VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

PREDICTING PASSWORD STRENGTH USING SUPERVISED MACHINE LEARNING TECHNIQUES

A PROJECT REPORT

Submitted by

JASWANTH REDDY L (21MAI0016)

ANUSHA P (21MAI0050)

PUNITHA G (21MAI0056)

VENKATA SAI KUMAR B (21MAI0063)

In partial fulfilment for the award of the degree

Of

MASTER OF TECHNOLOGY

IN

CSE WITH SPECIALIZATION IN AI & ML

VIT UNIVERSITY

Vellore- 632014

JUNE 2022

ACKNOWLEDGEMENT

We extend our sincere gratitude to our project guide **Dr. SUNIL KUMAR P V, Ph.D.,** for providing valuable insights and resources leading to the successful completion of our project. We would also like to thank the faculty members of our department for their critical advice and guidance which was indispensable for our work. Last but not the least, we place a deep sense of gratitude to our friends and families who have been a constant source of inspiration during the preparation of this project.

ABSTRACT

Passwords are a vital component of system security. Despite the fact that there are many alternatives, passwords are still the most effective process of ensuring identity in many applications. They reflect an individual's identity for a system and give a simple, straightforward manner of preserving it. There are so many ways for an outsider with little technical knowledge or talent to acquire the passwords of legitimate users. Users are repeatedly informed that a strong password is vital these days to protect sensitive data. To ensure that these vulnerabilities are not exploited, organizations must be aware of the consequences that passwords bring and implement strong procedures for governing password generation and use. Classification based machine learning algorithm to classify the strength of passwords within predefined categories. Techniques like Decision tree classifier, Multilayer Perceptron, Naive Bayes Classifier, and Support Vector Machine can be used. The model with high accuracy can help to predict whether the password strength is strong or weak.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Passwords have become increasingly important in today's world. Passwords are required for a variety of tasks by a normal computer user, including logging into accounts, receiving email from servers, transferring payments, shopping online, accessing applications, databases, networks, and web sites, and even reading the morning newspaper online. Every day, the issue of choosing and utilizing strong passwords becomes more essential. The number and value of services supplied via computers and networks is rapidly increasing, and many of these services require passwords or other kinds of user authentication. Users must use multiple passwords for different systems or services for a variety of reasons, including obvious security issues, making it more difficult to remember and protect one's password. Passwords are important not simply for logging in, but also in more advanced service- granting systems like Kerberos. Finally, passwords are required in authentication and encryption software, which is becoming increasingly important in many applications, to secure secret information that cannot be recalled by the user (e.g. private keys). The average user prefers a password that is easy to remember and guess, rather than one that is strong. Data thieves, hackers, and other criminals are preying on those who aren't security savvy these days, and the threat is real and growing. To guarantee that these weaknesses are not exploited, it is critical for companies to realize the risks that passwords face and to implement strong policies governing the generation and use of passwords.

1.1 INTRODUCTION TO THE DOMAIN

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

Classification

As the name suggests, Classification is the task of "classifying things" into sub-categories. But, by a machine. If that doesn't sound like much, imagine your computer being able to differentiate between you and a stranger. Between a potato and a tomato. Between an A grade and a F. In Machine Learning and Statistics, Classification is the problem of identifying to which of a set of categories (sub populations), a new observation belongs to, on the basis of a training set of data containing observations and whose categories membership is known. Classification is of two types. They are

Binary Classification

Multi-class Classification

1.2 PURPOSE OF THE PROJECT

Passwords, being the most common mechanism for authentication for its easy implementation, pave a way for attackers to break into accounts by guessing passwords. This happens due to predictable patterns that humans usually set these

passwords to like dictionary words, known phrases, names of people and places, keyboard patterns, etc. Many password cracking tools have been made to guess passwords either online or offline that mostly results in cracking such accounts with weak passwords or common password patterns. The proposed model provides a common and efficient way to defend against these online and offline attacks by forcing the users to choose a strong password by implementing multiple machine learning algorithms such as Decision Tree (DT), Naïve Bayes (NB), Multi-layer perceptron and Support Vector Machine on a web application over real time. This results in logging in into the user's account only if the password strength from all algorithms happens to be strong.

