

TWITTER PAGE AUTHENTICITY CHECKER

A PROJECT REPORT

Submitted by

HARINI PRIYA K

K.A.DHEEPTHAA

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

SSN COLLEGE OF ENGINEERING, CHENNAI

ANNA UNIVERSITY :: CHENNAI 600 025

APRIL 2019

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that the project report **“TWITTER PAGE AUTHENTICITY CHECKER”** is the bonafide work of **“HARINI PRIYA K (312215205030) and K.A.DHEEPTHAA (312215205042)”** who carried out the project work under my supervision.

SIGNATURE

Dr. T. NAGARAJAN

HEAD OF THE DEPARTMENT

Professor

Department of Information Technology

SSN COLLEGE OF ENGINEERING

Old Mahabalipuram Road,

Kalavakkam,

Tamil Nadu 603110

SIGNATURE

Dr. K.R.UTHAYAN

SUPERVISOR

Associate Professor

Department of Information Technology

SSN COLLEGE OF ENGINEERING

Old Mahabalipuram Road,

Kalavakkam,

Tamil Nadu 603110

Submitted for the University Project Viva-voce held at Sri Sivasubramaniya Nadar College of Engineering on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our gratitude to our Principal, Dr.S.Salivahanan, and to the Head of Department, Dr.T.Nagarajan, who gave us this opportunity to do this wonderful project on the topic, Twitter Page Authenticity Checker as our final year project, which was a great learning experience and helped us in doing a lot of research.

Next, it is a matter of pleasure of acknowledgement by indebtedness to our project guide, Dr.K.R.Uthayan, for his needy half in the completion of the project and able guidance.

We sincerely thank our project panel members in the Department of Information Technology for providing us with helpful pointers throughout the course of the project.

We would also like to thank our friends and family for constantly encouraging us throughout the course of the project. Your encouragement and kind words will never be forgotten.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	LIST OF TABLES	v
	LIST OF FIGURES	vi
1.	INTRODUCTION	1
	1.1 Current Scenario In Twitter	1
	1.2 Objective	2
	1.3 Overview Of Project	3
2.	LITERATURE SURVEY	4
	2.1 Information Credibility on Twitter	4
	2.2 Detecting Spammers on Twitter	5
	2.3 Profile Characteristics of Fake Twitter Accounts	6
	2.4 Fake Account Detection in Twitter Based on Minimum Weighted Feature set	7
3.	SYSTEM ARCHITECTURE	8
	3.1 Attributes Of Twitter Page	9
	3.2 Machine Learning Model	11
	3.3 Dataset	17
	3.4 Tweepy	18
	3.5 User Interface	19
4.	TECHNICAL IMPLEMENTATION	20
	4.1 Algorithm	20
	4.2 Program Code Description	22
	4.2.1 Twitter Data Fetcher	22

	4.2.2	Training Model	23
	4.2.3	Prediction Model	27
	4.2.4	Web Page	27
5.		RESULTS	29
	5.1	Verified account	29
	5.2	Authentic, non-verified account	30
	5.3	Fake account	31
	5.4	Training Epochs	32
6.		CONCLUSION	33
		REFERENCES	34

ABSTRACT

Twitter is a social media/ micro-blogging platform used by millions across the world. Due to its explosive growth, there are a total of 330 million active Twitter users every month, and close to 500 million tweets every day. Hence, it is not surprising that most celebrities have a Twitter account.

75% of Twitter users are said to use Twitter as a source of news. Therefore, for such users, it is important to know whether the celebrity accounts being followed are authentic or fake.

Although Twitter provides the “verified” feature, not all authentic celebrity accounts have been verified by Twitter. This may be because the person is a new celebrity, or he/she is not too well-known, or other such reasons. Hence, there is a need to distinguish between such accounts and the respective fake accounts created for those celebrities.

As mentioned, although most celebrities are on Twitter, there are many who don’t own a Twitter account. For such people, there is an ever-existing set of fake Twitter accounts. Hence, there is a need to know that they are fake as well.

Our project aims to provide an authenticity score for a given Twitter account URL. Users may simply provide the URL of the account they have doubts about, for which an authenticity score will be displayed.

LIST OF TABLES

Table No.	Title	Page No.
3.1	Normalised Attributes	10
3.2	Comparison of Algorithms	17
4.1	Tweeepy Methods	22

LIST OF FIGURES

Figure No.	Title	Page No.
1.1.1	Fake account	2
1.1.2	Authentic account	2
3.1	System Architecture	8
3.2	Backpropagation	11
3.3	LSTM	13
3.4	K-Nearest Neighbours	14
3.5	Decision Trees	14
3.6	LDA	15
3.7	Simple Neural Network	16
3.8	Tweepy Flow	18
5.1.1	Verified Account Form	29
5.1.2	Verified Account Result	30
5.2.1	Authentic Non-Verified Account Form	30
5.2.2	Authentic Non-Verified Account Result	31
5.3.1	Fake Account Form	31
5.3.2	Fake Account Result	32
5.4	Training Epochs	32

CHAPTER 1

INTRODUCTION

This chapter explains the status-quo in Twitter and the objective of this project followed by a brief overview.

1.1. Current scenario in Twitter

Twitter is an extensively used social media/micro-blogging platform used by millions of users. 75% of Twitter users are said to use Twitter as a source of news. There are thousands of fake profiles/accounts that seem to be merely authentic and spread wrong source of information. A disclosure from Twitter said the company's technology was able to identify more than 9.9 million potential spam accounts per week. Taken at face value, Twitter's efforts means more than 20 percent of the platform's 336 million strong active users could have been purged from the site.

A fake Twitter account is defined as an account owned by someone who has intentions of impersonating the respective person and who's likely to post fake news.

Different spam/fake accounts from following sources:

- Bots and Machines:

Around **15% - 20%** of the Twitter accounts are bots.

- Company Accounts:

Around **40%** of the Twitter handles one comes across will be brand accounts run by businesses or companies.

- Partial Automation

Even several personal accounts use different levels of automation depicting bot/machine like behaviour. Some of them really intended spam.



Fig.1.1.1: A fake account (Source: <https://twitter.com/BleedSACHINism>)



Fig.1.1.2: An authentic account (Source: https://twitter.com/sachin_rt)

1.2. Objective

Our project aims to provide an authenticity score for a given Twitter account URL. Users may simply provide the URL of the account they have

doubts about, for which an authenticity score will be displayed. This web-app helps to detect fake account of celebrities/news pages. It also helps us to follow more reliable accounts/pages. The purpose of using Twitter becomes more meaningful because users get to follow more authentic accounts.

