0 + \beta_1 x)} / p) - e^{(\beta_0 + \beta_1 x)}]$$

$$1 = [(e^{(\beta_0 + \beta_1 x)} / p) - e^{(\beta_0 + \beta_1 x)}]$$

$$p[1 + e^{(\beta_0 + \beta_1 x)}] = e^{(\beta_0 + \beta_1 x)}$$

$$p = e^{(\beta_0 + \beta_1 x)} / [1 + e^{(\beta_0 + \beta_1 x)}]$$

Now dividing by $e^{(\beta_0 + \beta_1 x)}$, we will get

$$p = 1 / [1 + e^{-(\beta_0 + \beta_1 x)}]$$

This is our sigmoid function. Now we have our logistic function, also called a sigmoid function. The graph of a sigmoid function is shown below. It squeezes a straight line into an S-curve.

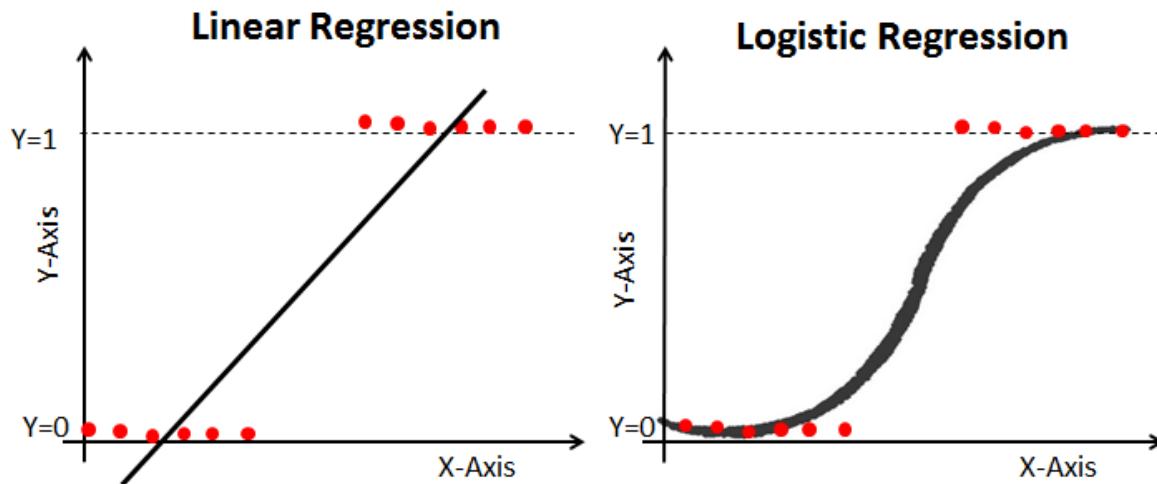


Fig.10. Comparison of Linear Regression and Logistic Regression

COST FUNCTION IN LOGISTIC REGRESSION

The difference between $y_{\text{predicted}}$ and y_{actual} , or the mean squared error, is used in linear regression and is obtained using the maximum likelihood estimator. This is how the cost function graph in linear regression looks:

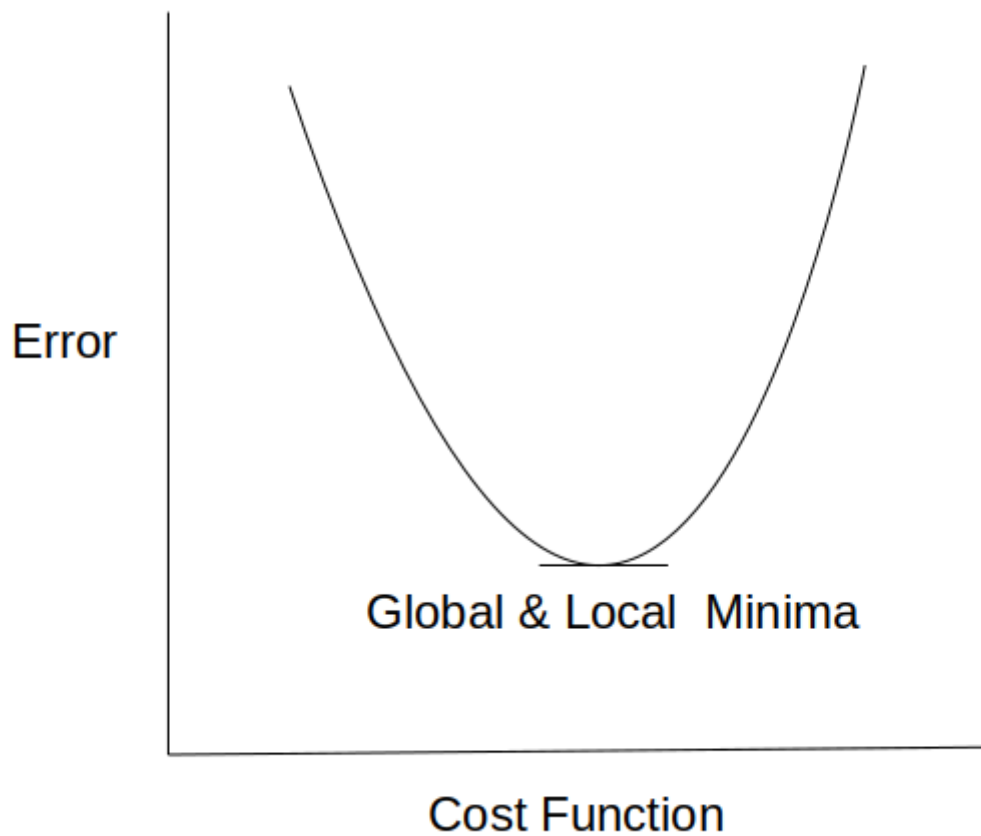


Fig.11. The cost function

Linear Regression Cost Function

$$J = \frac{\sum_{i=1}^n (\hat{Y}_i - Y_i)^2}{n}$$

In logistic regression Y_i is a non-linear function ($\hat{Y} = 1/(1 + e^{-z})$). If we use this in the above MSE equation then it will give a non-convex graph with many local minima as shown

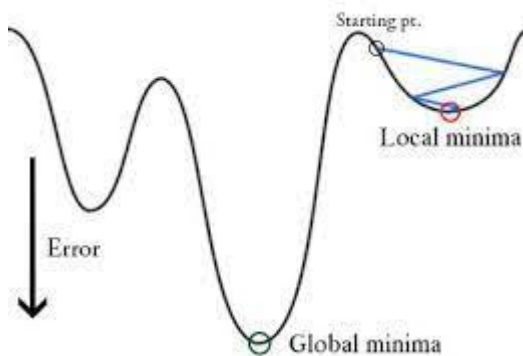


Fig.12. Global and local minima of the cost function

The difficulty with this cost function is that it will lead to results with local minima, which will increase our inaccuracy and hinder us from locating our global minima. To address this, we develop an additional logistic regression cost function, termed log loss, which is also obtained from the maximum likelihood estimation method.

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^N -(y_i * \log(\hat{Y}_i) + (1 - y_i) * \log(1 - \hat{Y}_i))$$

RANDOM FOREST CLASSIFICATION ALGORITHM

We must examine the ensemble learning method before comprehending how the random forest algorithm functions in machine learning. Ensemble is only a term for a grouping of models. Therefore, rather than using a single model to make predictions, a group of models is used.

The method's ability to handle complex datasets and reduce overfitting makes it a valuable tool for a range of machine learning prediction applications. One of the most important characteristics of the Random Forest Algorithm is its capacity to handle data sets containing both continuous variables, as in regression, and categorical variables, as in classification. It performs better in tasks that include classification and regression. In this lesson, we will understand the workings of random forests and apply them to a classification task.

WORKING OF RANDOM FOREST ALGORITHM

Prior to delving into the inner workings of the random forest algorithm in machine learning, it is necessary to examine the ensemble learning technique, which is simply defined as the combination of multiple models. This means that a collection of models is used to make predictions rather than a single model.

Ensemble uses two types of methods:

BAGGING

It uses replacement to build a distinct training subset from sample training data, and majority voting determines the final result. Consider Random Forest.

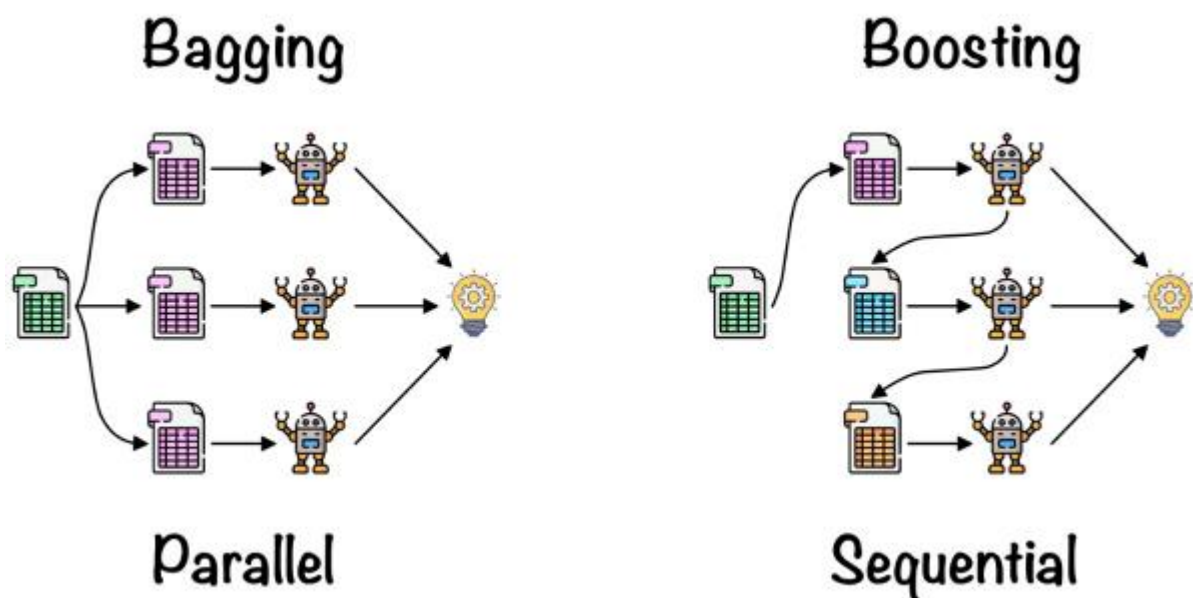


Fig.13. Bagging and Boosting Techniques

Random forest works on the Bagging principle.