1.3 SCOPE OF THE PROJECT

- Proper Authorization
- Reduce Fraud and Build Secure online Relationships
- Improved User Experience
- Increased Security
- Reduced administration overheads

1.4 SUMMARY

Dictionary words, well-known phrases, names of people and places, keyboard sequences, and other predictable patterns are used by humans to create passwords. Many password cracking tools have been developed to guess passwords online or offline, with the majority of accounts with weak passwords or common password patterns being cracked. By implementing various machine learning algorithms on a web application in real time, the suggested approach provides a consistent and efficient way to fight against these online and offline assaults by pushing users to choose a strong password. This results in the user's account being logged into only if the password strength from all algorithms is strong.

CHAPTER 2

LITERATURE SURVEY

This chapter briefly discusses the literature survey done on various projects based on aquaculture. Their performance, uses and limitations are analyzed.

2.1 Supervised Machine Learning Techniques, Cyber security Habits and Human Generated Password Entropy for Hacking Prediction

Authors: Pedro Taveras , Liliana Hernandez

Published in: Association for Information Systems AIS Electronic Library (AISeL)

Date of Conference: 2018

This study intended to determine to what extent end users' password habits correlate with the likelihood of a hacked account, as well as to develop model predictions that allow for the prediction of a hacking event using the described habits as input elements. Data on user security behaviors was collected using a survey instrument, and a password dictionary was generated by requesting that participants choose a password and write it down. Despite the preliminary studies indicating that the study is still restricted, machine learning algorithms, notably neural networks, can be utilized to simulate the prediction. In the following phases of this study, the researchers should try to explore the factors using a variety of tools and methods.

2.2 Measuring Password Strength by Simulating Password-Cracking Algorithms

Authors: Patrick Gage Kelley; Saranga Komanduri; Michelle L. Mazurek; Richard Shay; Timothy Vidas;

Published in: IEEE Symposium on Security and Privacy

Date of Conference: 2012

Despite substantial advancements in attackers' password cracking capabilities, text-based passwords remain the most common authentication technique in computer systems. As a result of this vulnerability, password composition policies have become more complicated. There is, however, a paucity of research defining criteria to define password strength and utilising them to assess password-composition strategies. In this research, we use an online study to examine 12,000 passwords gathered under seven different composition policies. We create a distributed approach for calculating how well various heuristic password-guessing algorithms guess passwords. We explore (a) the resistance of passwords formed under various situations to guessing; (b) the performance of guessing algorithms under various training sets using this method.(c) the association between passwords intentionally established under a certain composition policy and alternative passwords that happen to fit the same requirements and (d) the relationship between guessability, as judged by password-cracking techniques, and entropy estimates; Our findings contribute to a better understanding of password-composition policies and password-security metrics.

2.3 SUMMARY

In this fast pace of life, to secure confidential information that cannot be recalled by the user, passwords are necessary in authentication and encryption software, which is becoming increasingly vital in many applications. The average user would rather have a password that is simple to remember and guess than one that is complex. The threat of data thieves, hackers, and other criminals preying on those who aren't security savvy is real and expanding these days. To ensure that these flaws are not exploited, it is vital for businesses to understand the risks that passwords pose and to develop effective policies governing password generation and use.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