1.3. Overview of the project

There are 3 classes of Twitter users to consider for whom fake pages exist:

- Celebrities who have a verified account.
- Celebrities who have a non-verified but authentic account
- Celebrities who do not have an account.

We have used different machine learning models which include various classifiers and Keras frontend framework and TensorFlow backend. We have compared different models and the best accuracy was obtained in Backpropagation network model using Keras with accuracy of 96.8%.

The Tweepy API is used for fetching attributes such as followers, friends, likes, retweets, first search result etc. These attributes are used to train the model and the model is saved. The prediction is done using this saved model. The web-app runs on top of flask server on which it is deployed.

When the URL is fed into the web-app in real time, it fetches all the required attributes from Twitter through Tweepy API and the trained model is used for predicting how reliable the given page is the results are displayed in the UI which is returned by the model to flask app.

CHAPTER 2

LITERATURE SURVEY

For the purpose of inspiration and comparison, four publications were analysed in terms of

- Attributes/Features of the Twitter page
- Machine Learning models/algorithms
- Accuracy

2.1 Information Credibility on Twitter [1]

Authors: Barbara Poblete, Carlos Castillo, Marcelo Mendoza

According to the above authors, they analyze the information credibility of news propagated through Twitter. Research has shown that most of the messages posted on Twitter are truthful, but the service is also used to spread misinformation and false rumours, often unintentionally.

The attributes inspired from this publication were

- Has URL
- Has Description
- Has Profile Pic

The algorithm here involved minimization of Root Mean Square(RMS) error. Hence, Backpropagation was chosen for our project as it implements this principle.

In addition to the above attributes, there are many more attributes considered in this publication that are not considered in the project because we are not interested in mining the tweets' contents and also because of API constraints.

2.2 Detecting Spammers on Twitter [2]

Authors: Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, Virgílio Almeida

According to the above authors, they consider the problem of detecting spammers on Twitter. Firstly, they collected a large dataset of Twitter that includes more than 54 million users, 1.9 billion links, and almost 1.8 billion tweets. Using tweets related to three famous trending topics from 2009, they construct a large labelled collection of users, manually classified into spammers and non-spammers. They then identify a number of characteristics related to tweet content and user social behaviour, which could potentially be used to detect spammers. They used these characteristics as attributes of machine learning process for classifying users as either spammers or non-spammers.

The attributes inspired from this publication are:

- Followers and Friends fraction
- Average number Of URLs per tweet

This publication used classification algorithms which we too implemented as a part of inspiration. In addition to the above attributes, there are many more attributes considered in the paper that are our not considered in the project because we are not interested in mining the tweets' contents and also because of API constraints. Further, we also could not attain the extent of scalability as in the above publication.

2.3. Profile characteristics of fake Twitter accounts [3]

Authors: Supraja Gurajala, Joshua S White, Brian Hudson, Brian R Voter, Jeanna N Matthews

According to the above authors, the Sybils/ Socialbot type of fake accounts are aimed to be detected. These are fake profiles which are created automatically/semi automatically by cyber criminals through bots. Around 62 million accounts were collected using web crawling, and accounts were grouped based on whether they had same name, description and location. The screen name of such groups was analyzed. The results of this publication however stated that many false positives arose by just analyzing the screen name, hence we too did not consider analyzing it.

The update times of an account was then used for analysis. Those accounts which were updated periodically at certain times of the day/week/month etc. could be safely assumed to be the work of a bot. This seemed to be a strong feature but the Tweepy API does not provide such update details now, and hence we could not make use of it as an attribute.

2.4. Fake Account Detection in Twitter Based on Minimum Weighted Feature Set [4]

Authors: Ahmed El Azab, Amira M. Idrees, Mahmoud A. Mahmoud, Hesham Hefny

According to the above authors, a 5 fold cross validation classification algorithm was performed on a minimal number of attributes after which weights were assigned to the attributes using GAIN measure. The authors mentioned that they preferred using as minimal attributes as possible, because they had experimentally found that using a large number of attributes which do not show consistent behaviour for both authentic and fake accounts would affect the effectiveness. Hence, we too decided to choose an optimal number of attributes.

The attributes inspired from this publication were

- Number of retweets
- Number of likes

An attribute which isn't common, i.e., number of times the user is mentioned was felt to be difficult to implement as it would require searches across millions of tweets which we cannot achieve in our level of algorithm knowledge.

SUMMARY

A set of 4 publications, the problem statement of which were similar to ours, were analysed from which we were inspired to add 11 attributes. The machine learning algorithms to use were also decided upon from this survey.

CHAPTER 3

SYSTEM ARCHITECTURE

This chapter explains the various components involved in the project namely:

- Attributes of the Twitter Page
- Machine Learning Models
- Tweepy
- User Interface

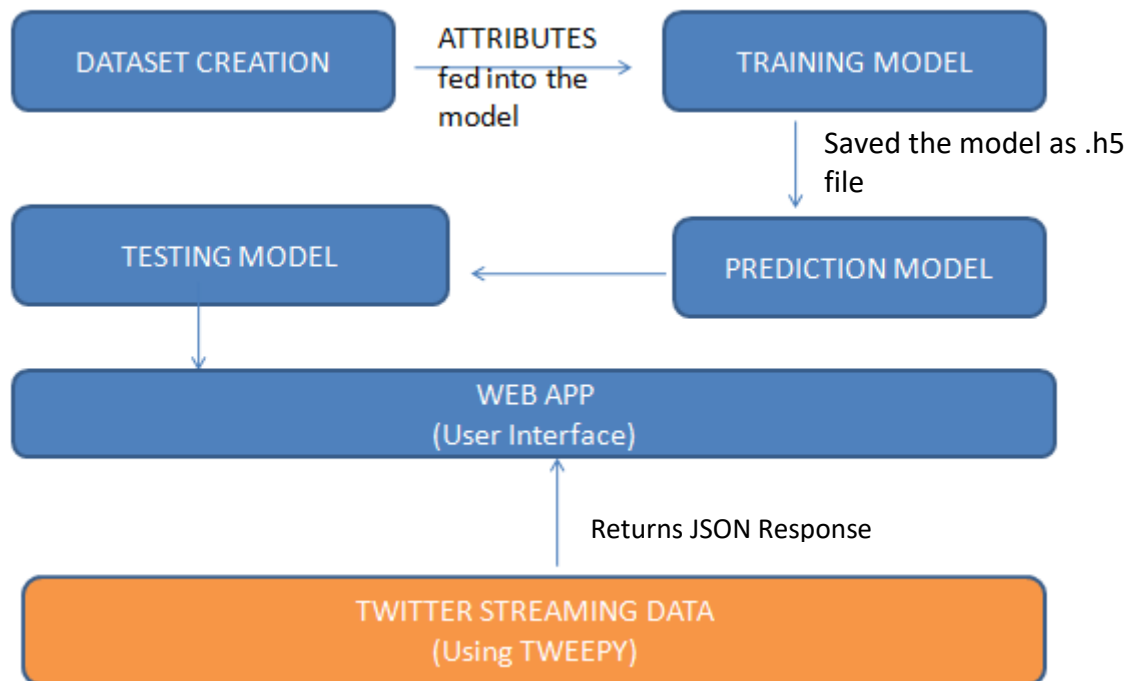


Fig.3.1: System Architecture

The above diagram depicts the flow of the project. Each component is explained in detail in the further sections.