In the Random Forest method, Bagging—also referred to as Bootstrap Aggregation—acts as the ensemble approach. The steps involved in bagging are as follows:

Selection of Subset: The first step in bagging is to select a subset, or random sample, from the full dataset.

Bootstrap Sampling: These samples, sometimes referred to as Bootstrap Samples, are extracted from the original data and replaced to generate each model. We call this procedure "row sampling."

Bootstrapping: Bootstrapping is the process of row sampling with replacement.

Independent Model Training: Using the appropriate Bootstrap Sample, each model is trained separately. Every model receives results from this training procedure.

Majority Voting: The outcomes of all the models are combined to produce the final output. Among the models, the most frequently anticipated result is chosen.

Aggregation: Aggregation is the process of integrating all the results and producing the final product by voting in a majority.

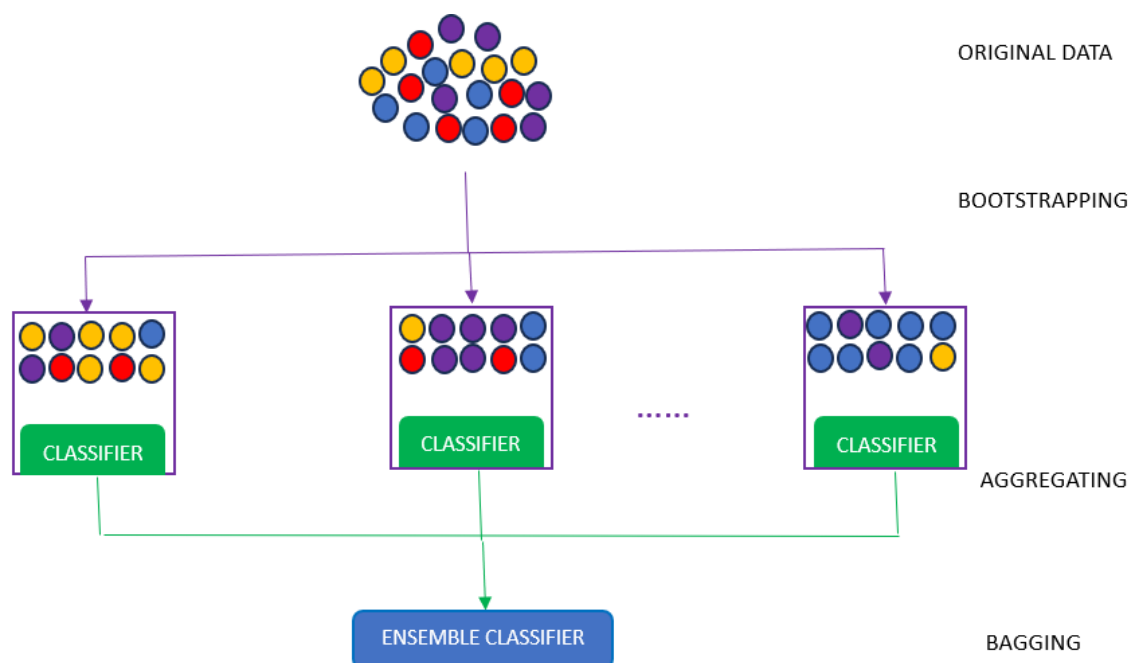


Fig. 14. Random Forest

BOOSTING

Through the creation of sequential models, it transforms poor learners into strong learners, ultimately achieving the highest accuracy. For instance, XG BOOST and ADA BOOST.

Among the methods that make advantage of the idea of ensemble learning is boosting. To produce the final result, a boosting method combines several elementary models, sometimes referred to as base estimators or weak learners. The process involves employing weak models in succession to develop a model.

STEPS INVOLVED IN RANDOM FOREST ALGORITHM

- **Step 1:** A subset of data points and a subset of characteristics are chosen for each decision tree construction in the Random Forest model. To put it simply, the data set with k records contains n random records and m characteristics.
- **Step 2:** Every sample has a unique decision tree built for it.
- **Step 3:** In the third step, every decision tree will produce an output.
- **Step 4:** For classification and regression, the final output is evaluated using either majority voting or averaging.

IMPORTANT HYPERPARAMETERS IN RANDOM FOREST

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.

HYPERPARAMETERS TO INCREASE THE PREDICTIVE POWER

- **n_estimators:** The number of trees the algorithm creates (n_estimators) prior to averaging the predictions.
- **max_features:** The most features that a random forest will take into account before splitting a node.
- **mini_sample_leaf:** Ascertains the bare minimum of leaves needed for an internal node to split.
- **criteria:** How should each tree's nodes be split(LogLoss/Entropy/GiniImpurity)
- **max_leaf_nodes:** Each tree's maximum number of leaf nodes

GAUSSIAN NAÏVE BAYES CLASSIFIER

The most straightforward and quick classification technique available is naive bayes, which works well when handling massive volumes of data. The Naive Bayes classifier has proven to be successful in a number of applications, including recommender systems, text classification, sentiment analysis, and spam filtering. It makes predictions about unknown classes using Bayesian probability theory.

The Naive Bayes machine learning model is a simple yet powerful probabilistic classification model that utilizes the Bayes Theorem.

A conditional probability of an event A occurring given that another event B has already occurred is provided by the Bayes theorem. The following is its mathematical formula: –

$$P(A|B) = P(B|A).P(A)/P(B)$$

Where

- A and B are two events
- $P(A|B)$ is the probability of event A provided event B has already happened.
- $P(B|A)$ is the probability of event B provided event A has already happened.
- $P(A)$ is the independent probability of A
- $P(B)$ is the independent probability of B

Now, this Bayes theorem can be used to generate the following classification model –

$$P(y|X) = P(X|y).P(y)/P(X)$$

Where

- $X = x_1, x_2, x_3, \dots, x_N$ are list of independent predictors
- y is the class label
- $P(y|X)$ is the probability of label y given the predictors X

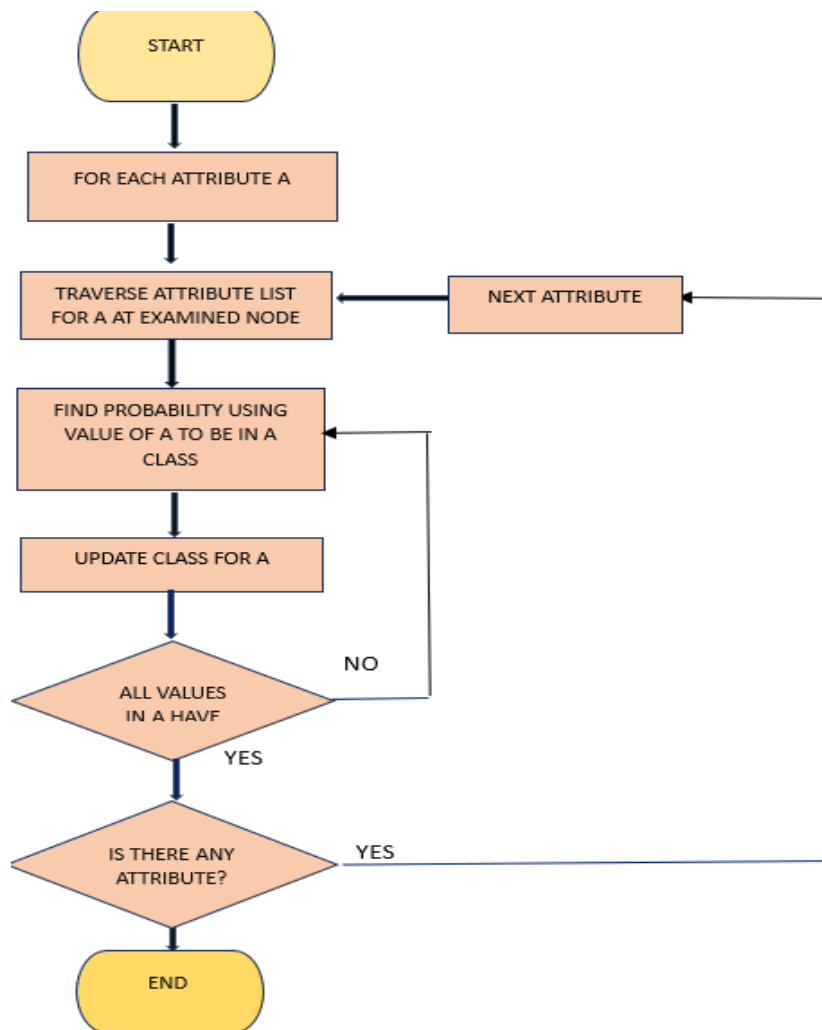


Fig.15. Flowchart depicting the Gaussian Naïve Bayes

TYPES OF NAIVE BAYES CLASSIFIERS

Naive Bayes Classifiers are classified into three categories —

i) Gaussian Naive Bayes

When the predictor values are continuous and are anticipated to follow a Gaussian distribution, this classifier is used.

ii) Bernoulli Naive Bayes

This classifier is used when the predictors are boolean in nature and are expected to follow the Bernoulli distribution.

iii) Multinomial Naive Bayes

This classifier, which uses a multinomial distribution, is frequently applied to text or document categorization problems.

