Password Strength Analysis Using Machine/Deep Learning Models

By

Tripurari Kumar Jha [19BEC1447]

Project Report Submitted

to

Dr. KIRUTHIKA V

SCHOOL OF ELECTRONICS ENGINEERING

in partial fulfilment of the requirements for the course of

ECE3009 – Neural Networks and Fuzzy Control

in

B. Tech. ELECTRONICS AND COMMUNICATION ENGINEERING

Vandalur – Kelambakkam Road

Chennai – 600127

JULY 2022

CERTIFICATE

This is to certify that the Project work entitled "**Password Strength Analysis Using Machine/Deep Learning Models**" that is being submitted by Ayush Singh [19BEC1032], Priya Ranjan [19BEC1349] and Tripurari Kumar Jha [19BEC1447] who carried out the Project work under my supervision and guidance for ECE3009-Neural Networks and Fuzzy Control.

Dr. KIRUTHIKA V

Assistant Professor Senior Grade,

School of Electronics Engineering (SENSE),

VIT University, Chennai

Chennai – 600 127.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Kiruthika V,** Assistant Professor Senior Grade, School of Electronics Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Susan Elias,** Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Head of the Department **Dr. Mohanaprasad** K for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

NAME WITH SIGNATURE   **NAME WITH SIGNATURE**

Tripurari Kumar Jha

Table of Contents

Problem Statement:

Idea

User authentication is an important way to ensure the security of your cyber account. Although there are various authentication methods such as iris and fingerprint, passwords will be the main authentication method for the time being due to their low cost and ease of implementation. The password strength score is used to measure password strength. This has been studied extensively.

However, of the current password strength evaluation methods, ignores the characteristics of password writers and does not consider a wide range of options to consider before providing an evaluation report. In order to overcome this issue we intend to assign a score depicting the strength of passwords rather than just commenting on its strength.

Password cracking is the process of retrieving passwords from a computer or data transmitted by a computer. This need not be a sophisticated procedure. Password cracking is a brute-force assault that checks all potential combinations. If the password is stored in plaintext, a hacker can access all account information by hacking the database. Most passwords, however, are now saved with a key derivation function. This creates a 'hash' by running a password through a one-way cypher. The hash of the password is stored in the server. Some most popular Password Cracking Techniques are:

- **Brute Force Attack:**
  Password crackers have a brute force attack as a final resort if everything else fails. It essentially entails attempting all feasible combinations until you strike gold. Password cracking tools, on the other hand, allows users to modify the attack and cut down on the time it takes to examine all possible permutations.
- **Dictionary Attack:**
  A dictionary attack is a sort of brute force attack that is frequently combined with other types of brute force attacks. It immediately scans the dictionary to see whether the password isn't a common phrase. Passwords from other hacked accounts could likewise be added by the attacker.
- **Spidering:**
  Spidering is a password cracking technique that can be used in conjunction with the above-mentioned brute force and dictionary attacks. The idea is to make a word list that will aid in guessing the password more quickly.

- **Guessing:**
  While guessing isn't the most prevalent password cracking method, it does relate to the business-oriented spidering discussed earlier. It's not always necessary for the attacker to obtain information on the victim because trying some of the most frequent passwords is often enough.
- **Rainbow Table Attack:**
  Experienced hackers usually have a rainbow table with leaked and already cracked passwords, which makes it more effective. Rainbow tables frequently contain all potential passwords, causing them to be extraordinarily large, occupying hundreds of GBs.

Some of the popular tools which use the above-mentioned techniques are:

- **John the Ripper**:
  John the Ripper is a command-based application which is free and open-source. It's available for Linux and macOS, whereas Hash Suite, built by a contributor, is available for Windows and Android.
- **Cain and Abel**:
  Another famous password cracking tool is Cain & Abel. But, unlike John the Ripper, it has a graphical user interface, making it instantly more user-friendly.
- **Ophcrack:**
  Rainbow table attacks are the specialty of Ophcrack, a free and open-source password cracking program. It cracks LM and NTLM hashes, the former for Windows XP and previous OSs, and the latter for Windows Vista and 7.
- **THC Hydra:**
  THC Hydra's tactics include brute force and dictionary attacks, as well as the use of wordlists supplied by other tools. Because of the multi-threaded combination testing, this password cracker is noted for its speed.
- **Hashcat:**
  Hashcat, claimed as the world's fastest password cracker, is a free open-source software that runs on Windows, Mac OS X, and Linux. It includes a variety of approaches, ranging from simple brute force to a hybrid mask with wordlist.

With these methods and tools available, passwords are always at a risk and there is a need for more stronger passwords.

Scope

Passwords provide the first line of defense against unauthorized access to any computer or security system. The stronger is the password, the more protected/secured a system will be from hackers and malicious software. Although there are many alternatives to passwords for access control, passwords are the most

convincing credentials in many applications. They provide a simple and straight forward way to protect a system, and they represent an individual's identity and authentication to a system.

The main weakness of passwords is their nature. Many times, these days, strong passwords are essential to protect personal data, as there are many ways for unauthorized people with little technical knowledge or skills to learn passwords for legitimate users. Therefore, it is important for organizations to understand the vulnerabilities to which passwords are exposed and to set strict guidelines on how to create and use passwords to prevent these vulnerabilities from being exploited.

Novelty

A password strength analysis system shows how resistant a given password might be to password cracking attempts like brute force and dictionary attacks.

- It can be used to check and strengthen the password.
- It can be used in various websites like Facebook, Instagram, Gmail, etc.
- It can be used to compare the password strength predicted by machine learning and password strength based on rules.

Our work is very unique as:

- We help the user to have a strong password which will help him/her to protect from vulnerability.
- Our project will help the user to analyze his/her password.
- Find out the best model for password evaluation.
- Comparative analysis and advantages over existing password evaluation technologies.
- As our project provides both methods, i.e., machine learning as well as standard rules for rating passwords, it can be used to compare both of these methods.
- Our project also tells the user whether his/her password has been breached or not by checking against a huge database of 7 billion+ exposed passwords. By using this feature the user can get to know whether his/her password has been breached or not and thus immediately change it if breached.