3.1. Attributes of the Twitter Page

- *Followers*

This attribute gives the total followers count for the particular account. Ideally, celebrities have a large number of followers in the range of thousands to millions.

- *Friends*

This attribute gives the total number of accounts being followed by the particular account. Ideally, celebrities follow less number of accounts.

- *IsVerified*

This attribute denotes whether the given account is verified by Twitter or not. Twitter generally assigns a “blue tick” to such verified accounts.

- *Likes*

This attribute denotes the number of likes for a tweet of the given account. Likes are generally high in number for celebrity accounts.

- *Retweets*

This attribute denotes the number of replies/mentions that a particular tweet in an account has received. Retweets are generally high in number for celebrity accounts.

- *Profile Pic*

This attribute checks the presence of a profile picture for the given account. Celebrities generally have a Profile Picture uploaded.

- *Bio/Description*

This attribute refers to the presence of the bio/about section of the given account.

- *First Search Result*

This attribute denotes whether the first search result for the name of the given account is the account itself or not. Authentic accounts usually appear as first search result for their respective name.

NORMALISED ATTRIBUTES FED TO THE ML MODEL

ATTRIBUTES	CALCULATION
Followers	Followers/Max Followers
Friends	Friends/Max Friends
Followers Ratio	Followers/(Followers+Friends)
Friends Ratio	Friends/(Followers+Friends)
Average Number Of Likes	Average number of likes for 'n' tweets
Average Number Of Retweets	Average Number of retweets for 'n' tweets
Has Profile Pic	True - if exists False – otherwise
Has Bio	True - if exists False – otherwise
First Search Result	True - if search result for name of account is first False – otherwise
Retweets Fraction	Number of tweets which have been retweeted / Total number of tweets considered
Average Number Of URLs	Average Number Of URLs in 'n' tweets

Table 3.1: Normalised Attributes

3.2. MACHINE LEARNING MODELS

The purpose of using machine learning is to train a network such that given the page attributes an authenticity score can be calculated/predicted by the network.

For this project we made use of the following algorithms:

- Backpropagation
- Naive Bayes
- Long short term memory (LSTM)
- K-Nearest Neighbours
- Decision Trees
- Linear Discriminant Analysis (LDA)

(i) BACKPROPAGATION

Backpropagation is a method used in artificial neural network to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network's layers. It is commonly used to train deep neural networks.

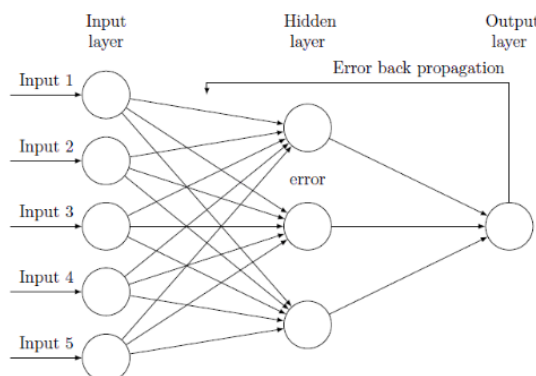
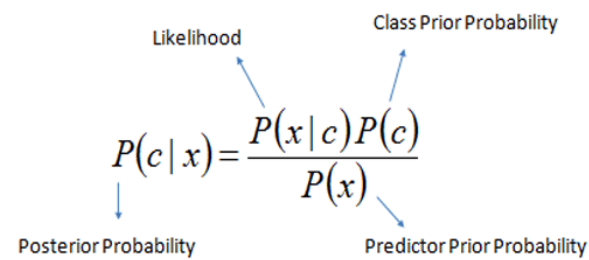


Fig.3.2: Backpropagation

(Source: https://www.researchgate.net/figure/Systematic-diagram-of-ANN-and-its-error-back-propagation-architecture-20_fig1_316091277)

(ii) NAÏVE BAYES

This algorithm works on Bayes theorem of probability to predict the class of unknown data set. In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.



The diagram shows the Bayes' Theorem formula $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ with four labels and arrows: 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$
$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Eqn 3.1: Bayes Theorem

(iii) LONG SHORT TERM MEMORY (LSTM)

Long Short Term Memory is an extension for recurrent neural networks. LSTM enable RNN's to remember their inputs over a long period of time. This is because LSTM's contain their information in a memory. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

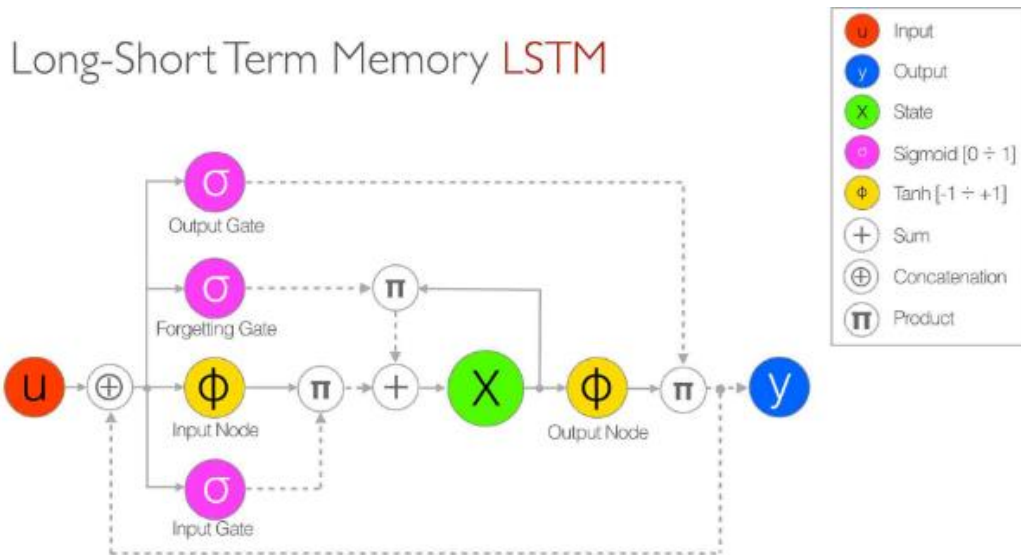


Fig.3.3: LSTM (Source: <https://www.add-for.com>)

(iv) K NEAREST NEIGHBOURS

In pattern recognition, the k-nearest neighbours algorithm (**k-NN**) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k -NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbour.
- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its k nearest neighbours.