XGBOOST CLASSIFICATION ALGORITHM

A machine learning method called XGBoost is a component of the ensemble learning subset gradient boosting architecture. Using decision trees as foundation learners, it applies regularization approaches to enhance model generalization. Because of its skill with feature importance analysis, missing data management, and computing economy, XGBoost is a well-liked option for computationally intensive tasks including regression, classification, and ranking.

Xtreme Gradient Boosting (XGBoost) is a machine learning technique designed for ensemble learning. It's in for supervised learning tasks such as classification and regression. XGBoost builds a predictive model by iteratively combining the predictions of many models, usually decision trees. Weak learners are progressively introduced to the ensemble so that the algorithm can work; each new learner focuses on correcting the errors made by the preceding ones. It uses gradient descent, an optimization technique, to minimize a preset loss function throughout training. The ability to handle complex relationships in data, regularization techniques to prevent overfitting, and the incorporation of parallel processing for efficient computation are the main features of the XGBoost Algorithm. The excellent prediction performance and wide applicability of XGBoost in numerous areas.

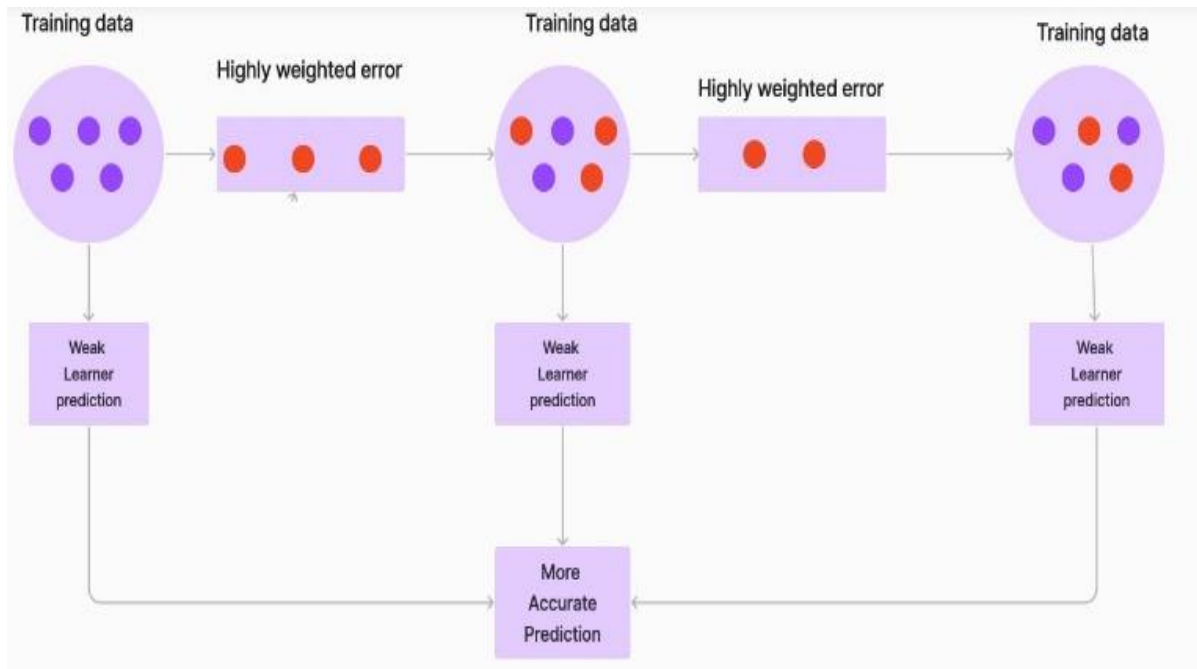


Fig.16. Gradient Boosting

The following steps are involved in gradient boosting:

- $F_0(x)$ – with which we initialize the boosting algorithm – is to be defined:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$$

- The gradient of the loss function is computed iteratively:

$$r_{im} = -\alpha \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}, \text{ where } \alpha \text{ is the learning rate}$$

- Each $h_m(x)$ is fit on the gradient obtained at each step
- The multiplicative factor γ_m for each terminal node is derived and the boosted model $F_m(x)$ is defined:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

K-FOLD CROSS VALIDATION

The entire collection of data is split into k sets of about similar sizes in this resampling procedure. The model is trained on the remaining $k-1$ sets once the first set is chosen as the test set. The model is then fitted to the test data, and the test error rate is computed.

The second set is chosen as the test set in the second iteration, while the remaining $k-1$ sets are utilized to train the data and compute the error. This procedure is carried out for every k sets.

The CV test error estimate is computed as the mean of the errors from all the iterations.

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^k MSE_i.$$

The number of folds (k) in the K-Fold CV is less than the total number of observations in the data. This approach's greatest benefit is that each data point occurs exactly once in the training set and k times in the test set. As the number of folds k increases, the variance also decreases (low variance). This approach produces intermediate bias since each training set has more observations $(k-1)n/k$ than in the Hold Out method but fewer than in the Leave One Out method.

THE CONFUSION MATRIX

A confusion matrix, which shows a classification model's accuracy, is a machine learning performance evaluation tool. The quantity of false positives, false negatives, true positives, and true negatives is shown. This matrix helps with misclassification detection, predictive accuracy improvement, and model performance analysis.

An $N \times N$ matrix, where N is the total number of target classes, is called a confusion matrix and is used to assess how well a classification model performs. The machine learning model's projected values are compared with the actual target values in the matrix. This provides us with a comprehensive understanding of the types of mistakes and performance metrics of our classification model.

For a binary classification problem, we would have a 2×2 matrix, as shown below, with 4 values:

ACTUAL VALUES			
		POSITIVE	NEGATIVE
PREDICTED VALUES	POSITIVE	TP	FP
	NEGATIVE	FN	TN

Fig.17. The Confusion Matrix

- The target variable has two values: Positive or Negative
- The columns represent the actual values of the target variable
- The rows represent the predicted values of the target variable

IMPORTANT TERMS IN A CONFUSION MATRIX

True Positive (TP)

- The predicted value matches the actual value, or the predicted class matches the actual class.
- The actual value was positive, and the model predicted a positive value.

True Negative (TN)

- The predicted value matches the actual value, or the predicted class matches the actual class.
- The actual value was negative, and the model predicted a negative value.

False Positive (FP) – Type I Error

- The predicted value was falsely predicted.
- The actual value was negative, but the model predicted a positive value.
- Also known as the type I error.

False Negative (FN) – Type II Error

- The predicted value was falsely predicted.
- The actual value was positive, but the model predicted a negative value.
- Also known as the type II error.

The confusion matrix can be used to calculate a variety of metrics, such as accuracy, precision, recall, and F1 score.

PRECISION AND RECALL

Precision tells us how many of the correctly predicted cases actually turned out to be positive.

Here's how to calculate Precision:

$$\text{PRECISION} = \text{TP} / (\text{TP} + \text{FP})$$

This would determine whether our model is reliable or not.

Recall tells us how many of the actual positive cases we were able to predict correctly with our model.

And here's how we can calculate Recall:

$$\text{RECALL} = \text{TP} / (\text{TP} + \text{FN})$$

Precision is a useful metric in cases where False Positive is a higher concern than False Negatives.

Recall is a useful metric in cases where False Negative trumps False Positive. The Confusion Matrix, particularly focusing on recall, provides a more insightful measure.

F1-SCORE

In practice, when we try to increase the precision of our model, the recall goes down, and vice-versa. The F1-score captures both the trends in a single value:

$$\text{F1-SCORE} = 2/[(1/\text{RECALL})+(1/\text{PRECISION})]$$

F1-score is a harmonic mean of Precision and Recall, and so it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.

The interpretability of the F1-score is poor. This means that we don't know what our classifier is maximizing – precision or recall. So, we use it in combination with other evaluation metrics, giving us a complete picture of the result.