Related Works

Monte Carlo Strength Evaluation: Fast and Reliable Password Checking- Matteo Dell'Amico and Maurizio Filippone. 2015 [1]

This paper proposed a unique approach to estimate the wide variety of guesses had to discover a password the usage of cutting-edge assaults. Their approach calls for little resources, applies to a huge set of probabilistic models, and is characterized via way of means of rather suited convergence properties. The experiments display the scalability and generality of the proposal. In particular, the experimental evaluation reviews opinions on a huge variety of password strengths, and of ultra-modern assaults on very huge datasets, consisting of assaults that might were prohibitively steeply-priced to address with current simulation-primarily based totally approaches.

Design and Evaluation of a Data-Driven Password Meter- Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. 2017 [2]

This paper analyzed password meters deployed in selected popular websites and password managers. It documented obfuscated source available meters, inferred the algorithm behind the closed source ones, and measured the strength labels assigned to common passwords from several password dictionaries. From this empirical analysis with millions of passwords, it sheds light on how the server end of some web service meters functions and provides examples of highly inconsistent strength outcomes for the same password in different meters, along with examples of many weak passwords being labeled as strong or even excellent. These weaknesses and inconsistencies may confuse users in choosing a stronger password, and thus may weaken the purpose of these meters.

A Password Meter without Password Exposure- Kim, P.; Lee, Y.; Hong, Y.-S.; Kwon, T. 2021 [3]

This paper explores a brand-new online password meter idea that doesn't necessitate the publicity of user`s passwords for comparing user-selected password applicants at the server side. The concept is to evolve completely homomorphic encryption (FHE) schemes to construct one of these gadgets however its overall performance fulfillment is significantly challenging. The paper employs diverse overall performance enhancement strategies and

implements the NIST (National Institute of Standards and Technology) metering method. Thus, it offers an excessive stage of privateness to the passwords being tested. The reality that the password test is done consistent with the identical predefined technique for all encrypted inputs can save you side-channel assaults of the server, which may be made through measuring the execution time of the password meter.

Better passwords through science (and neural networks)- William Melicher, Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2017 [4]

This article describes how to use neural networks to accurately measure password strength and how to use this feature to build effective password counters. This article will first show you how to use a neural network to infer passwords, and how to use this method to create password inferences to better model inference attacks. Finally, the author of the paper describes how to design and build password counters based on neural networks. This provides users with more accurate and helpful guidance on how to create passwords that are resistant to guessing attacks.

Password Strength Prediction Using Supervised Machine Learning Techniques- Vijaya M.S, Jamuna K.S, Karpagavalli S., Christina V. 2010 [5]

In this paper, the password energy prediction become modeled as a class venture and supervised gadget mastering strategies had been employed. Widely used supervised gadget mastering strategies particularly C 4.5 Decision tree classifier, Multilayer Perceptron, Naïve Bayes Classifier and Support Vector Machine had been used for mastering the model. The effects of the fashions had been as compared and determined that SVM plays well. The effects of the fashions had been additionally as compared with the prevailing password energy checking tools. The findings display that the gadget mastering technique has sizable functionality to categories the acute cases – Strong and vulnerable passwords.

A New Multimodal Approach for Password Strength Estimation— Part I: Theory and Algorithms - J. Galbally, I. Coisel and I. Sanchez. 2017 [6]

This paper proposes a new multimodal intensity metric that combines several incomplete individual metrics to take advantage of their strengths and overcome many of their weaknesses. The final multimodal metric consists of various modules based on both heuristics and statistics, and when merged, provides real-time, realistic and reliable feedback on password "recognizability". On the one hand, service providers can be used as a security control to force the adoption of strong passwords by preventing users from choosing weak passwords, or at least integrating them into the password selection process so that warnings are displayed. End users, on the other hand, receive real-time feedback on the potential for password compromise in the event of an attack or breach.

A New Multimodal Approach for Password Strength Estimation— Part II: Experimental Evaluation - J. Galbally, I. Coisel and I. Sanchez. 2017 [7]

This paper was an extension of the paper and was the successor to the previous paper. This suggests an experimental framework used to evaluate the new approach above. Experimental evaluations are performed not only on new measuring devices, but also on other strength estimators from prior art to compare their overall performance. In addition to consistent results, the proposed method is extremely flexible and can be adapted to a particular environment or a particular password policy. In addition, it can evolve over time to naturally adapt to the new password selection trends that users are following.

Real Time Password Strength Analysis on a Web Application Using Multiple Machine Learning Approaches- Umar Farooq. 2020 [8]

The model proposed in the article provides a common and effective way to protect against these online and offline attacks by forcing the user to choose a strong password by implementing several algorithms. machine learning like Decision Trees (DT), Naive Bayes (NB), Linear Regression. (LR), Random Forest (RF) and Neural Networks (NN) on a real-time web application. Using Burp Suite software, three types of password cracking attacks, such as brute force attack, dictionary attack and reverse brute force attack, were performed on the Web application. They created 250 accounts, of which 150 had strong passwords and another 100 had weak passwords. Strong passwords were set for 150 accounts that were not cracked by any of these three attack types while 86 out of 100 weak passwords were cracked.

Dataset

The passwords used in the dataset are from the internet leak of 000webhost available on Kaggle[9]. Georgia Tech University has developed a program called PARS that integrates all commercial password metres. For each commercial password strength metre, the passwords were given to the tool, which produced fresh files. The passwords were stored in files with an additional column indicating their strength as determined by commercial password strength metres. The commercial password strength algorithms used are of Twitter, Microsoft and battle. Rather than using rules, this strength algorithm relies solely on machine learning. Only passwords that were marked as weak, medium, or strong by all three strength metres were kept. This indicates that all of the passwords were weak, medium, or strong in nature.

It had a total of 3 million passwords, but only 0.7 million passwords remained after intersection of all commercial metre classifications. Only passwords that were highlighted in a certain category by all three algorithms were used, which resulted in the reduction. Some of the key features of this data set are:

- o Password - 6701k unique values for password collected online
- o Strength - Three values (0, 1, 2), i.e. 0 for weak, 1 for medium and 2 for strong
- o Strength of the password is based on rules (Containing digits, special characters, etc.)

Test Bed

We conducted our project in a python 3.9 virtual environment. We used Python command line arguments to create the environment as it supports most of the skiKit libraries which are required in construction of the ML joblib files to be used by the web application. We used the sklearn library to get the models and the Flask container to construct the Web application. Additionally, we also used Enzoic's

Live database which consisted of various compromised credentials using the API keys provided by the website to check for the occurrence of the passwords in public domains.