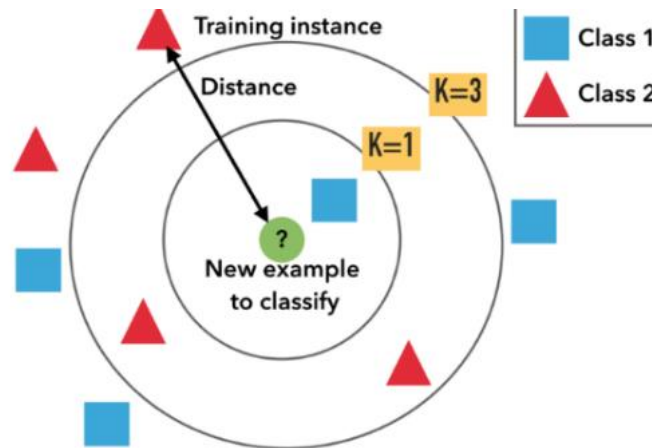


Fig.3.4: K-Nearest Neighbours (Source: https://cdn-images-1.medium.com/max/800/0*Sk18h9op6uK9EpT8)

(v) DECISION TREES

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

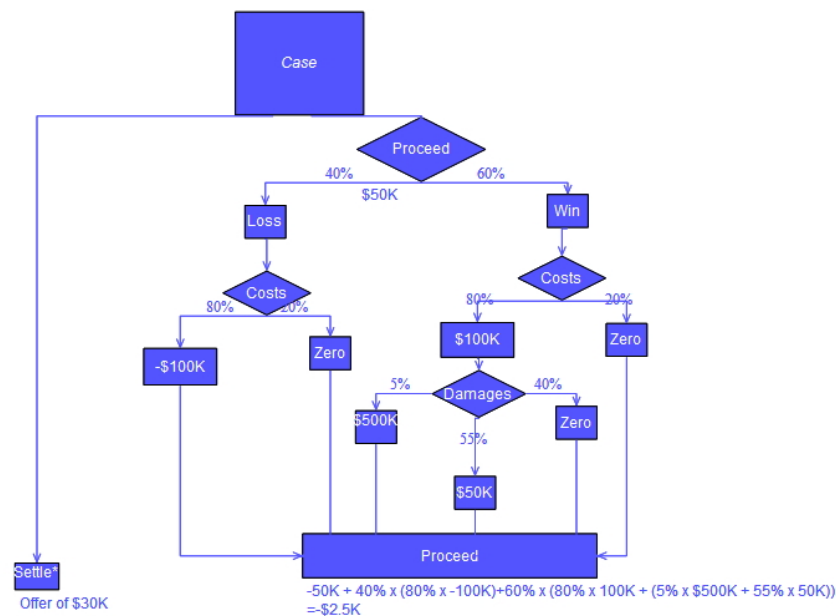


Fig.3.5: Decision Tree (Source: https://en.wikipedia.org/wiki/Decision_tree)

(vi) LINEAR DISCRIMINANT ANALYSIS

Linear discriminant analysis (LDA), normal discriminant analysis (NDA), or discriminant function analysis is a generalization of Fisher's linear discriminant, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.

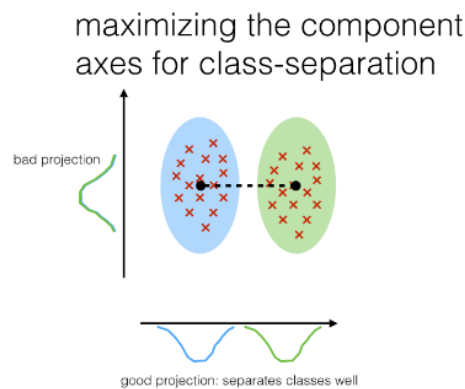


Fig.3.6: LDA

(Source: https://sebastianraschka.com/images/blog/2014/linear-discriminant-analysis/lda_1.png)

ABOUT KERAS

Keras is an open-source neural-network library written in Python. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras acts as a simple frontend interface to TensorFlow.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation

functions, optimizers, and a host of tools to make working with image and text data easier.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

ABOUT TENSORFLOW

TensorFlow is a free and open source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

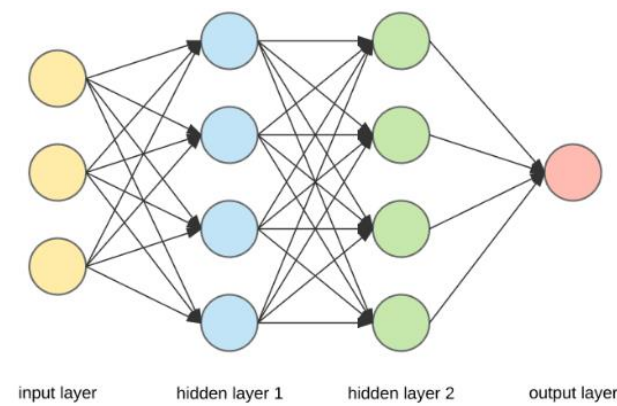


Fig.3.7: Simple Neural Network
(Source: https://en.wikipedia.org/wiki/Artificial_neural_network)

ALGORITHM	RESULT (BINARY/RANGE)	ACCURACY
Backpropagation	Range	96.83%
Naive Bayes	Binary	95.52%
LSTM	Range	96%
K-Nearest Neighbours	Binary	96.27%
Decision trees	Binary	95.37%
LDA	Binary	95.67%

Table 3.2: Comparison of Algorithms

Due to its highest accuracy and range output, we choose Backpropagation as the final algorithm.

3.3. Dataset

The model was trained with a dataset of almost 2500 instances containing various authentic and fake profiles, the respective attributes and a binary value result:

1 – authentic account

0 – fake account

During prediction, an authenticity score from range 0 to 1 was predicted and displayed as authenticity percentage to the user.