IV. IMPLEMENTATION CODE

BACKEND CODE-

```
import pandas as pd
```

```
import numpy as np
```

```
import seaborn as sns
```

```
import missingno as msno
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import classification_report
```

```
from imblearn.over_sampling import SMOTE
```

```
from sklearn.metrics import roc_curve, auc
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn import datasets, metrics
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn.model_selection import KFold
```

```
from sklearn.model_selection import cross_val_score

import warnings

warnings.filterwarnings('ignore')

import pickle


user=pd.read_csv("users.csv")

fake=pd.read_csv("fusers.csv")


#adding a column for detecting fake or not

idNo_user=np.zeros(1481) #zero is adding for user

idNo_fake=np.ones(1337) #ones is adding for fake users


user["isFake"]=idNo_user

fake["isFake"]=idNo_fake


df_user=pd.concat([user,fake],ignore_index=True)

df_user.columns


#shuffle the whole data

df_user=df_user.sample(frac=1).reset_index(drop=True)
```

```
df_user.info()
```

```
user_counts = df_user['isFake'].value_counts()
```

```
# Plot the bar graph
```

```
plt.bar(['Real', 'Fake'], user_counts.values, color=['green', 'red'])
```

```
# Add labels and title
```

```
plt.xlabel('User Type')
```

```
plt.ylabel('Number of Users')
```

```
plt.title('Distribution of Real and Fake Users')
```

```
# Add labels to the bars
```

```
for i, value in enumerate(user_counts.values):
```

```
    plt.text(i, value + 10, str(value), ha='center')
```

```
# Show the plot
```

```
plt.show()
```

```
df_user=df_user[[

    'statuses_count',

    'followers_count',

    'friends_count',

    'listed_count',

    'favourites_count',

    'lang',

    'default_profile',

    'profile_use_background_image',

    'isFake'

]]
```

```
df_user.info()
```

```
null_counts = df_user.isnull().sum()
```

```
plt.figure(figsize=(10, 6))
```

```
# Plot the bar graph
```

```
plt.bar(null_counts.index, null_counts.values, color='blue')
```

```
# Add labels and title
```

```
plt.xlabel('Columns')
```

```
plt.ylabel('Number of Null Values')
```

```
plt.title('Null Values in Dataset')
```

```
# Rotate x-axis labels for better readability
```

```
plt.xticks(rotation=45, ha='right')
```

```
# Show the plot
```

```
plt.show()
```

```
df_user=df_user.fillna(0)
```

```
null_counts = df_user.isnull().sum()
```

```
plt.figure(figsize=(10, 6))
```

```
# Plot the bar graph
```

```
plt.bar(null_counts.index, null_counts.values, color='blue')
```

```
# Add labels and title
```

```
plt.xlabel('Columns')
```

```
plt.ylabel('Number of Null Values')
```

```
plt.title('Null Values in Dataset')
```

```
# Rotate x-axis labels for better readability
```

```
plt.xticks(rotation=45, ha='right')
```

```
# Show the plot
```

```
plt.show()
```

```
def plotting(col, df=df_user):
```

```
    return df.groupby(col)['isFake'].value_counts().unstack().plot(kind='bar', figsize=(10,5),  
title=f'Distribution of isFake by {col}')
```

```
# Example usage:
```

```
plotting("statuses_count")
```

```
plotting("followers_count")
```

```
plotting("friends_count")
```

```
plotting("listed_count")
```

```
plotting("favourites_count")
```

```
plotting("lang")
```

```
plotting("default_profile")
```

```
plotting("profile_use_background_image")
```

```
plt.figure(figsize=(12, 3))
```

```
# Create the boxplot
```

```
sns.boxplot(data=df_user)
```

```
# Show the plot
```



```
plt.show()
```

```
#data_df = pd.DataFrame(df_user)
```

```
Q1 = df_user.quantile(0.25)
```

```
Q3 = df_user.quantile(0.75)
```

```
IQR = Q3 - Q1
```

```
print(IQR)
```

```
df_user = df_user[~((df_user < (Q1 - 1.5 * IQR)) |(df_user > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
df_user
```

```
plt.figure(figsize=(12, 5))
```

```
# Create the boxplot
```

```
sns.boxplot(data=df_user)
```

```
# Show the plot
```

```
plt.show()
```

```
numeric_columns = df_user.drop(columns=['isFake']).select_dtypes(include=[np.number])
```

```
plt.figure(figsize=(8, 5))
```

```
# Calculate correlation matrix
```

```
correlation_matrix = numeric_columns.corr()
```

```
# Draw the heatmap with the mask and correct aspect ratio
```

```
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=True, fmt=".2f", square=True)
```

```
# Show plot
```

```
plt.show()
```

```
for col in df_user.columns:
```

```
    le=LabelEncoder()
```

```
    df_user[col]=le.fit_transform(df_user[col])
```

```
df_user.head(20)
```

```
y=df_user['isFake']
```

```
x=df_user.drop(['isFake'], axis=1)
```

```
#dataset-2
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

```
x_train
```

```
x_test
```

```
y_train
```

```
y_test
```

```
sm = SMOTE(random_state=42, k_neighbors=5)
```

```
x, y = sm.fit_resample(x, y)
```

```
a=y.value_counts()
```

```
a
```

```
#user_counts = df_user['isFake'].value_counts()
```

```
# Plot the bar graph
```

```
plt.bar(['Real', 'Fake'], a.values, color=['green', 'red'])
```

```
# Add labels and title
```

```
plt.xlabel('User Type')
```

```
plt.ylabel('Number of Users')
```

```
plt.title('Distribution of Real and Fake Users')
```

```
# Add labels to the bars
```

```
for i, value in enumerate(a.values):
```

```
    plt.text(i, value + 10, str(value), ha='center')
```

```
# Show the plot
```

```
plt.show()
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn import metrics
```

```
from sklearn.impute import SimpleImputer
```

```
# Replace NaN values with the mean
```

```
imputer = SimpleImputer(strategy='mean')
```

```
x_train_imputed = imputer.fit_transform(x_train)
```

```
from sklearn.impute import SimpleImputer
```

```
# Create the imputer with strategy='mean' or 'median'
```

```
imputer = SimpleImputer(strategy='mean')
```

```
# Fit and transform the imputer on the training data
```

```
x_train_imputed = pd.DataFrame(imputer.fit_transform(x_train), columns=x_train.columns)
```

```
# Transform the test data using the imputer fitted on the training data
```

```
x_test_imputed = pd.DataFrame(imputer.transform(x_test), columns=x_test.columns)
```

```
Training_Accuracy_L=[]
```

```
Test_Accuracy_L=[]
```

```
Sensitivity_L=[]
```

```
Specificity_L=[]
```

```
F1Score_L=[]
```

```
Precision_L=[]
```

```
Negative_Predictive_Value_L=[]
```

False_Negative_Rate_L=[]

False_Positive_Rate_L=[]

False_Discovery_Rate_L=[]

False_Omission_Rate_L=[]

average_cv_accuracy_L=[]

Training_Accuracy_L=[]

Test_Accuracy_L=[]

Sensitivity_L=[]

Specificity_L=[]

F1Score_L=[]

Precision_L=[]

Negative_Predictive_Value_L=[]

False_Negative_Rate_L=[]

False_Positive_Rate_L=[]

False_Discovery_Rate_L=[]

False_Omission_Rate_L=[]

average_cv_accuracy_L=[]

import math

```
def rounder(n):
```

```
    try:
```

```
        return math.ceil(n * 1000) / 1000
```

```
    except:
```

```
        return n
```

```
def fun(model,name):
```

```
    test_pred = model.predict(x_test)
```

```
    train_pred = model.predict(x_train)
```

```
    train_acc=rounder(accuracy_score(y_train,train_pred)*100)
```

```
    test_acc=rounder(accuracy_score(y_test,test_pred)*100)
```

```
    Training_Accuracy_L.append(train_acc)
```

```
    Test_Accuracy_L.append(test_acc)
```

```
    print("\nTraining Accuracy:", train_acc)
```

```
    print("\nTesting Accuracy:",test_acc)
```

```
print(classification_report(y_test,test_pred))
```

```
test_conf_matrix = confusion_matrix(y_test,test_pred)
```

```
plt.figure(figsize=(4, 4))
```

```
sns.heatmap(test_conf_matrix, annot=True, fmt='g', cmap='Greens', cbar=False)
```

```
t=name+' Confusion Matrix - Test Set'
```

```
plt.title(t)
```

```
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
```

```
plt.show()
```

```
tn, fp,fn,tp = test_conf_matrix.ravel()
```

```
Sensitivity=rounder((tp) / (tp + fn))
```

```
Sensitivity_L.append(Sensitivity)
```

```
Specificity=rounder((tn) / (tn + fp))
```

```
Specificity_L.append(Specificity)
```

```
F1Score=rounder( (2 * tp) / (2 * tp+ fp + fn))
```

```
F1Score_L.append(F1Score)
```



```
Precision=rounder((tp) / (tp +fp))
```

```
Precision_L.append(Precision)
```

```
Negative_Predictive_Value= rounder((tn) / (tn + fn))
```

```
Negative_Predictive_Value_L.append(Negative_Predictive_Value)
```

```
False_Negative_Rate=rounder((fn) / (fn + tp))
```

```
False_Negative_Rate_L.append(False_Negative_Rate)
```

```
False_Positive_Rate=rounder((fp) / (fp + tn))
```

```
False_Positive_Rate_L.append(False_Positive_Rate)
```

```
False_Discovery_Rate=rounder((fp) / (fp + tp))
```

```
False_Discovery_Rate_L.append(False_Discovery_Rate)
```

```
False_Omission_Rate=rounder((fn) / (fn+ tn))
```

```
False_Omission_Rate_L.append(False_Omission_Rate)
```

```
print('Sensitivity:', Sensitivity)
```

```
print('Specificity:', Specificity)
```

```
print('F1 Score:', F1Score)
```

```
print('Precision:', Precision)
```

```
print('Negative Predictive Value:', Negative_Predictive_Value)
```

```
print('False Negative Rate:', False_Negative_Rate)
```

```
print('False Positive Rate:', False_Positive_Rate)
```

```
print('False Discovery Rate:', False_Discovery_Rate)
```

```
print('False Omission Rate:', False_Omission_Rate)
```

```
num_folds = 5
```

```
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
```

```
cv_scores = cross_val_score(model, x_train, y_train, cv=kf, scoring='accuracy')
```

```
print(f"\n{num_folds}-Fold Cross-Validation Scores:")
```

```
print(cv_scores)
```

```
average_cv_accuracy = rounder(np.mean(cv_scores))
```

```
print(f"\nAverage Cross-Validation Accuracy: {average_cv_accuracy * 100:.2f}%")
```

```
average_cv_accuracy_L.append(average_cv_accuracy)
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.impute import SimpleImputer
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn.model_selection import train_test_split, cross_val_score
```

```
from sklearn.metrics import roc_curve, auc
```

```
from sklearn.model_selection import cross_val_score, KFold
```

```
# Assuming x_train, y_train, x_test, y_test are defined
```

```
# Use SimpleImputer to fill in missing values
```

```
imputer = SimpleImputer(strategy='mean') # You can use other strategies like 'median' or  
'most_frequent'
```

```
x_train_imputed = imputer.fit_transform(x_train)
```

```
x_test_imputed = imputer.transform(x_test)
```

```
# Create a Random Forest model
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42) # You can adjust the  
number of trees (n_estimators) as needed
```

```
# Fit the model
```

```
rf.fit(x_train, y_train)
```

```
# Predict on the test set
```

```
y_pred_rf = rf.predict(x_test)
```

```
# Evaluate the model
```

```
accuracy_rf = accuracy_score(y_test, y_pred_rf)
```

```
print(" Random Forest Accuracy:", accuracy_rf)
```

```
cv_accuracy_rf = cross_val_score(rf, x_train, y_train, cv=5)
```

```
print(f"Cross-validated Random Forest Accuracy: {cv_accuracy_rf.mean()*100}")
```