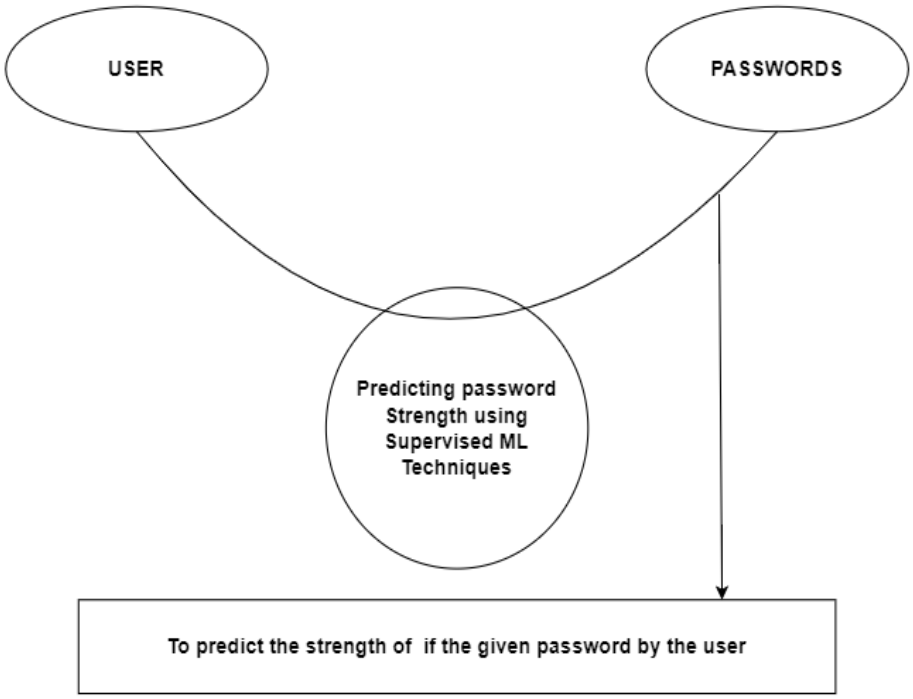
3.1 PROBLEM DEFINITION

Despite their ease of implementation, passwords are the most often used authentication technique, allowing attackers to hack into accounts by guessing passwords. The average user would rather have a password that is simple to remember and guess than one that is complex. The threat of data thieves, hackers, and other criminals preying on those who aren't security savvy is real and expanding these days. To ensure that these flaws are not exploited, it is vital for businesses to understand the risks that passwords pose and to develop effective policies governing password generation and use.

3.2 NEED ANALYSIS

The Bull Diagram describes the purpose of the System and about all the Actors who are involved in the system.