Authentic Instances – 1243

Fake Instances – 1243

3.4. Tweepy

Tweepy is a Python library for accessing Twitter API. It is an open-sourced library, hosted on GitHub and enables Python to communicate with Twitter Platform and uses its API. Tweepy supports accessing Twitter via Basic Authentication and the newer method, OAuth. Twitter has stopped accepting Basic Authentication so OAuth is now the only way to use the Twitter API. Main Model classes in the Twitter API are Tweets, Users, Entities and Places. Access to each returns a JSON-formatted response and traversing through information is very easy in Python.

OAuth:

Application-only authentication is a form of authentication where an application makes API requests on its own behalf, without the user context. This method is for developers that just need read-only to access public information.

Example: If a developer is trying to view Tweets or lists that are publically available, then they just have to use application-only authentication.

To use this method, one needs to use a bearer token. This bearer token can be generated by passing one's consumer key and secret through the POST `oauth2 / token` endpoint.

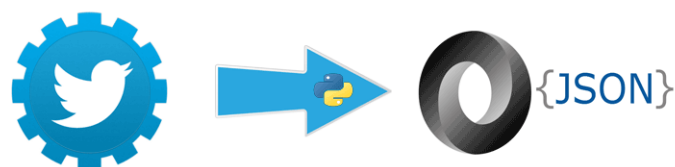


Fig.3.6: Tweepy Flow (Source: <https://nocodewebscraping.com/wp-content/uploads/2016/08/twitter-json-examples.png>)

3.5. User Interface

A web application was developed and deployed using Flask server. The purpose of using Flask was that it is a Python based Full stack Micro-framework.

Frontend - HTML, CSS, Chart.js

Backend - Python

In this application, the UI consists of 2 pages:

In the first page, user enters the URL of the Twitter account in a form and hits the submit button which is followed by the next page where the authenticity percentage is fetched from the prediction model and is displayed along with a table of stats namely:

- Followers Count
- Friends Count
- Average Likes
- Average Retweets

In addition to the above stats, a pie chart depicting followers-friends ratio is also displayed in the user interface of the web application. Chart.js was used for this purpose. It is an open source JavaScript library licensed by MIT. Chart.js can be used to produce many charts such as Pie, Line, Scatterplot etc.

CHAPTER 4

TECHNICAL IMPLEMENTATION

This chapter explains the algorithmic details of the project as well as the various functions and classes in the codebase.

4.1. Algorithm

The algorithm which was finalized for the purpose of this project was Backpropagation implemented in a neural network as it gave a range output and maximum accuracy.

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network's layers. It is commonly used to train deep neural networks.

Backpropagation is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. It is closely related to the Gauss–Newton algorithm and is part of continuing research in neural Backpropagation.

In the context of learning, Backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.

The motivation for Backpropagation is to train a multi-layered neural network such that it can learn the appropriate internal representations to allow it to learn any arbitrary mapping of input to output.

BACKPROPAGATION MATHEMATICAL CONCEPT

Consider a simple neural network with two input units, one output unit and no hidden units, and in which each neuron uses a linear output that is the weighted sum of its input.

Initially, before training, the weights will be set randomly. Then the neuron learns from training examples, which in this case consist of a set of tuples where x_1 and x_2 are the inputs to the network and t is the correct output (the output the network should produce given those inputs, when it has been trained). The initial network, given x_1 and x_2 , will compute an output y that likely differs from t (given random weights). A common method for measuring the discrepancy between the expected output t and the actual output y is the squared error measure:

$$E = (t - y)^2,$$

where E is the discrepancy or error.

Therefore, the problem of mapping inputs to outputs can be reduced to an optimization problem of finding a function that will produce the minimal error.

4.2. Program Code Description

4.2.1. Twitter Data Fetcher

This program uses Tweepy API to fetch the required attributes for the respective Twitter accounts.

The Tweepy API functions used are:

FUNCTION	ATTRIBUTE
<code>get_user(id)</code> id : Twitter ID of the account	<ul style="list-style-type: none">• Followers• Friends• Bio• Profile Pic
<code>search_users(name)</code> name : Display Name of the account	<ul style="list-style-type: none">• First Search Result
<code>user_timeline(id,count)</code> id : Twitter ID of the account count : Number of tweets to look up for	<ul style="list-style-type: none">• Retweeets• Likes• Average Number Of URLs• Retweet Fraction

Table 4.1: Tweepy Methods

4.2.2. Training Model

Keras sequential model was used to implement the neural network. The simplest model is defined in the Sequential class which is a linear stack of layers. One can create a Sequential model and define all of the layers in the constructor.

(i) Backpropagation

```
# create model
model = Sequential()
model.add(Dense(20, input_dim=11, init='normal', activation='relu'))
model.add(Dense(16, input_dim=11, init='normal', activation='relu'))
model.add(Dense(16, input_dim=11, init='normal', activation='relu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

The above code describes 4 neural network layers which takes 11 input parameters and 20 neuron units at the first layer followed by two layers of 16 units and final layer which gives 1 unit as output.

ACTIVATION FUNCTIONS

Sigmoid Activation function: It is a activation function of form $f(x) = 1 / 1 + \exp(-x)$. Its Range is between 0 and 1. It is an S—shaped curve.

ReLu- Rectified Linear units: It was recently proved that it had 6 times improvement in convergence from Tanh function. It's just $R(x) = \max(0, x)$ i.e. if $x < 0$, $R(x) = 0$ and if $x \geq 0$, $R(x) = x$.

KERNEL INITIALIZERS

The neural network needs to start with some weights and then iteratively update them to better values. Kernel_initializer is a term for which statistical distribution or function is used for initializing the weights. In

case of statistical distribution, the library will generate numbers from that statistical distribution and use as starting weights.

For example in the above code, normal distribution will be used to initialise weights. We can use other functions (constants like 1s or 0s) and distributions (uniform) too.

```
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Compile function configures the model for training. It takes the following arguments:

Optimizer: String (name of optimizer) or optimizer instance.

Loss: String (name of objective function) or objective function. If the model has multiple outputs, we can use a different loss on each output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses.