```
try:
```

```
    y_prob_rf = rf.predict_proba(x_test_imputed)[:, 1]
```

```
# Compute ROC curve and ROC area for each class
```

```
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)
```

```
roc_auc_rf = auc(fpr_rf, tpr_rf)
```

```
# Plot ROC curve
```

```
plt.figure()
```

```
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve - rf (area = %0.2f)' %  
roc_auc_rf)
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
```

```
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic (ROC) - Random Forest')
```

```
plt.legend(loc="lower right")
```

```
plt.show()
```

```
except AttributeError as e:
```

```
    print("AttributeError:", e)
```

```
fun(rf,'Random Forest Classifier')
```

```
from sklearn.metrics import classification_report, accuracy_score
```

```
import matplotlib.pyplot as plt
```

```
# Train the model
```

```
rf.fit(x_train, y_train.values.ravel())
```

```
# Get predictions
```

```
y_pred_train = rf.predict(x_train)
```

```
y_pred_test = rf.predict(x_test)
```

```
# Calculate metrics for training data
```

```
report_train = classification_report(y_train, y_pred_train, output_dict=True)
```

```
accuracy_train = accuracy_score(y_train, y_pred_train)
```

```
# Calculate metrics for test data
```

```
report_test = classification_report(y_test, y_pred_test, output_dict=True)
```

```
accuracy_test = accuracy_score(y_test, y_pred_test)
```

```
# Extracting metrics for training data
```

```
precision_train = report_train['macro avg']['precision']
```

```
recall_train = report_train['macro avg']['recall']
```

```
f1_score_train = report_train['macro avg']['f1-score']
```

```
# Extracting metrics for test data
```

```
precision_test = report_test['macro avg']['precision']
```

```
recall_test = report_test['macro avg']['recall']
```

```
f1_score_test = report_test['macro avg']['f1-score']
```

```
# Plotting

labels = ['Accuracy', 'Precision', 'Recall', 'F1-Score']

train_scores = [accuracy_train, precision_train, recall_train, f1_score_train]

test_scores = [accuracy_test, precision_test, recall_test, f1_score_test]


x = range(len(labels))

width = 0.35


fig, ax = plt.subplots()

rects1 = ax.bar(x, train_scores, width, label='Train')

rects2 = ax.bar([p + width for p in x], test_scores, width, label='Test')


ax.set_ylabel('Scores')

ax.set_title('Random Forest Classifier')

ax.set_xticks([p + width/2 for p in x])

ax.set_xticklabels(labels)

ax.legend()


plt.show()
```

```
from sklearn.linear_model import LogisticRegression
```

```
# Assuming x_train, y_train, x_test, y_test are defined
```

```
# Create a logistic regression model
```

```
lr = LogisticRegression()
```

```
# Use SimpleImputer to fill in missing values
```

```
imputer = SimpleImputer(strategy='mean') # You can use other strategies like 'median' or  
'most_frequent'
```

```
x_train = imputer.fit_transform(x_train)
```

```
x_test= imputer.transform(x_test)
```

```
# Fit the model
```

```
lr.fit(x_train, y_train)
```

```
# Predict on the test set
```

```
y_pred_log_reg = lr.predict(x_test)
```

```
# Evaluate the model
```

```
accuracy_lr = accuracy_score(y_test, y_pred_log_reg)
```



```
print("Logistic Regression Accuracy:", accuracy_lr)
```

```
cv_log_reg = cross_val_score(lr, x_train_imputed, y_train, cv=5)
```

```
print(f"Cross-validated Logistic Regression Accuracy: {cv_log_reg.mean()*100}")
```

```
try:
```

```
    y_prob_log_reg = lr.predict_proba(x_test_imputed)[:, 1]
```

```
# Compute ROC curve and ROC area for each class
```

```
fpr, tpr, _ = roc_curve(y_test, y_prob_log_reg)
```

```
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve
```

```
plt.figure()
```

```
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve - lr (area = %0.2f)' %  
roc_auc)
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
```

```
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic (ROC)')
```

```
plt.legend(loc="lower right")
```

```
plt.show()
```

```
except AttributeError as e:
```

```
    print("AttributeError:", e)
```

```
fun(lr,'Logistic Regression')
```

```
from sklearn.metrics import classification_report, accuracy_score
```

```
import matplotlib.pyplot as plt
```

```
# Train the model
```

```
lr.fit(x_train, y_train.values.ravel())
```

```
# Get predictions
```

```
y_pred_train = lr.predict(x_train)
```

```
y_pred_test = lr.predict(x_test)
```

```
# Calculate metrics for training data
```

```
report_train = classification_report(y_train, y_pred_train, output_dict=True)
```

```
accuracy_train = accuracy_score(y_train, y_pred_train)
```

```
# Calculate metrics for test data
```

```
report_test = classification_report(y_test, y_pred_test, output_dict=True)
```

```
accuracy_test = accuracy_score(y_test, y_pred_test)
```

```
# Extracting metrics for training data
```

```
precision_train = report_train['macro avg']['precision']
```

```
recall_train = report_train['macro avg']['recall']
```

```
f1_score_train = report_train['macro avg']['f1-score']
```

```
# Extracting metrics for test data
```

```
precision_test = report_test['macro avg']['precision']
```

```
recall_test = report_test['macro avg']['recall']
```

```
f1_score_test = report_test['macro avg']['f1-score']
```

```
# Plotting
```

```
labels = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
```

```
train_scores = [accuracy_train, precision_train, recall_train, f1_score_train]
```

```
test_scores = [accuracy_test, precision_test, recall_test, f1_score_test]
```

```
x = range(len(labels))
```

```
width = 0.35
```

```
fig, ax = plt.subplots()
```

```
rects1 = ax.bar(x, train_scores, width, label='Train')
```

```
rects2 = ax.bar([p + width for p in x], test_scores, width, label='Test')
```

```
ax.set_ylabel('Scores')
```

```
ax.set_title('Logistic Regression')
```

```
ax.set_xticks([p + width/2 for p in x])
```

```
ax.set_xticklabels(labels)
```

```
ax.legend()
```

```
plt.show()
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.impute import SimpleImputer
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn.model_selection import train_test_split, cross_val_score
```

```
# Assuming x_train, y_train, x_test, y_test are defined
```

```
# Create a Gaussian Naive Bayes model
```

```
nb = GaussianNB()
```

```
# Use SimpleImputer to fill in missing values
```

```
imputer = SimpleImputer(strategy='mean') # You can use other strategies like 'median' or  
'most_frequent'
```

```
x_train = imputer.fit_transform(x_train)
```

```
x_test= imputer.transform(x_test)
```

```
# Fit the model
```

```
nb.fit(x_train, y_train)
```

```
# Predict on the test set
```

```
y_pred_nb = nb.predict(x_test)
```

```
# Evaluate the model
```

```
accuracy_nb = accuracy_score(y_test, y_pred_nb)
```

```
print("Navie Bayes Accuracy:", accuracy_nb)
```

```
cv_accuracy = cross_val_score(nb, x_train, y_train, cv=5)

print(f"Cross-validated naive bayes Accuracy: {cv_accuracy.mean()*100}")

try:

    y_prob_nb = nb.predict_proba(x_test)[:, 1]

# Compute ROC curve and ROC area for each class

    fpr_nb, tpr_nb, _ = roc_curve(y_test, y_prob_nb)

    roc_auc_nb = auc(fpr_nb, tpr_nb)

# Plot ROC curve

    plt.figure()

    plt.plot(fpr_nb, tpr_nb, color='darkorange', lw=2, label='ROC curve - nb (area = %0.2f)' %
roc_auc_nb)

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

    plt.xlim([0.0, 1.0])

    plt.ylim([0.0, 1.05])

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

    plt.title('Receiver Operating Characteristic (ROC)')

    plt.legend(loc="lower right")
```

```
plt.show()
```

```
except AttributeError as e:
```

```
    print("AttributeError:", e)
```

```
fun(nb,'Navie Bayes')
```

```
from sklearn.metrics import classification_report, accuracy_score
```

```
import matplotlib.pyplot as plt
```

```
# Train the model
```

```
nb.fit(x_train, y_train.values.ravel())
```

```
# Get predictions
```

```
y_pred_train = nb.predict(x_train)
```

```
y_pred_test = nb.predict(x_test)
```

```
# Calculate metrics for training data
```

```
report_train = classification_report(y_train, y_pred_train, output_dict=True)
```

```
accuracy_train = accuracy_score(y_train, y_pred_train)
```

```
# Calculate metrics for test data
```

```
report_test = classification_report(y_test, y_pred_test, output_dict=True)
```

```
accuracy_test = accuracy_score(y_test, y_pred_test)
```

```
# Extracting metrics for training data
```

```
precision_train = report_train['macro avg']['precision']
```

```
recall_train = report_train['macro avg']['recall']
```

```
f1_score_train = report_train['macro avg']['f1-score']
```

```
# Extracting metrics for test data
```

```
precision_test = report_test['macro avg']['precision']
```

```
recall_test = report_test['macro avg']['recall']
```

```
f1_score_test = report_test['macro avg']['f1-score']
```

```
# Plotting
```

```
labels = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
```

```
train_scores = [accuracy_train, precision_train, recall_train, f1_score_train]
```

```
test_scores = [accuracy_test, precision_test, recall_test, f1_score_test]
```

```
x = range(len(labels))
```



```
width = 0.35
```

```
fig, ax = plt.subplots()
```

```
rects1 = ax.bar(x, train_scores, width, label='Train')
```

```
rects2 = ax.bar([p + width for p in x], test_scores, width, label='Test')
```

```
ax.set_ylabel('Scores')
```

```
ax.set_title('Navie Bayes')
```

```
ax.set_xticks([p + width/2 for p in x])
```

```
ax.set_xticklabels(labels)
```

```
ax.legend()
```

```
plt.show()
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.impute import SimpleImputer
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn.model_selection import train_test_split, cross_val_score
```