Expected Result

Our project is aimed at performing Password Strength Analysis using Machine Learning Algorithms. We will be providing an interface to the user to check his password strength. Lastly, performing a comparative analysis of the results of all the algorithms.

Thus, in the end an additional result will be displayed which would tell the user whether his/her password was breached or not by checking against a huge database of seven billion+ compromised passwords. It will also provide different ratings for passwords based on various parameters such as lowercase letters, uppercase letters, numbers etc.

Architecture

High Level Design
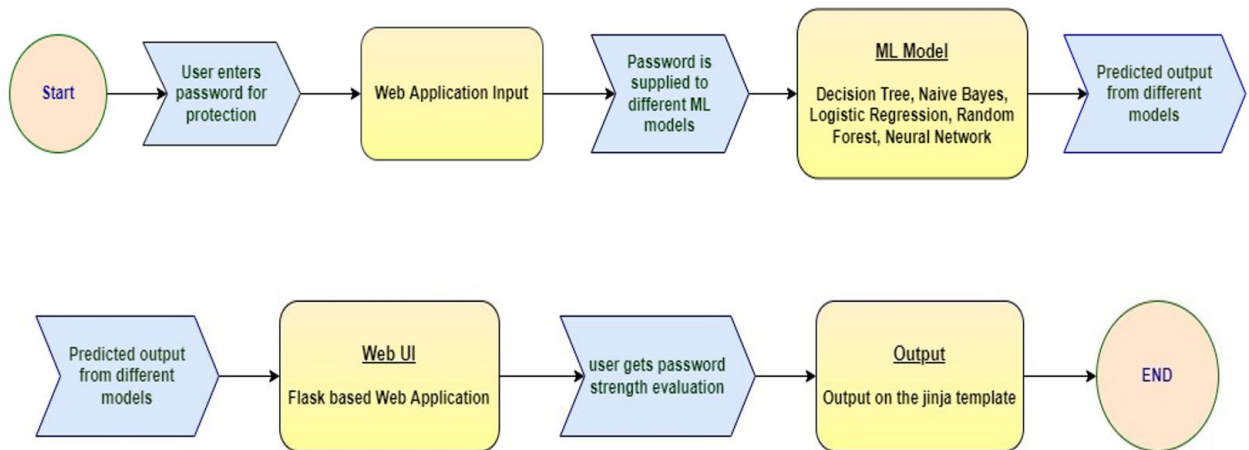
The Flow diagram included above explains the working of our proposed architecture where the work of each of the component is described:

- User enters password for evaluation in the Web Application.
- Password is supplied to the ML models-
  - Decision Tree
  - Naïve Bayes
  - Logistic Regression
  - Neural Network.
- The output is predicted from different models and sent to the Flask based Application.
- Output is presented on the Web UI and the user gets the Password Strength Analysis.

Low Level Design

Algorithms used:

- o Naïve Bayes
- o Logistics Regression
- o Decision Tree
- o Neural Networks

All the above-mentioned algorithms proceed with their training in a similar fashion following the given flow diagram:

Following the above-mentioned flow diagram each of the ML algorithms proceed to their training in a similar fashion.

Logistics Regression

- Data Pre-processing step: Since the dataset had no missing values, no pre-processing was required.

- Fitting Logistic Regression to the Training set: The model is trained on the complete dataset and its result was recorded for future comparisons
- Evaluating the test result: The model was evaluated with test data obtained by splitting the original data.
- Test accuracy of the result(Creation of Confusion matrix)
- Parameter Tuning: Since sklearn libraries were used, the data was directly sent to the LR classifier model provided by the libraries.
- Predictions: Finally, the model was deployed on the web and predictions were made.

Decision Tree

- Data Pre-processing step: Since the dataset had no missing values, no pre-processing was required.
- Fitting Decision Tree to the Training set: The model is trained on the complete dataset and its result was recorded for future comparisons
- Evaluating the test result: The model was evaluated with test data obtained by splitting the original data.
- Test accuracy of the result(Creation of Confusion matrix)
- Parameter Tuning: Since sklearn libraries were used, the data was directly sent to the Decision Tree classifier model provided by the libraries.
- Predictions: Finally, the model was deployed on the web and predictions were made.

Naïve Bayes

- Data Pre-processing step: Since the dataset had no missing values, no pre-processing was required.
- Fitting Naïve Bayes to the Training set: The model is trained on the complete dataset and its result was recorded for future comparisons
- Evaluating the test result: The model was evaluated with test data obtained by splitting the original data.
- Test accuracy of the result(Creation of Confusion matrix)

- Parameter Tuning: Since sklearn libraries were used, the data was directly sent to the Naïve Bayes classifier model provided by the libraries.
- Predictions: Finally, the model was deployed on the web and predictions were made.

Neural Networks

- Data Pre-processing step: Since the dataset had no missing values, no pre-processing was required.
- Fitting Applied Neural Networks to the Training set: The model is trained on the complete dataset and its result was recorded for future comparisons
- Evaluating the test result: The model was evaluated with test data obtained by splitting the original data.
- Test accuracy of the result(Creation of Confusion matrix)
- Parameter Tuning: Since sklearn libraries were used, the data was directly sent to the Applied Neural Networks classifier model provided by the libraries.
- Predictions: Finally, the model was deployed on the web and predictions were made.