Metrics: List of metrics to be evaluated by the model during training and testing. Typically we will use `metrics=['accuracy']`. To specify different metrics for different outputs of a multi-output model, one could also pass a dictionary, such as `metrics={'output_a': 'accuracy'}`.

```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=150, batch_size=50)
```

Fit function trains the model for a given number of epochs (iterations on a dataset). It takes the following arguments:

- *x*: Numpy array of training data (if the model has a single input), or list of Numpy arrays (if the model has multiple inputs).
- *y*: Numpy array of target (label) data (if the model has a single output), or list of Numpy arrays (if the model has multiple outputs).
- *batch_size*: Integer or None. Number of samples per gradient update. If unspecified, *batch_size* will default to 32.
- *epochs*: Integer. Number of epochs to train the model. An epoch is an iteration over the entire *x* and *y* data provided. The model is not trained for a number of iterations given by epochs, but merely until the epoch of index *epochs* is reached.
- *validation_data*: tuple (*x_val*, *y_val*) or tuple (*x_val*, *y_val*, *val_sample_weights*) on which to evaluate the loss and any model metrics at the end of each epoch.

```
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

The model was saved in .h5 format to be used in the prediction file.

(ii) Long Short Term Memory (LSTM)

```
model = Sequential()
model.add(Embedding(max_features, output_dim=256))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

The above code uses LSTM algorithm to train the neural network. A sequence of vectors of dimension 128 is returned.

(iii) *Naive Bayes*

```
#Create a Gaussian Classifier  
model = GaussianNB()
```

Naive Bayes algorithm was implemented using the above function. The model was saved as a Pickle file so that it could be used in the prediction file.

(iv) *K-Nearest Neighbours*

```
# Spot Check Classifier  
model = KNeighborsClassifier()
```

K-Nearest Neighbours classifier was implemented using the above function. The model was dumped as a joblib file so that it could be used in the prediction file.

(v) *Decision Tree*

```
# Spot Check Classifier  
model = DecisionTreeClassifier()
```

Decision Tree classifier was implemented using the above function. The model was dumped as a joblib file so that it could be used in the prediction file.

(vi) *Linear Discriminant Analysis*

```
# Spot Check Classifier
model = LinearDiscriminantAnalysis()
```

Linear Discriminant Analysis classifier was implemented using the above function. The model was dumped as a joblib file so that it could be used in the prediction file.

4.2.3 Prediction Model

This program takes a Twitter ID as a parameter and fetches the page attributes from Twitter Data Fetcher (Ref section 4.2.1). The attributes are then normalised and passed to the predict() method of the loaded model (Ref section 4.2.2).

```
# NORMALIZATION OF ATTRIBUTES
normalized_followers = min((followers / max_followers), 1)
normalized_friends = min((friends / max_friends), 1)
if followers + friends != 0:
    followers_ratio = followers / (followers + friends)
    friends_ratio = friends / (followers + friends)
else:
    followers_ratio = 0
    friends_ratio = 0
normalized_avg_retweets = min((avg_retweets / max_avg_retweets), 1)
normalized_avg_likes = min((avg_likes / max_avg_likes), 1)
normalized_avg_no_urls = avg_no_urls / max_avg_no_urls
```

The above code performs normalization (Ref section 3.1.) of attributes.

```
# PREDICTION MODEL
X = array([[normalized_followers, normalized_friends, followers_ratio, friends_ratio, normalized_avg_retweets,
            normalized_avg_likes, bio, profile_pic, first_search_result, retweet_fraction, normalized_avg_no_urls]])
print(X)

# calculate prediction
prediction = loaded_model.predict(X)
```

The array of normalized attributes is passed to the inbuilt predict() method of loaded model.

4.2.4 Web Page

The app page consists of two routing methods:

- Routes to the homepage
- Routes to result page on submission of the form

These pages were implemented using HTML, CSS and JavaScript and hosted on a Flask Server.

```
@app.route('/', methods=['POST', 'GET'])
def result():
    if request.method == 'POST':
        user = request.form['url']
        id = user.split(".com/", 1)[1]
        (result, followers, friends, avg_retweets, avg_likes) = prediction_model.Predictor().predict(id, loaded_model)
        labels = ["Followers", "Followees"]
        values = [followers, friends]
        colors = ["#EAD1F3", "#F6B1C8"]
        return render_template('result.html', result=result*100, set=zip(values, labels, colors),
                               followers=followers, friends=friends, avg_retweets=avg_retweets, avg_likes=avg_likes,
                               id=id)
```

The user enters the URL of the Twitter account in the homepage form. On submission, the ID is passed to the prediction model (Ref section 4.2.3) which returns an authenticity percentage. This is then displayed on the result page.

CHAPTER 5

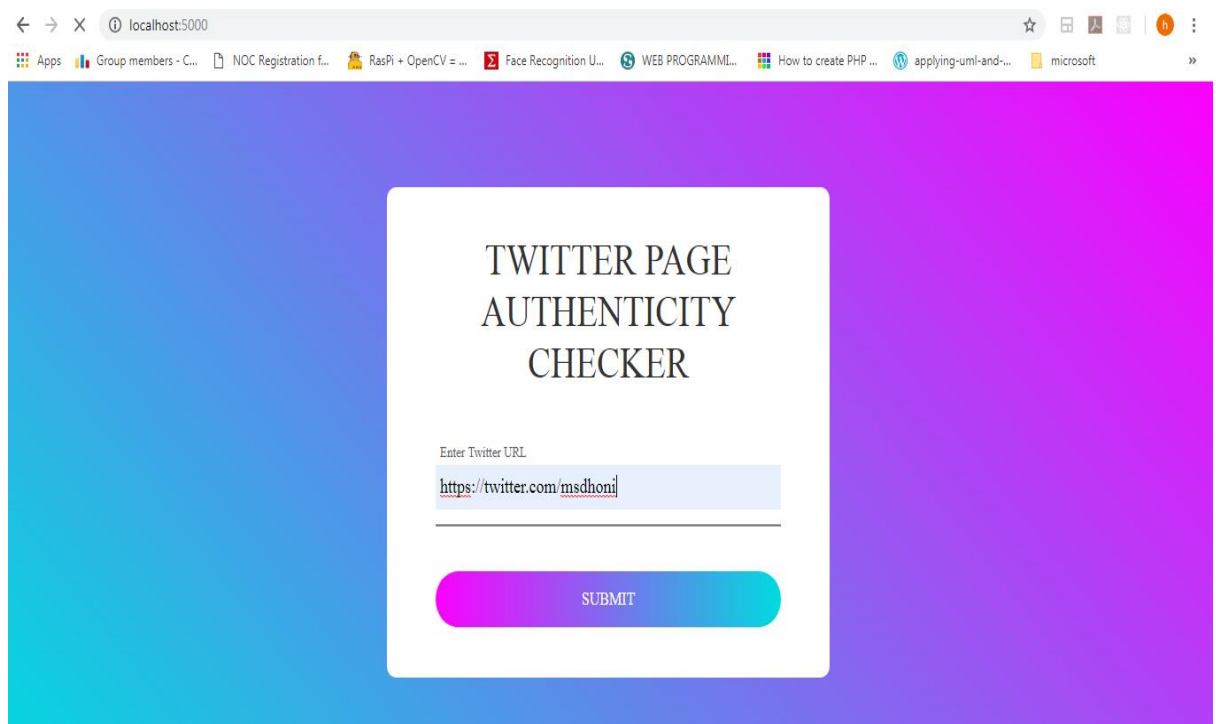
RESULTS

This chapter covers the results for 3 types of Twitter accounts, namely:

- Verified accounts
- Authentic accounts which are not verified
- Fake accounts

Additionally, it also covers the results of the training of the machine learning model

5.1. Verified accounts



The image shows a web browser window with the address bar displaying 'localhost:5000'. The browser's tab bar shows several open tabs, including 'Apps', 'Group members - C...', 'NOC Registration f...', 'RasPi + OpenCV = ...', 'Face Recognition U...', 'WEB PROGRAMMI...', 'How to create PHP ...', 'applying-uml-and-...', and 'microsoft'. The main content area of the browser displays a web application titled 'TWITTER PAGE AUTHENTICITY CHECKER'. The application has a white background with a blue-to-purple gradient border. It features a text input field labeled 'Enter Twitter URL' containing the text 'https://twitter.com/msdhoni'. Below the input field is a blue button with the text 'SUBMIT'.

Fig.5.1.1: Verified Account Form

User enters a *verified* Twitter account in the homepage form.

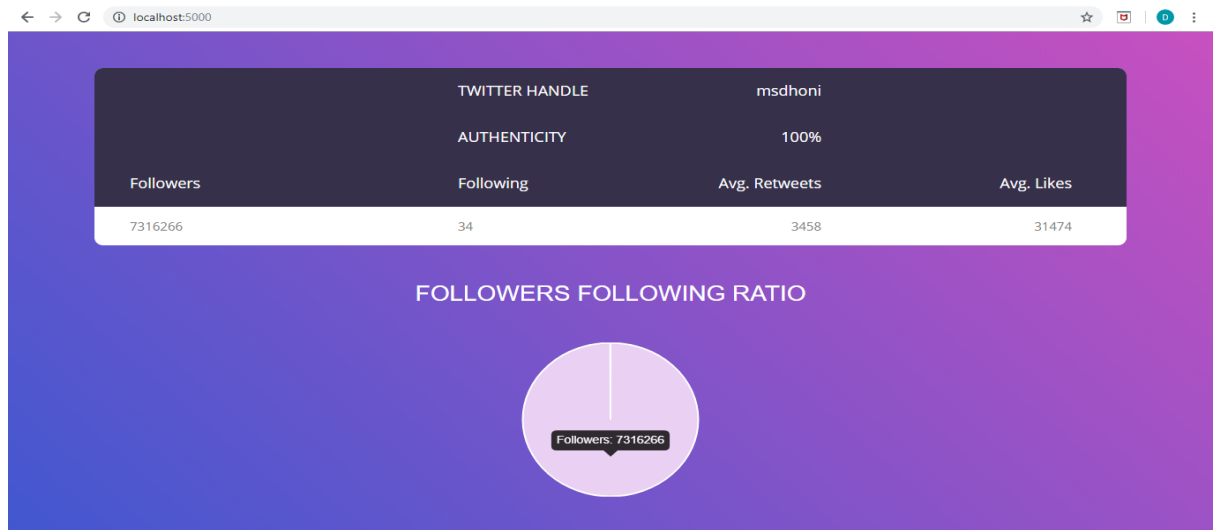


Fig.5.1.2: Verified Account Result

The user is redirected to the result page after hitting the submit button and the authenticity percentage along with stats are displayed.

5.2. Authentic, non-verified accounts

TWITTER PAGE AUTHENTICITY CHECKER

Enter Twitter URL

<https://twitter.com/raizawilson>

SUBMIT

Fig.5.2.1: Authentic, Non-Verified Account Form

User enters *authentic, non-verified* Twitter account in the homepage form.

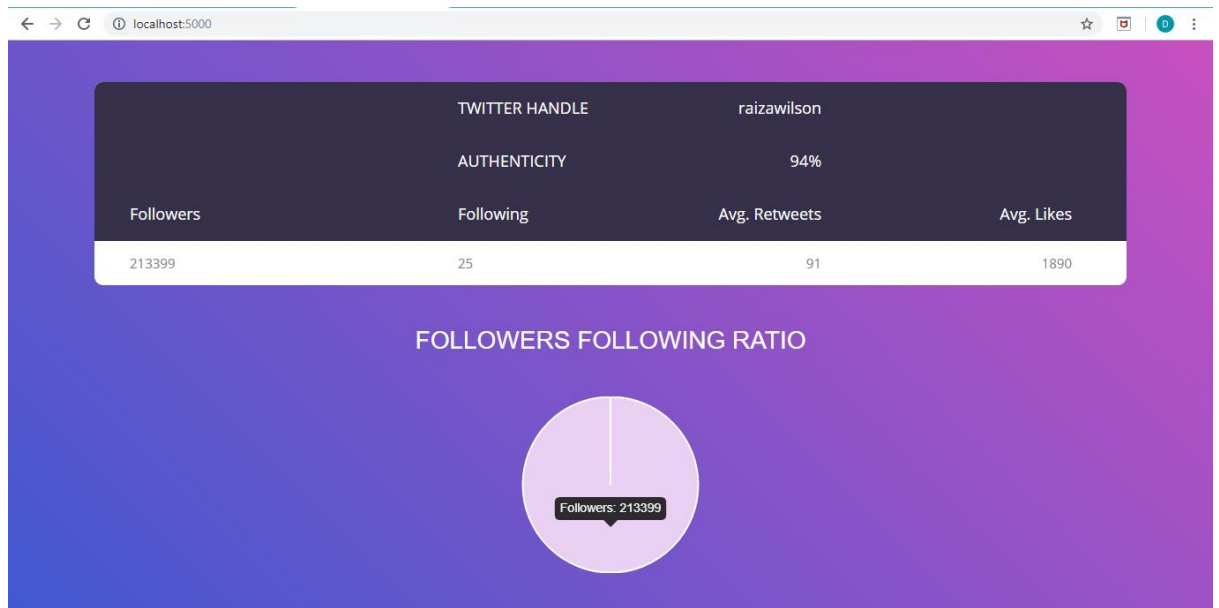


Fig.5.2.2: Authentic, Non-Verified Account Result

The user is redirected to the result page after hitting the submit button and the authenticity percentage along with stats are displayed.