3.2.1 BULL DIAGRAM

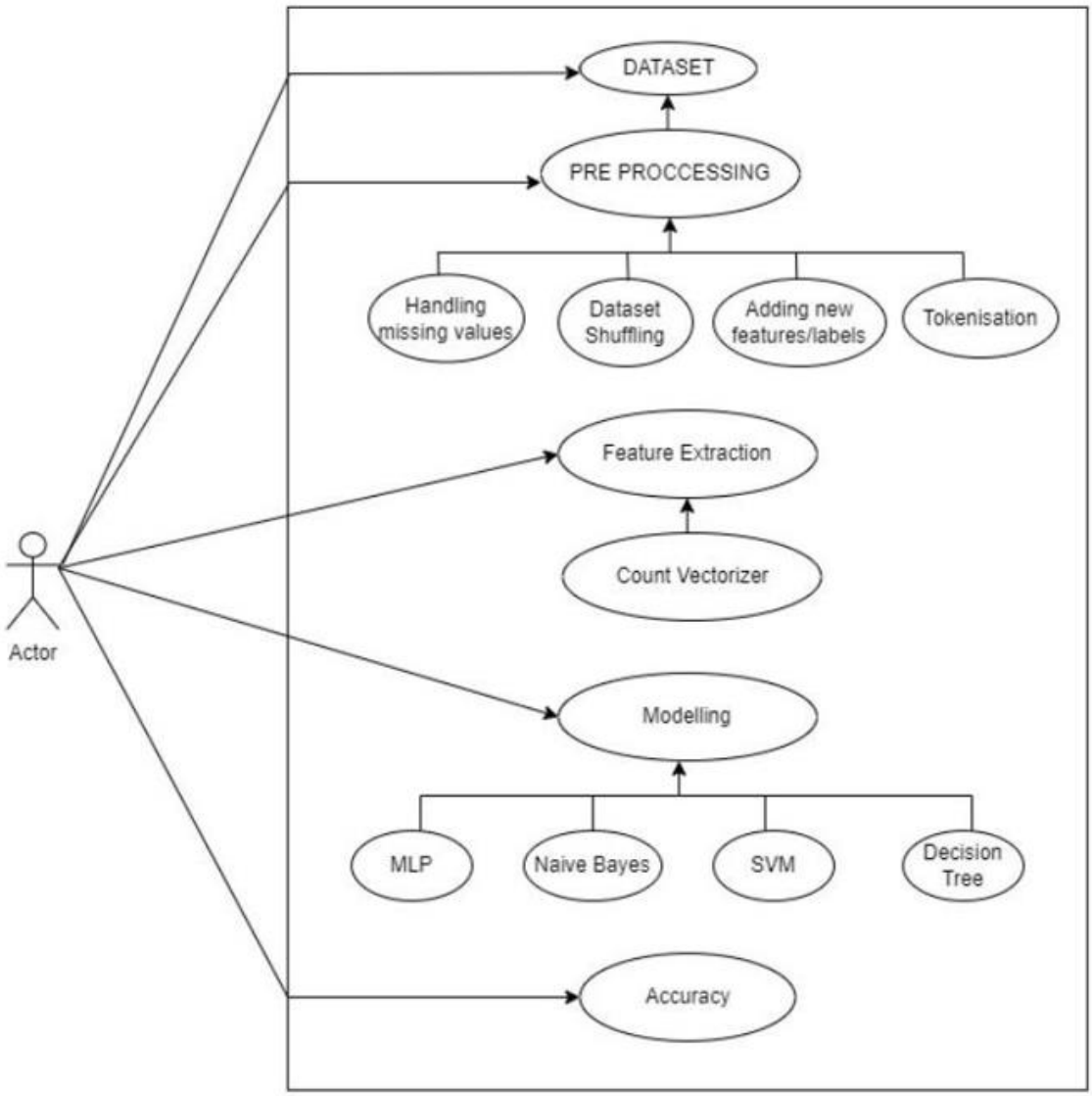


3.2 UML DIAGRAM

UML diagrams are drafted below with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the ‘Predicting Password Strength using Supervised Machine Learning Techniques’.

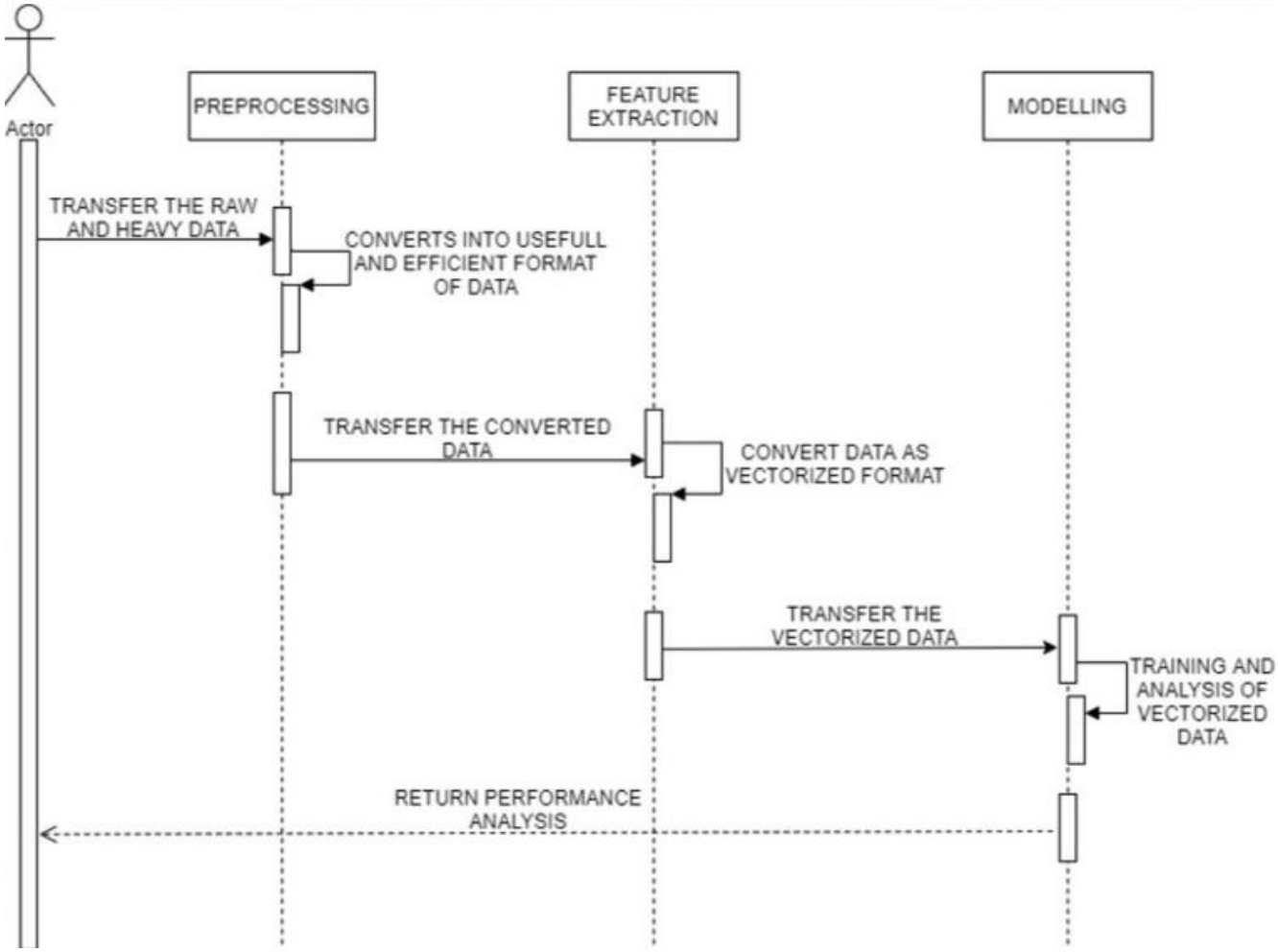
3.2.1 USE CASE DIAGRAM

A Use Case Diagram is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.