Implementation:

Algorithm Proposed
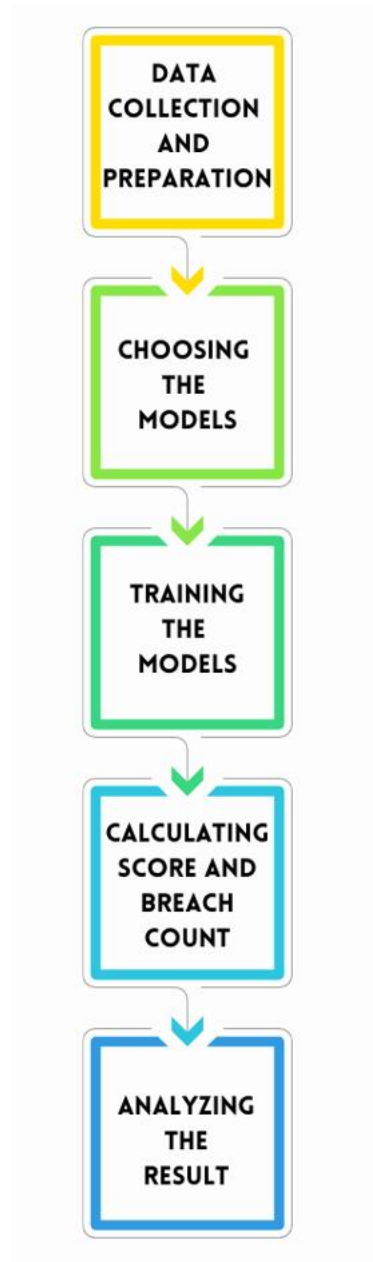
For evaluating the strength of user passwords, the above workflow is devised. The different processes mentioned in the algorithms are explained as follows:

- **Data Collection and preparation:**
  The dataset contains 669639 passwords. It contains three attributes: index, password and strength. This data set we obtained from Kaggle is already in prepared state with no NULL values and can be implemented directly for training and testing the models.

- **Choosing the ML Models:**
  Next, we identify the different Machine learning Algorithms which we will use to evaluate the strength of the passwords. This selection can be done as per the requirement specified. ML Algorithms used for this scenario are Neural Networks, Decision Tree, Logistics Regression and Naïve Bayes.

- **Training the ML Models:**
  The training of the models is done using the sklearn libraries provided by sciKit to easily train the models. Scikit-learn provides us with a bundle of machine learning algorithms from various domains such as supervised, unsupervised and semi-supervised. It features various classification, regression and clustering algorithms such as Random Forest, Decision Trees, Naive Bayes, etc

- **Predicting Score and Breach Count:**
  The input password is checked for its strength(Score) which is calculated on different rules such as
    1. Its length
    2. Consecutive upper-case characters
    3. Consecutive lower-case characters
    4. It contains numerical
    5. It contains special characters

  Further the password hash will be checked in a large database (enzoic api) containing 7 billion of "not secure" password hashes and the breach count (like how many times in the past has the input password been breached). By using this feature the user can get to know whether his/her password has been breached or not and thus immediately change it if breached.

- **Evaluating the Strength:**
  All the results are consolidated and displayed on the Web application which is developed using Flask container as it is based on Python programming Language which is used to train the ML models. The result of the different ML models commenting on the Passwords strength including the breach count are displayed on the web application.

Result Analysis:

Python Scripts

Naive Bayes script (*NaiveBayes.py*)

```python
# Naive Bayes
import pandas as pd
import time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline

# Classifier Model (Naive Bayes)
from sklearn.naive_bayes import BernoulliNB
import joblib

start_time = time.time()
data = pd.read_csv('data.csv')
features = data.values[:, 1].astype('str')
labels = data.values[:, -1].astype('int')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,
y_test=train_test_split(features,labels,test_size=0.25,random_state=
0)
classifier_model = Pipeline([
            ('tfidf', TfidfVectorizer(analyzer='char')),
            ('bernoulliNB',BernoulliNB()),
])

classifier_model.fit(X_train, y_train)
y_pred = classifier_model.predict(X_test)

from sklearn.metrics import confusion_matrix, classification_report
cm=confusion_matrix(y_test,y_pred)
print(classification_report(y_test, y_pred, digits=4))
print("Confusion Matrix: \n", cm)
accuracy =
(cm[0][0]+cm[1][1]+cm[2][2])/(cm[0][0]+cm[0][1]+cm[0][2]+cm[1][0]+cm
[1][1]+cm[1][2]+cm[2][0]+cm[2][1]+cm[2][2])
print('Training Accuracy: ',classifier_model.score(features, labels))
```

```
print("Testing Accuracy = ", accuracy)
print("Time Taken to train the model = %s seconds" %
round((time.time() - start_time),2))
# Save model
joblib.dump(classifier_model, 'NaiveBayes_Model.joblib')
```

Logistics Regression script (*LogisticsRegression.py*)

```
# Logistic Regression
import pandas as pd
import time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline

# Classifier Model (Logistic Regression)
from sklearn.linear_model import LogisticRegression
import joblib

start_time = time.time()
data = pd.read_csv('data.csv')
features = data.values[:, 1].astype('str')
labels = data.values[:, -1].astype('int')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,
y_test=train_test_split(features,labels,test_size=0.25,random_state=
0)
classifier_model = Pipeline([
              ('tfidf', TfidfVectorizer(analyzer='char')),
              ('logisticRegression',LogisticRegression(multi_class
='multinomial', solver='sag')),
])
# Fit the Model
classifier_model.fit(X_train, y_train)
y_pred = classifier_model.predict(X_test)

from sklearn.metrics import confusion_matrix, classification_report
cm=confusion_matrix(y_test,y_pred)
print(classification_report(y_test, y_pred, digits=4))
print("Confusion Matrix: \n", cm)
```

```python
accuracy =
(cm[0][0]+cm[1][1]+cm[2][2])/(cm[0][0]+cm[0][1]+cm[0][2]+cm[1][0]+cm
[1][1]+cm[1][2]+cm[2][0]+cm[2][1]+cm[2][2])
print('Training Accuracy: ',classifier_model.score(features, labels))
print("Testing Accuracy = ", accuracy)
print("Time Taken to train the model = %s seconds" %
round((time.time() - start_time),2))
# Save model
joblib.dump(classifier_model, 'LogisticRegression_Model.joblib')
```