5.3. Fake accounts

The screenshot shows a web browser window with the URL 'localhost:5000'. The page displays the 'TWITTER PAGE AUTHENTICITY CHECKER' form. The form is centered on a white background with a blue and purple gradient. It includes a text input field for the Twitter URL, a 'SUBMIT' button, and a 'Enter Twitter URL' label. The URL entered is 'https://twitter.com/SureshRaina__'.

Fig.5.3.1: Fake Account Form

User enters *fake* Twitter account in the homepage form.

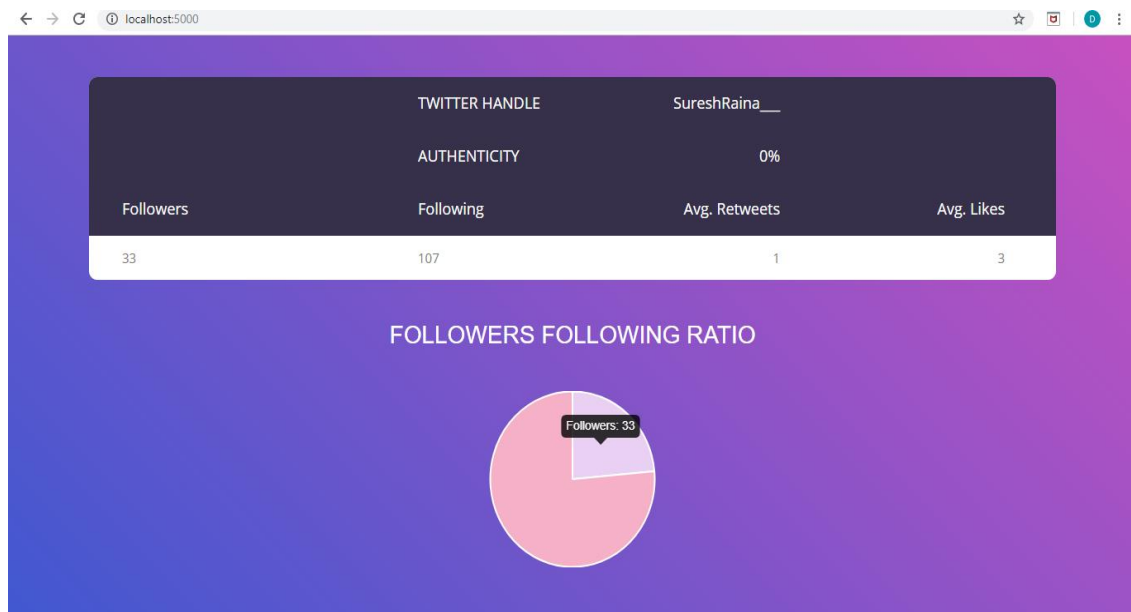


Fig.5.3.2: Fake Account Result

The user is redirected to the result page after hitting the submit button and the authenticity percentage along with stats are displayed.

5.4. Training Epochs

```

* 50/1988 [.....] - ETA: 0s - loss: 0.0307 - acc: 1.000
1950/1988 [=====] - ETA: 0s - loss: 0.1211 - acc: 0.966
1988/1988 [=====] - 0s 30us/step - loss: 0.1210 - acc:
0.9668 - val_loss: 0.1000 - val_acc: 0.9759
Epoch 147/150
* 50/1988 [.....] - ETA: 0s - loss: 0.1566 - acc: 0.940
1988/1988 [=====] - 0s 25us/step - loss: 0.1204 - acc:
0.9673 - val_loss: 0.0989 - val_acc: 0.9739
Epoch 148/150
* 50/1988 [.....] - ETA: 0s - loss: 0.2142 - acc: 0.940
1988/1988 [=====] - 0s 28us/step - loss: 0.1209 - acc:
0.9673 - val_loss: 0.0988 - val_acc: 0.9739
Epoch 149/150
* 50/1988 [.....] - ETA: 0s - loss: 0.0816 - acc: 0.980
1988/1988 [=====] - 0s 24us/step - loss: 0.1207 - acc:
0.9678 - val_loss: 0.0987 - val_acc: 0.9739
Epoch 150/150
* 50/1988 [.....] - ETA: 0s - loss: 0.0771 - acc: 0.980
1988/1988 [=====] - 0s 25us/step - loss: 0.1206 - acc:
0.9668 - val_loss: 0.0985 - val_acc: 0.9739
Saved model to disk

```

Fig.5.4: Training Epochs

CHAPTER 6

CONCLUSION

For the purpose of this project, a set of 4 publications, the problem statement of which were similar to ours, were analysed from which we were inspired to add 11 attributes. The machine learning algorithms to use were also decided upon from this survey.

Thus, the attributes and characteristics of authentic and fake Twitter pages were studied and various machine learning algorithms were explored. We observed that for this use-case the Backpropagation algorithm was found to have maximum accuracy and was able to perform authenticity predictions accurately to a large extent. This use-case was developed and deployed as a web-application using Flask.

The future scope of this project extends to including more number of attributes, studying many other machine learning algorithms and improved efficiency of the User Interface without any lag in the routing.

REFERENCES

1. Barbara Poblete, Carlos Castillo, Marcelo Mendoza (2011), 'Information Credibility on Twitter', WWW 2011 – Session: Information Credibility, Hyderabad, India
http://chato.cl/papers/castillo_mendoza_poblete_2010_Twitter_credibility.pdf
2. Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, Virgílio Almeida, 'Detecting Spammers on Twitter', Computer Science Department, Universidade Federal de Minas Gerais Belo Horizonte, Brazil
<https://homepages.dcc.ufmg.br/~fabricio/download/ceas10.pdf>
3. Supraja Gurajala, Joshua S White, Brian Hudson, Brian R Voter, Jeanna N Matthews (2016), 'Profile characteristics of fake Twitter accounts', Big Data & Society
https://www.researchgate.net/publication/3093003063_Profile_characteristics_of_fake_Twitter_accounts
4. Ahmed El Azab, Amira M. Idrees, Mahmoud A. Mahmoud, Hesham Hefny (2016), '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, No:1, 2016
<https://waset.org/publications/10003176/fake-account-detection-in-Twitter-based-on-minimum-weighted-feature-set>
5. <https://machinelearningmastery.com/tutorial-first-neural-network-python-Keras>
6. <https://www.analyticsvidhya.com/blog/2017/09/machine-learning-models-as-apis-using-flask/>
7. <https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/>

8. <https://en.wikipedia.org/wiki/Backpropagation>
9. <https://en.wikipedia.org/wiki/Keras>
10. <https://colorlib.com/wp/css3-table-templates/>
11. <https://chartjs.org>
12. <https://Tweepy.readthedocs.io>
13. <https://Keras.io/models/about-Keras-models/>