```
# Assuming x_train, y_train, x_test, y_test are defined

gbc= GradientBoostingClassifier(n_estimators = 15, max_features = None,
min_samples_split = 2)


# Use SimpleImputer to fill in missing values

imputer = SimpleImputer(strategy='mean') # You can use other strategies like 'median' or
'most_frequent'

x_train = imputer.fit_transform(x_train)

x_test = imputer.transform(x_test)


# Fit the model

gbc.fit(x_train, y_train)


# Predict on the test set

y_pred_gbc = gbc.predict(x_test)


# Evaluate the model

accuracy_gbc = accuracy_score(y_test, y_pred_gbc)

print("Gradient Boosting Accuracy:", accuracy_gbc)


cv_accuracy_gbc = cross_val_score(gbc, x_train, y_train, cv=5)
```

```
print(f"Cross-validated Gradient Bossting Classifier Accuracy:  
{cv_accuracy_gbc.mean()*100}")
```

```
try:
```

```
    y_prob_gbc = gbc.predict_proba(x_test)[:, 1]
```

```
# Compute ROC curve and ROC area for each class
```

```
fpr_gbc, tpr_gbc, _ = roc_curve(y_test, y_prob_gbc)
```

```
roc_auc_gbc = auc(fpr_gbc, tpr_gbc)
```

```
# Plot ROC curve
```

```
plt.figure()
```

```
plt.plot(fpr_nb, tpr_nb, color='darkorange', lw=2, label='ROC curve - gbc (area = %0.2f)'  
% roc_auc_gbc)
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
```

```
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic (ROC)')
```

```
plt.legend(loc="lower right")
```

```
plt.show()
```

```
except AttributeError as e:
```

```
    print("AttributeError:", e)
```

```
fun(gbc,'Gradient Boosting Classifier')
```

```
from sklearn.metrics import classification_report, accuracy_score
```

```
import matplotlib.pyplot as plt
```

```
# Train the model
```

```
gbc.fit(x_train, y_train.values.ravel())
```

```
# Get predictions
```

```
y_pred_train = gbc.predict(x_train)
```

```
y_pred_test = gbc.predict(x_test)
```

```
# Calculate metrics for training data
```

```
report_train = classification_report(y_train, y_pred_train, output_dict=True)
```

```
accuracy_train = accuracy_score(y_train, y_pred_train)
```

```
# Calculate metrics for test data
```

```
report_test = classification_report(y_test, y_pred_test, output_dict=True)
```

```
accuracy_test = accuracy_score(y_test, y_pred_test)
```

```
# Extracting metrics for training data
```

```
precision_train = report_train['macro avg']['precision']
```

```
recall_train = report_train['macro avg']['recall']
```

```
f1_score_train = report_train['macro avg']['f1-score']
```

```
# Extracting metrics for test data
```

```
precision_test = report_test['macro avg']['precision']
```

```
recall_test = report_test['macro avg']['recall']
```

```
f1_score_test = report_test['macro avg']['f1-score']
```

```
#Plotting
```

```
labels = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
```

```
train_scores = [accuracy_train, precision_train, recall_train, f1_score_train]
```

```
test_scores = [accuracy_test, precision_test, recall_test, f1_score_test]
```

```
x = range(len(labels))
```

```
width = 0.35
```

```

fig, ax = plt.subplots()

rects1 = ax.bar(x, train_scores, width, label='Train')

rects2 = ax.bar([p + width for p in x], test_scores, width, label='Test')


ax.set_ylabel('Scores')

ax.set_title('Gradient Boosting Classifier')

ax.set_xticks([p + width/2 for p in x])

ax.set_xticklabels(labels)

ax.legend()


plt.show()

```

```

compare = pd.DataFrame({'Model': ['RANDOM FOREST','GRADIENT BOOSTING
CLASSIFIER','LOGISTIC REGRESSION','NAIVE BAYES'],

                        'Accuracy': [accuracy_rf*100,accuracy_gbc*100, accuracy_lr*100,
accuracy_nb*100],

                        'Training Accuracy':Training_Accuracy_L,

                        'Test Accuracy':Test_Accuracy_L,

                        'Sensitivity':Sensitivity_L,

                        'Specificity':Specificity_L,

```

```

        'F1 Score':F1Score_L,

        'Precision':Precision_L,

        'Negative Predictive Value':Negative_Predictive_Value_L,

        'False Negative Rate':False_Negative_Rate_L,

        'False Positive Rate':False_Positive_Rate_L,

        'False Discovery Rate':False_Discovery_Rate_L,

        'False Omission Rate':False_Omission_Rate_L,

        'Average cv-accuracy':average_cv_accuracy_L })

compare.sort_values(by='Accuracy', ascending=False)

colors = ['skyblue', 'orange', 'green', 'red', 'purple', 'pink']

plt.figure(figsize=(10, 6))

plt.barh(compare['Model'], compare['Accuracy'], color=colors)

plt.xlabel('Accuracy (%)')

plt.title('Model Comparison - Accuracy')

plt.xlim(0, 100)

for index, value in enumerate(compare['Accuracy']):

    plt.text(value, index, f'{value:.2f}', va='center', fontsize=10)

plt.show()

```

```
import pickle
```

```
with open('model.pkl','wb') as f:
```

```
    pickle.dump(rf,f)
```

```
model=pickle.load(open('model.pkl','rb'))
```

FLASK CODE-

```
import numpy as np
```

```
from flask import Flask, request, jsonify, render_template
```

```
import pickle
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import LabelEncoder
```

```
app = Flask(__name__)
```

```
# Load the trained model
```

```
model=pickle.load(open('model.pkl','rb'))
```

```
@app.route('/')
```

```
def index():
```

```
    return render_template('index.html')
```

```
@app.route('/predict', methods=['POST'])
```

```
def predict():
```



```

# Collect features from form data

statuses_count=int(request.form['statuses_count'])

followers_count=float(request.form['followers_count'])

friends_count=float(request.form['friends_count'])

listed_count=int(request.form['listed_count'])

favourites_count=int(request.form['favourites_count'])

lang=int(request.form['lang'])

default_profile=int(request.form['default_profile'])

profile_use_background_image=int(request.form['profile_use_background_image'])

feature_array=[[statuses_count,followers_count,friends_count,listed_count,favourites_count,l
ang,default_profile,

                profile_use_background_image]]

#print(features_array)

"encoded_features = []

for feature in features:

    if isinstance(feature, str):

        print("string")

        encoded_feature = le.fit_transform([feature])

        encoded_features.append(encoded_feature[0])

    else:

        encoded_features.append(feature)

```

```
features_array = np.array(encoded_features).reshape(1, -1)

print(features_array)

'''

features_array = np.array(feature_array).reshape(1, -1)

print(feature_array)

prediction = model.predict(features_array)[0]

print(prediction)

# Output the prediction

if prediction == 1:

    output='FAKE'

else:

    output='REAL'

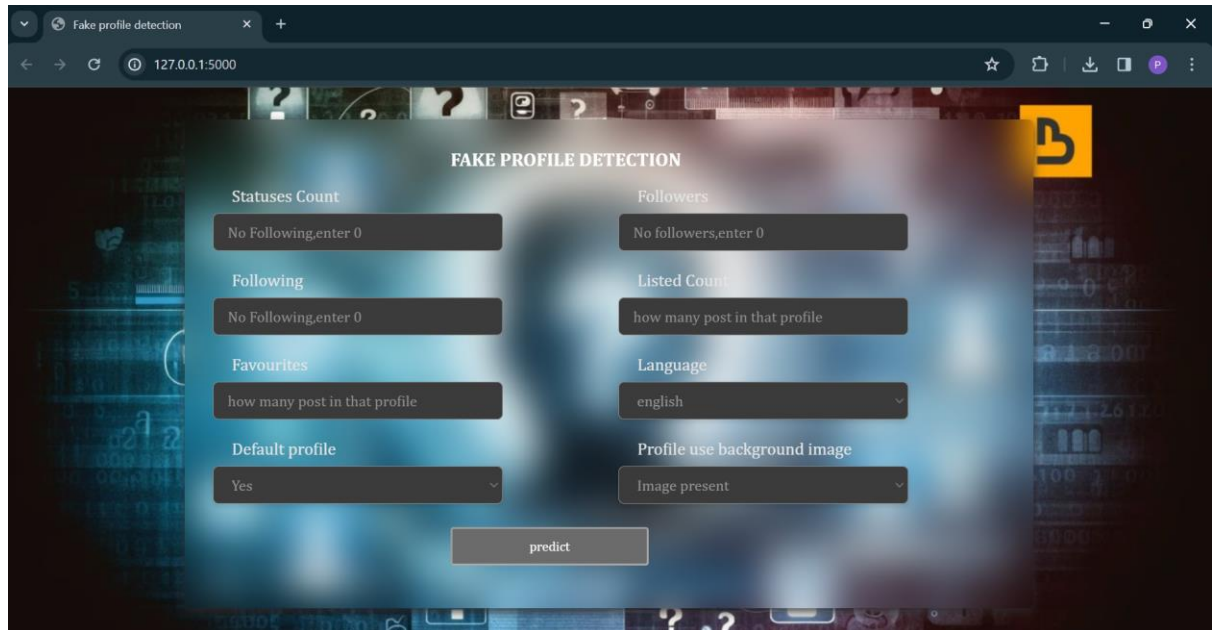
return render_template('index.html', prediction_text='Prediction: {}'.format(output))

if __name__ == "__main__":