Neural Network script (NeuralNetwork.*py*)

```python
# Applied Neural Network
import pandas as pd
import time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline

# Classifier Model (MultiLayer Perceptron)
from sklearn.neural_network import MLPClassifier
import joblib

start_time = time.time()
data = pd.read_csv('data.csv')
features = data.values[:, 1].astype('str')
labels = data.values[:, -1].astype('int')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,
y_test=train_test_split(features,labels,test_size=0.25,random_state=
0)

# Sequentially apply a list of transforms and a final estimator
classifier_model = Pipeline([
                ('tfidf', TfidfVectorizer(analyzer='char')),
                ('mlpClassifier', MLPClassifier(solver='adam',
                                         alpha=1e-5,
                                         max_iter=400,
```

```python
                                        activation='logistic
')),
])

# Fit the Model
classifier_model.fit(X_train, y_train)
y_pred = classifier_model.predict(X_test)

from sklearn.metrics import confusion_matrix, classification_report
cm=confusion_matrix(y_test,y_pred)
print(classification_report(y_test, y_pred, digits=4))
print("Confusion Matrix: \n", cm)
accuracy =
(cm[0][0]+cm[1][1]+cm[2][2])/(cm[0][0]+cm[0][1]+cm[0][2]+cm[1][0]+cm
[1][1]+cm[1][2]+cm[2][0]+cm[2][1]+cm[2][2])
print('Training Accuracy: ',classifier_model.score(features, labels))
print("Testing Accuracy = ", accuracy)
print("Time Taken to train the model = %s seconds" %
round((time.time() - start_time),2))
# Save model
joblib.dump(classifier_model, 'NeuralNetwork_Model.joblib')
```

Decision Tree script (DecisionTree.*py*)

```python
# Decision Tree Classifier
import pandas as pd
import time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier

# To save the trained model on local storage
import joblib

start_time = time.time()
data = pd.read_csv('data.csv')
features = data.values[:, 1].astype('str')
labels = data.values[:, -1].astype('int')
```

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,
y_test=train_test_split(features,labels,test_size=0.25,random_state=
0)
classifier_model = Pipeline([
                ('tfidf', TfidfVectorizer(analyzer='char')),
                ('decisionTree',DecisionTreeClassifier()),
])

# Fit the Model
classifier_model.fit(X_train, y_train)
y_pred = classifier_model.predict(X_test)

from sklearn.metrics import confusion_matrix, classification_report
cm=confusion_matrix(y_test,y_pred)
print(classification_report(y_test, y_pred, digits=4))
print("Confusion Matrix: \n", cm)
accuracy =
(cm[0][0]+cm[1][1]+cm[2][2])/(cm[0][0]+cm[0][1]+cm[0][2]+cm[1][0]+cm
[1][1]+cm[1][2]+cm[2][0]+cm[2][1]+cm[2][2])
print('Training Accuracy: ',classifier_model.score(features, labels))
print("Testing Accuracy = ", accuracy)
print("Time Taken to train the model = %s seconds" %
round((time.time() - start_time),2))
# Save model
joblib.dump(classifier_model, 'DecisionTree_Model.joblib')
```

Web App script (app.*py*)

```python
from flask import Flask, render_template, flash, request
import joblib as joblib
import os
import hashlib
import requests
import json
import base64


app = Flask(__name__)
port = int(os.environ.get('PORT', 4000))
```

```python
    app.config['SECRET_KEY'] = 'abc123@567$#'


def callEnzoicAPI(password):
    apiKey = 'bafa4ec6c2c041fbad38fa848cb3b3f4'
    secretKey = 'G3n!=q$@cuEA-5KqVeQ!XPpfA$PWqz3t'
    authorizationParameter = apiKey+":"+secretKey
    authorizationParameter = authorizationParameter.encode('ascii')
    authorizationParameter =
(base64.b64encode(authorizationParameter)).decode('ascii')
    authorizationParameter = "basic "+str(authorizationParameter)
    password = password.encode('utf-8')
    sha1HashTemp = hashlib.sha1(password)
    sha1Hash = str(sha1HashTemp.hexdigest())
    sha256HashTemp = hashlib.sha256(password)
    sha256Hash = str(sha256HashTemp.hexdigest())
    md5HashTemp = hashlib.md5(password)
    md5Hash = str(md5HashTemp.hexdigest())
    rawData = {'partialSHA1':sha1Hash, 'partialSHA256':sha256Hash,
'partialMD5':md5Hash}
    url = 'https://api.enzoic.com/passwords'
    response = requests.post(url, data =
json.dumps(rawData),headers={'content-type':'application/json',
    'authorization':authorizationParameter})
    if(response.status_code == 404):
        print("Not found")
        return (False,0)
    finalResponse = json.loads(response.content.decode('ascii'))
    return (finalResponse["candidates"][0]["revealedInExposure"],
finalResponse["candidates"][0]["exposureCount"])


def scoreCalculate(password):
    length = len(password)
    score = 0
    upper = 0
    lower = 0
    numbers = 0
    symbols = 0
    consecutiveLower = 0
    consecutiveUpper = 0
```

```python
        consecutiveNumber = 0
        for i in range(0,length-1):
            if(password[i].isupper()):
                upper += 1
            elif(password[i].islower()):
                lower += 1
            elif(password[i].isdigit()):
                numbers += 1
            else:
                symbols += 1

            if(password[i].isupper() and password[i+1].isupper()):
                consecutiveUpper += 1
            else:
                score = score-consecutiveUpper*2
                consecutiveUpper = 0
            if(password[i].islower() and password[i+1].islower()):
                consecutiveLower += 1
            else:
                score = score-consecutiveLower*2
                consecutiveLower = 0
            if(password[i].isdigit() and password[i+1].isdigit()):
                consecutiveNumber += 1
            else:
                score = score-consecutiveNumber*2
                consecutiveNumber = 0

        score = score + length*4+((length-lower)*2)+((length-
upper)*2)+(numbers*4)+symbols*6

        if(numbers==0 and symbols==0):
            score = score - length

        if(upper==0 and lower==0 and symbols==0):
            score = score - length

        return score

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

```python
    def homepage():
        return render_template('index.html')


    @app.route('/main/', methods=['GET', 'POST'])
    def mainpage():
        if request.method == "POST":
            enteredPassword = request.form['password']
        else:
            return render_template('index.html')


        # Load the algorithm models
        DecisionTree_Model = joblib.load('DecisionTree_Model.joblib')
        LogisticRegression_Model =
joblib.load('LogisticRegression_Model.joblib')
        NaiveBayes_Model = joblib.load('NaiveBayes_Model.joblib')
        NeuralNetwork_Model = joblib.load('NeuralNetwork_Model.joblib')


        Password = [enteredPassword]


        # Predict the strength
        DecisionTree_Test = DecisionTree_Model.predict(Password)
        LogisticRegression_Test =
LogisticRegression_Model.predict(Password)
        NaiveBayes_Test = NaiveBayes_Model.predict(Password)
        NeuralNetwork_Test = NeuralNetwork_Model.predict(Password)
        status, count = callEnzoicAPI(enteredPassword)
        score = scoreCalculate(enteredPassword)


        return render_template("main.html",
DecisionTree=DecisionTree_Test,
                                            LogReg=LogisticRegression_Te
st,

                                            NaiveBayes=NaiveBayes_Test,
                                            NeuralNetwork=NeuralNetwork_
Test,

                                            status=status,
                                            count=count,
                                            score=score)
```
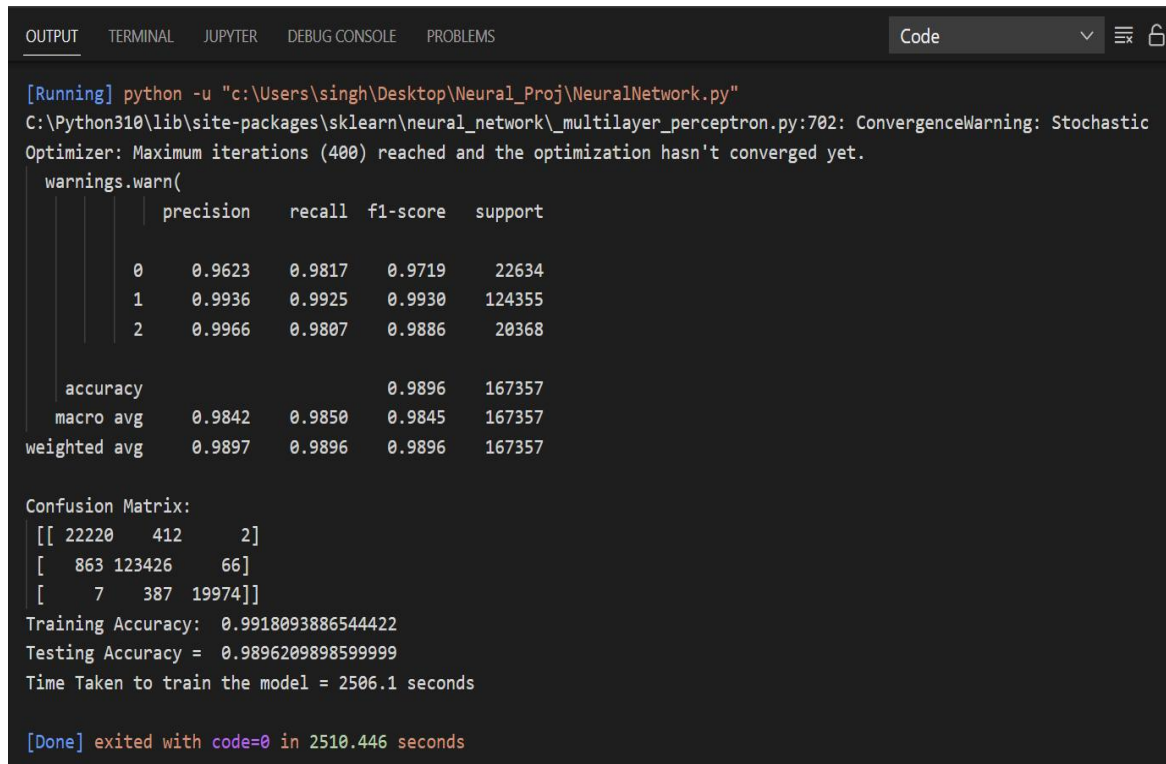
```
if __name__ == "__main__":
    app.run(host='127.0.0.1', port=port, debug=True)
```

Web Interface:
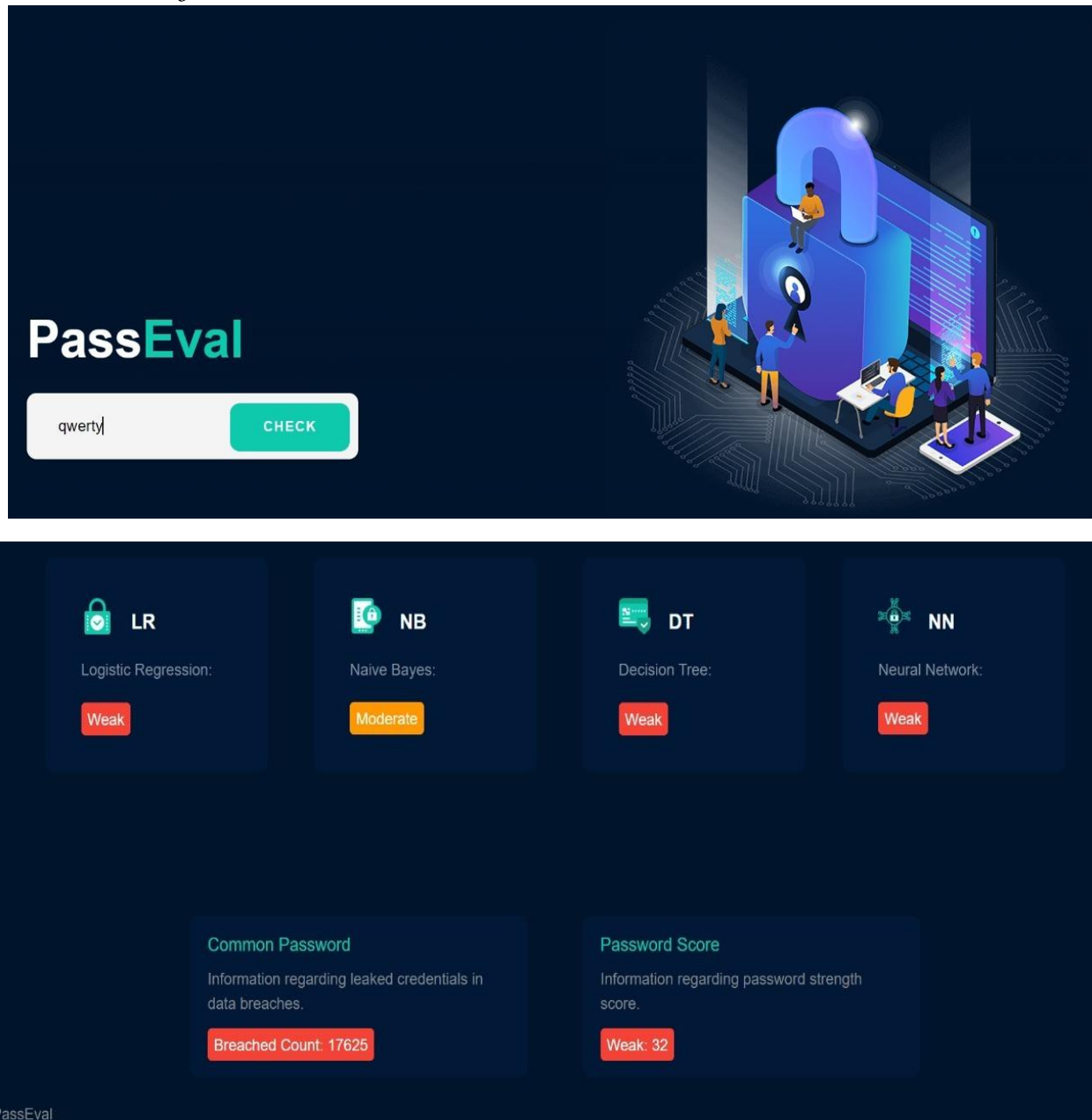
ML Algorithms Training Analysis:

Neural Network

The above figure gives the classification report of Neural Networks algorithm. All the details like f1 score, precision, accuracy, recall, confusion matrix, training and testing accuracies are mentioned.

Decision Tree

```
OUTPUT    TERMINAL    JUPYTER    DEBUG CONSOLE    PROBLEMS

[Running] python -u "c:\Users\singh\Desktop\Neural_Proj\DecisionTree.py"
              precision    recall   f1-score    support

         0       0.8402     0.8401     0.8402      22634
         1       0.9522     0.9538     0.9530     124355
         2       0.8641     0.8557     0.8599      20368

    accuracy                           0.9265     167357
   macro avg       0.8855     0.8832     0.8843     167357
weighted avg       0.9264     0.9265     0.9264     167357

Confusion Matrix:
 [[ 19014   3385     235]
 [  3239 118609    2507]
 [   376   2563   17429]]
Training Accuracy:  0.9814258505433768
Testing Accuracy =  0.926474542445192
Time Taken to train the model = 162.03 seconds

[Done] exited with code=0 in 164.031 seconds
```

The above figure gives the classification report of Decision Tree algorithm. All the details like f1 score, precision, accuracy, recall, confusion matrix, training and testing accuracies are mentioned.

Naive Bayes

```
OUTPUT    TERMINAL    JUPYTER    DEBUG CONSOLE    PROBLEMS

[Running] python -u "c:\Users\singh\Desktop\Neural_Proj\NaiveBayes.py"
              precision    recall   f1-score    support

         0       0.7324     0.0414     0.0783      22634
         1       0.8146     0.9675     0.8845     124355
         2       0.7979     0.7206     0.7573      20368

    accuracy                           0.8122     167357
   macro avg       0.7816     0.5765     0.5734     167357
weighted avg       0.8015     0.8122     0.7600     167357

Confusion Matrix:
 [[   936  21686      12]
 [   341 120308    3706]
 [     1   5689   14678]]
Training Accuracy:  0.8122806886507077
Testing Accuracy =  0.8121680001434061
Time Taken to train the model = 19.89 seconds

[Done] exited with code=0 in 22.618 seconds
```
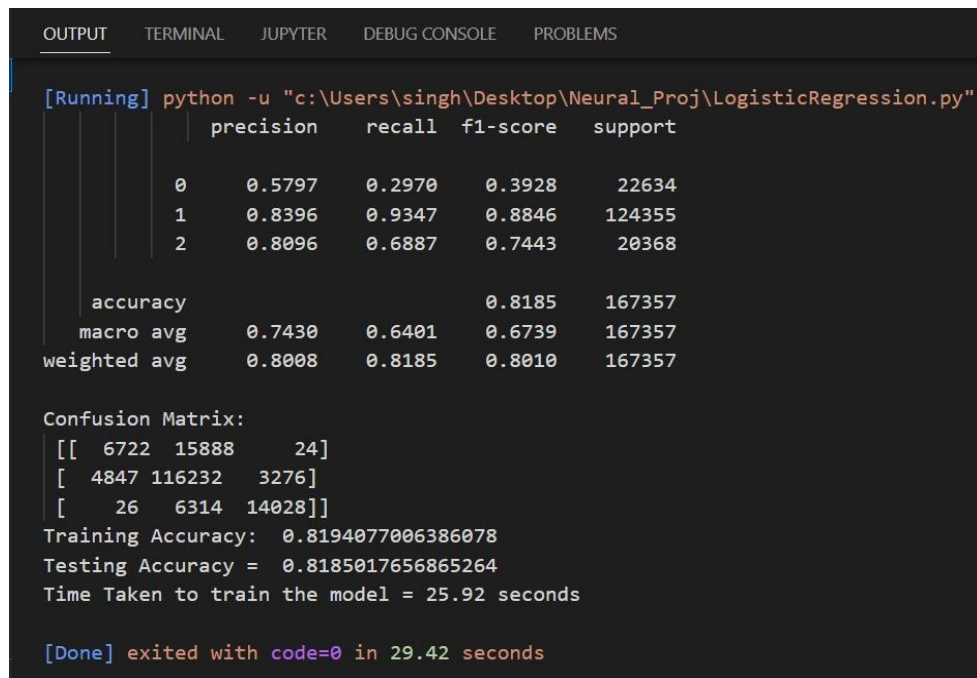
The above figure gives the classification report of Naïve Bayes algorithm. All the details like f1 score, precision, accuracy, recall, confusion matrix, training and testing accuracies are mentioned.