    app.run(debug=True)
```

V. SCREEN SHOTS

HOME PAGE:

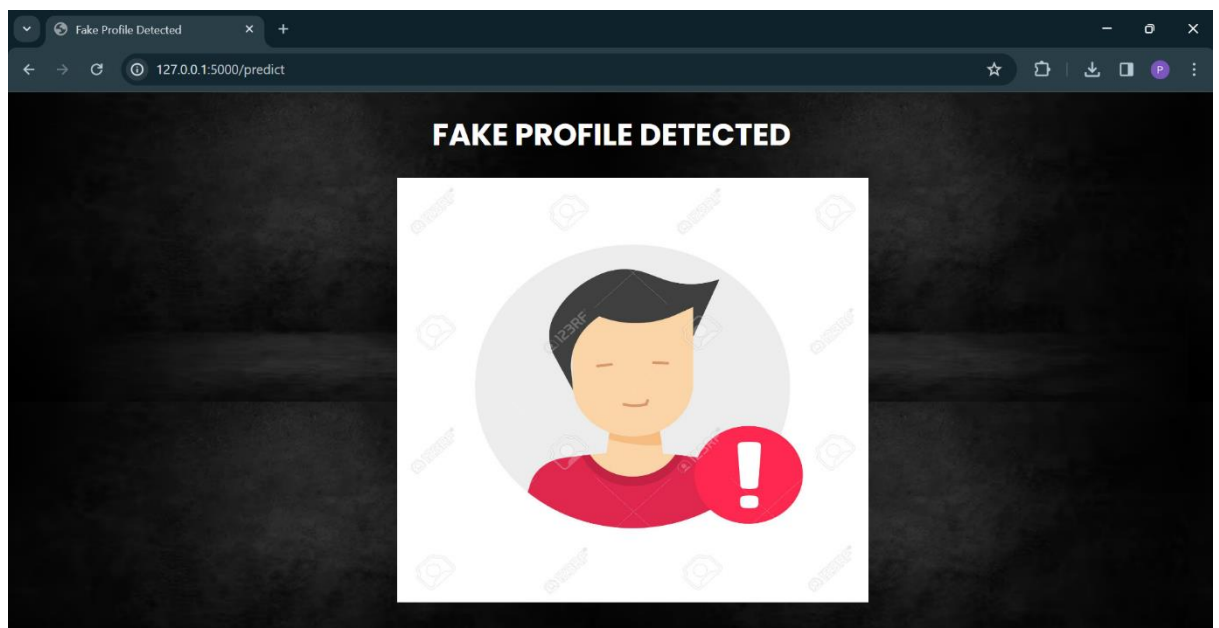


A screenshot of a web browser displaying the 'FAKE PROFILE DETECTION' home page. The browser's address bar shows '127.0.0.1:5000'. The page has a dark background with a grid of input fields for profile data. The fields are arranged in two columns. The left column includes 'Statuses Count' (with a placeholder 'No Following,enter 0'), 'Following' (with a placeholder 'No Following,enter 0'), 'Favourites' (with a placeholder 'how many post in that profile'), and 'Default profile' (a dropdown menu set to 'Yes'). The right column includes 'Followers' (with a placeholder 'No followers,enter 0'), 'Listed Count' (with a placeholder 'how many post in that profile'), 'Language' (a dropdown menu set to 'english'), and 'Profile use background image' (a dropdown menu set to 'Image present'). A 'predict' button is centered at the bottom of the form area.

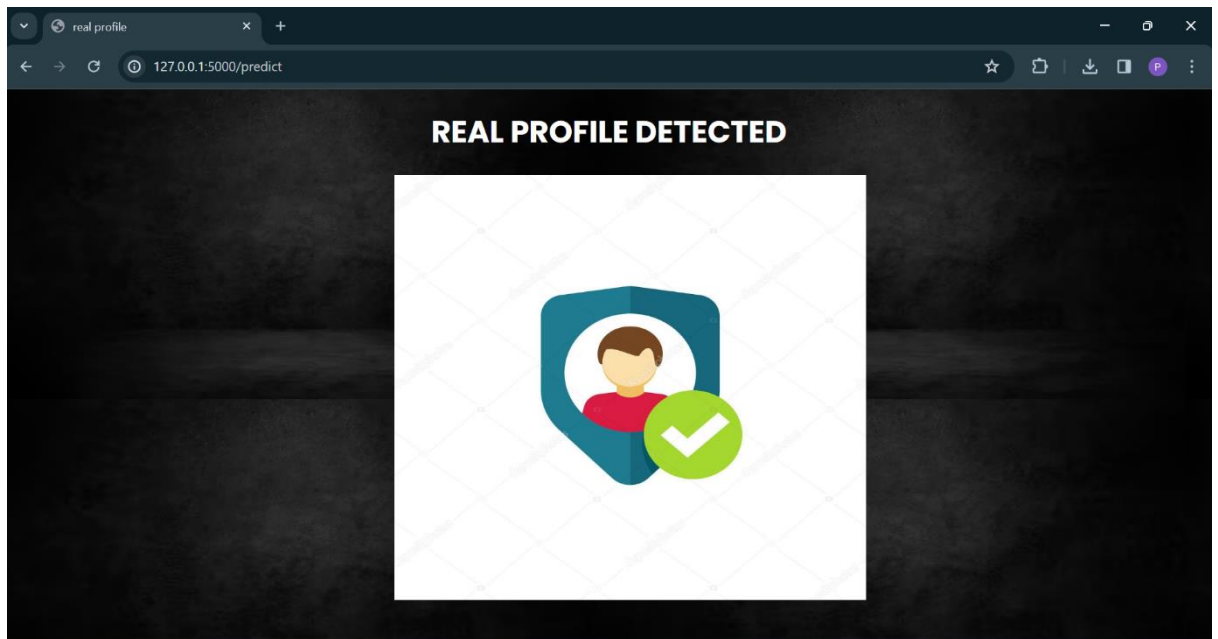
Field	Value
Statuses Count	No Following,enter 0
Following	No Following,enter 0
Favourites	how many post in that profile
Default profile	Yes
Followers	No followers,enter 0
Listed Count	how many post in that profile
Language	english
Profile use background image	Image present

predict

PREDICTION PAGE IF PROFILE IS DETECTED AS FAKE:



PREDICTION PAGE IF PROFILE IS DETECTED AS REAL:



VI. RESULTS AND ANALYSIS

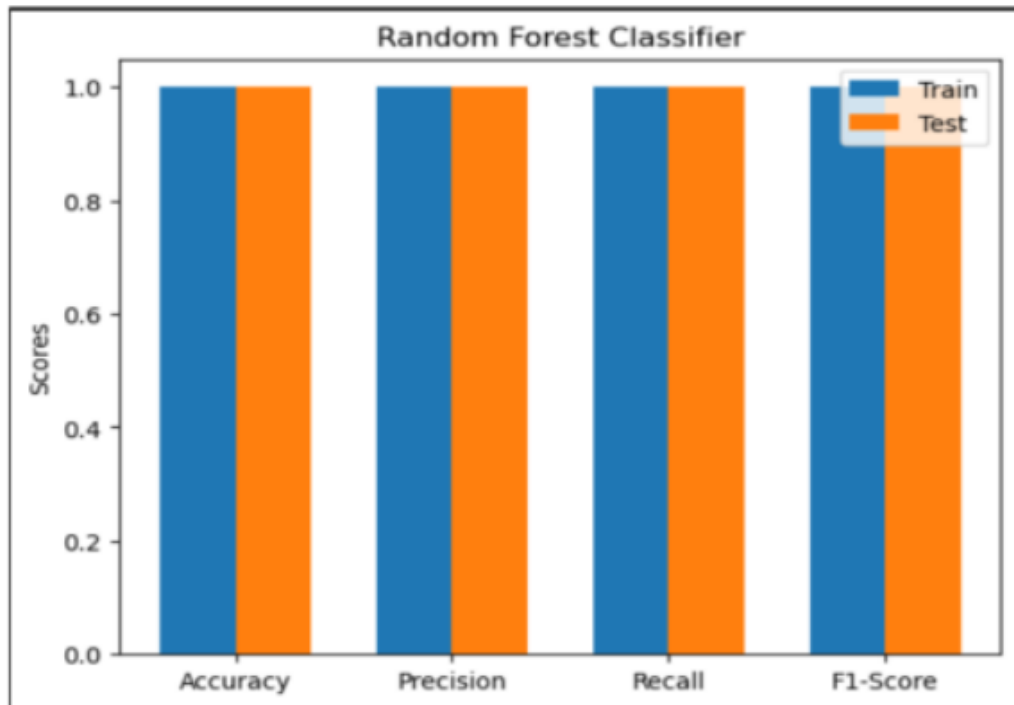


Fig.18 Metrics of the Random Forest Classifier

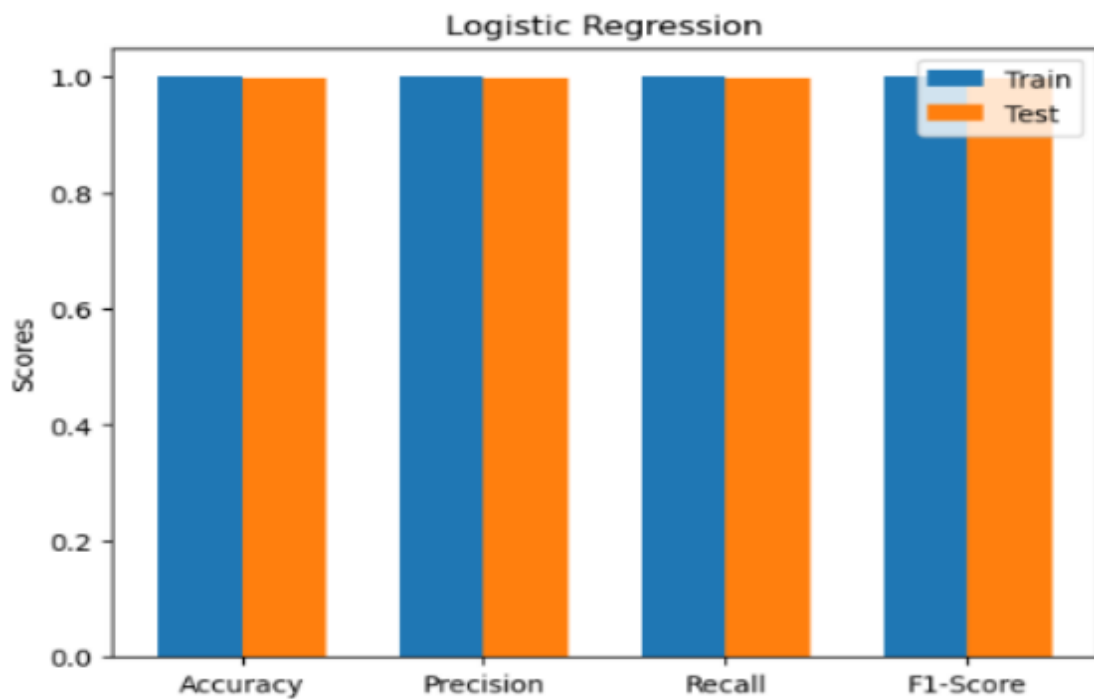


Fig.19. Evaluation metrics of Logistic regression

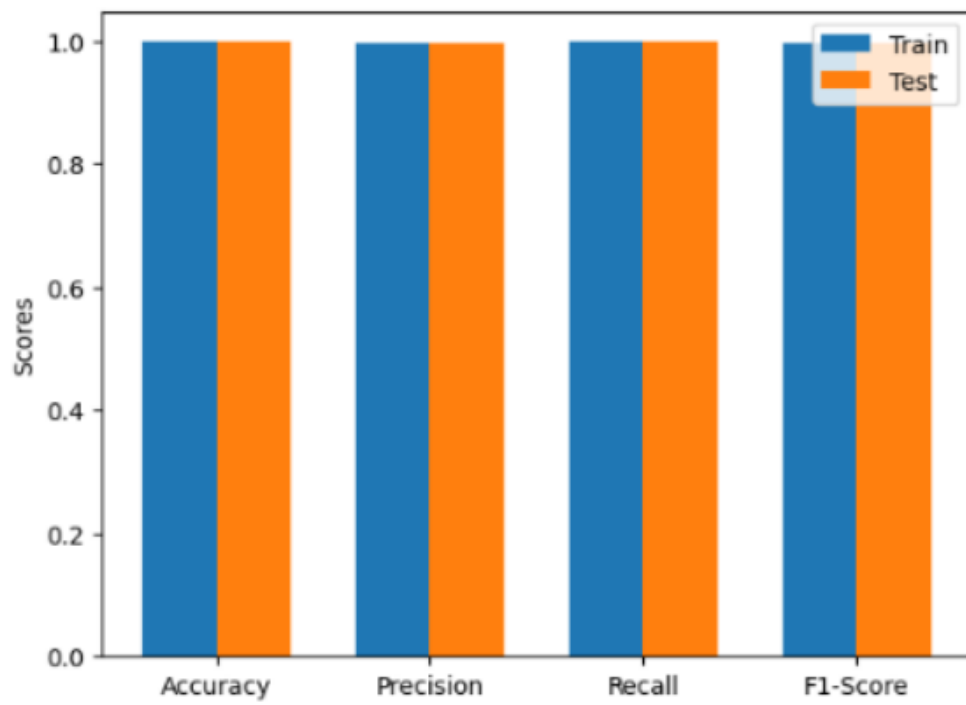


Fig.20. Evaluation metrics of naïve bayes

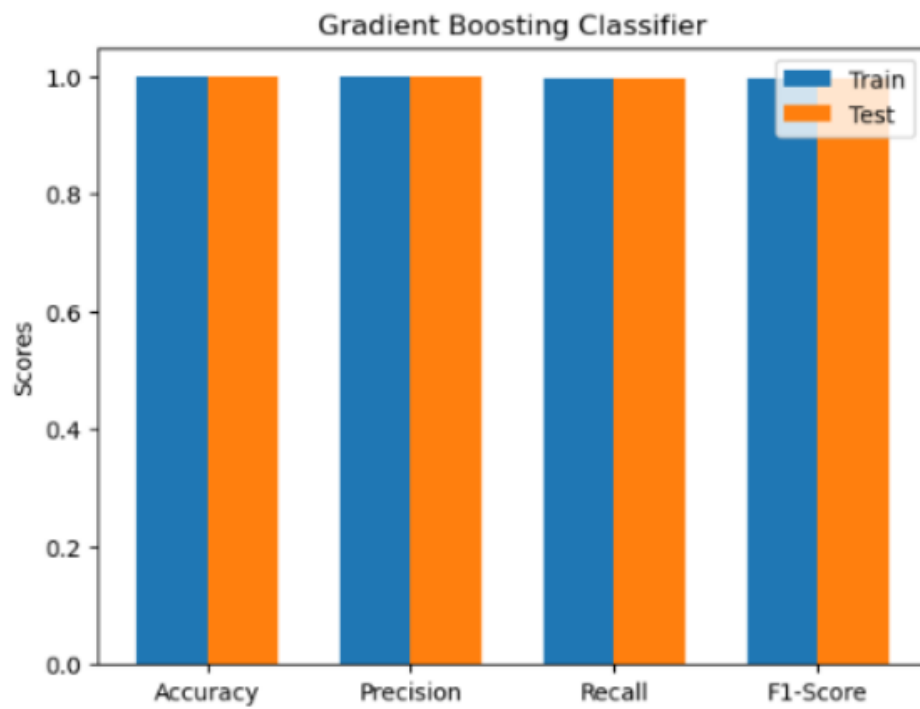


Fig.21. Evaluation metrics of Gradient Boosting Classifier

VII. CONCLUSION

We have used 4 machine learning algorithms namely Random forest, Naïve Bayes, Logistic Regression, and the gradient boosting classifier in order to predict whether a user profile is fake or genuine. The accuracy varies for different algorithms. The accuracy for the random forest algorithm is 99.46. The accuracy for the naïve bayes classification algorithm is 29.22. The accuracy for the logistic regression is 98.93. The accuracy for the gradient boosting classifier is 99.46. After performing the k-fold on the random forest, we obtain the model accuracy of 99.59. After performing the k-fold on the gradient boosting classifier, we obtain the model accuracy of 99.52. After performing the k-fold on the naïve bayes classification algorithm, we obtain the model accuracy of 99.19. After performing the k-fold on the logistic regression, we obtain the model accuracy of 99.59. We also applied the cross-validation on these algorithms. On applying the cross-validation on the random forest algorithm, we obtained the average cross-validation accuracy score of 100. On applying the cross-validation on the naïve bayes classification algorithm, we obtained the average cross-validation accuracy score of 99.30. On applying the cross-validation on the gradient boosting classification algorithm, we obtained the average cross-validation accuracy score of 99.60. On applying the cross-validation on the logistic regression algorithm, we obtained the average cross-validation accuracy score of 99.20.