Logistics Regression

```
OUTPUT    TERMINAL    JUPYTER    DEBUG CONSOLE    PROBLEMS

[Running] python -u "c:\Users\singh\Desktop\Neural_Proj\LogisticRegression.py"
              precision    recall  f1-score   support

           0     0.5797    0.2970    0.3928     22634
           1     0.8396    0.9347    0.8846    124355
           2     0.8096    0.6887    0.7443     20368

    accuracy                         0.8185    167357
   macro avg     0.7430    0.6401    0.6739    167357
weighted avg     0.8008    0.8185    0.8010    167357

Confusion Matrix:
 [[  6722  15888     24]
 [  4847 116232   3276]
 [    26   6314  14028]]
Training Accuracy:  0.8194077006386078
Testing Accuracy =  0.8185017656865264
Time Taken to train the model = 25.92 seconds

[Done] exited with code=0 in 29.42 seconds
```
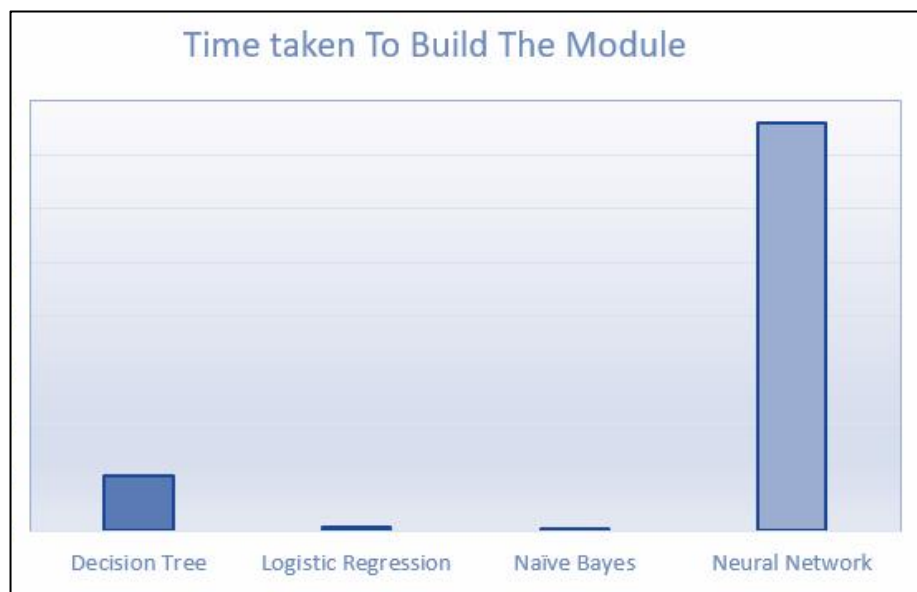
The above figure gives the classification report of Decision Tree algorithm. All the details like f1 score, precision, accuracy, recall, confusion matrix, training and testing accuracies are mentioned.

## *Graphical Analysis:*

Building Modules

The above figure gives the details regarding the time taken to train and test the ML models. As depicted the Neural Networks ML model took the most time.

Training Accuracy

The above figure gives the details regarding the training accuracy of the ML models. As depicted the Neural Networks ML model has the highest accuracy among the others (99.18%).

Testing Accuracy
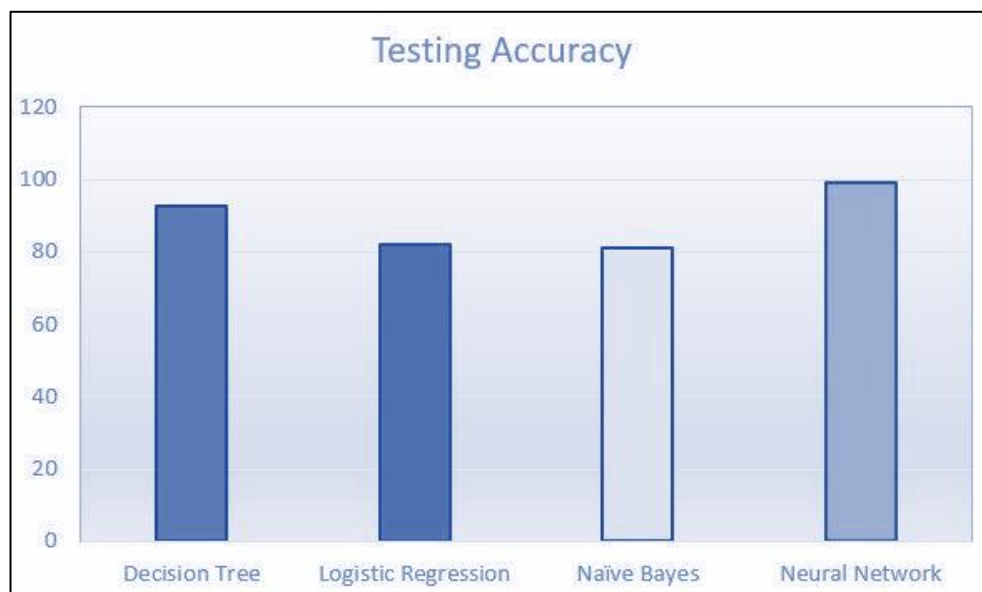
The above figure gives the details regarding the testing accuracy of the ML models. As depicted the Neural Networks ML model has the highest accuracy among the others (98.962%).

From the figure it is established that the training accuracy of the Decision Tree method & Neural network are almost similar while there is a slight difference between Logistic Regression and Naive Bayes.

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable.
The Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

References

[1] Matteo Dell'Amico and Maurizio Filippone. 2015. Monte Carlo Strength Evaluation: Fast and Reliable Password Checking. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). Association for Computing Machinery, New York, NY, USA, 158

[2] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. 2017. Design and Evaluation of a Data-Driven Password Meter.

[3] Kim, P.; Lee, Y.; Hong, Y.-S.; Kwon, T. A Password Meter without Password Exposure. Sensors 2021, 21, 345

[4] Better passwords through science (and neural networks), William Melicher, Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor (2017).

[5] Password Strength Prediction Using Supervised Machine Learning Techniques, Vijaya M.S, Jamuna K.S, Karpagavalli S.(Advances in Computing, Control, and Telecommunication Technologies).

[6] J. Galbally, I. Coisel and I. Sanchez, "A New Multimodal Approach for Password Strength Estimation—Part I: Theory and Algorithms," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 12, pp. 2829-2844, Dec. 2017, doi: 10.1109/TIFS.2016.2636092.

[7] J. Galbally, I. Coisel and I. Sanchez, "A New Multimodal Approach for Password Strength Estimation—Part II: Experimental Evaluation," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 12, pp. 2845-2860, Dec. 2017, doi: 10.1109/TIFS.2017.2730359.

[8] Umar Farooq, 2020, Real Time Password Strength Analysis on a Web Application Using Multiple Machine Learning Approaches, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 09, Issue 12 (December 2020).

[9] https://www.kaggle.com/bhavikbb/password-strength-classifier-dataset