VIII. FUTURE SCOPE

This project can be developed into a mobile application in the future.

REFERENCES

- [1] Gururaj, H.L., Tanuja, U., Janhavi, V., Ramesh, B.: Detecting malicious users in the social networks using machine learning approach. *Int. J. Soc. Comput. CyberPhys. Syst.* 2(3), 229–243 (2021).
- [2] 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.
- [3] 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.
- [4] Khaled, S., El-Tazi, N. and Mokhtar, H.M. (2018) Detecting Fake Accounts on Social Media. 2018 IEEE International Conference on Big Data (Big Data), Seattle, 10-13 December 2018, 3672-3681.
- [5] K.Harish, R.Naveen Kumar, Dr.J.Briso Becky Bell(2023) Fake profile detection using machine learning.
- [6] Fake Profile Identification Using Machine Learning: Asso. Prof. G Swathi, R Vaishnavi, Shaik Noorus Sabiha, P Rakesh Anand, P Nithish Kumar Sreyas Institute of Engineering and Technology(2023).
- [7] Kulkarni, Sumit Milind, and Vidya Dhamdhare."Automatic detection of fake profiles in online social networks." *Open access International Journal of Science and Engineering* 3.1 (2018):70-73. M. Young, *The Technical Writers Handbook*. Mill Valley, CA: University Science,1989.
- [8] Sarah Khaled, Neamat El-Tazi, Hoda M. O.Mokhta: Detecting Fake Accounts on Social Media,2018 IEEE International Conference on Big Data (Big Data).
- [9] 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.
- [10] Joshi, U.D., Singh, A.P., Pahuja, T.R., Naval, S. and Singal, G. (2021) Fake Social Media Profile Detection. In: Srinivas, M., Sucharitha, G., Matta, A. and Chatterjee, P., Eds.,

Machine Learning Algorithms and Applications, Scrivener Publishing LLC, Beverly, MA, 193-209.

[11] Reddy, S.D.P. (2019) Fake Profile Identification Using Machine Learning. International Research Journal of Engineering and Technology (IRJET), 6, 1145-1150.

[12] Exploring machine learning techniques for fake profile detection in online social networks: Bharti, Nasib Gill, Preeti Gulia vol. 3, June 2023, pp. 2962~2971.

[13] Sk.Shama, K. Siva Nandini, P.Bhavya Anjali, K. Devi Manaswi, Fake Profile.Identification in Online Social Network, 2019.

[14] Prof of. Vivek Solvande, Vaishnavi Ambolkar, Siddhesh Birmole, Divya Gawas, Dnyanada Juvale and Fake Profile Identification Using Machine Learning Algorithm,2021.

[15] S. Sheikhi, An Efficient Method for Detection of Fake Accounts on the Instagram Platform, 2020.

[16] T. Sudhakar, Bhuvana Chendrica Gogineni, J. Vijaya "Fake Profile Identification Using Machine Learning" IEEE International Women in Engineering(WIE) Conference on Electrical and Computer Engineering(WIECON-ECE) 2022.