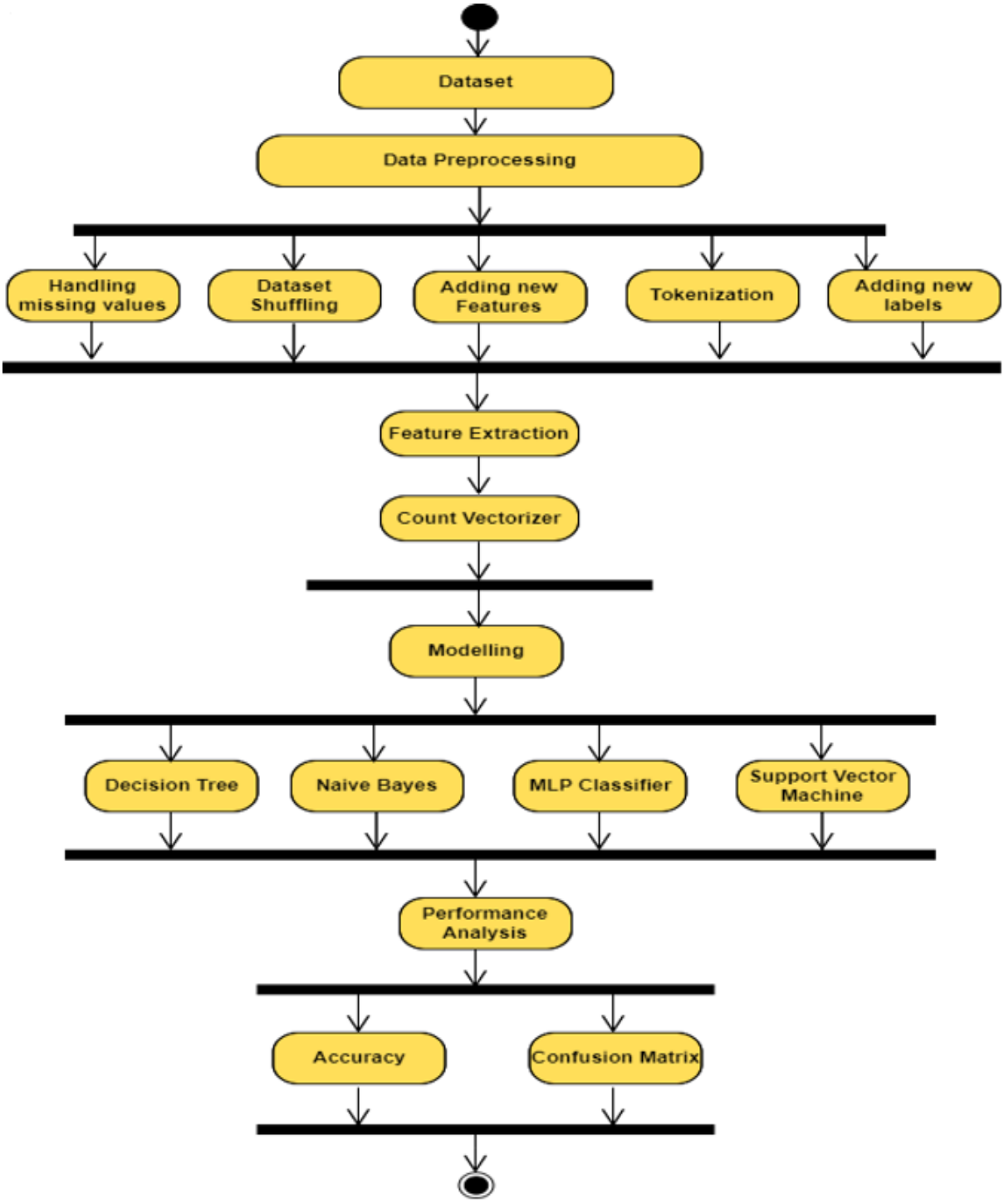
3.2.2 SEQUENCE DIAGRAM

A Sequence Diagram shows object interactions arranged in time sequence. It depicts the sequence of messages exchanged between the objects to carry out the functionality of the scenario.



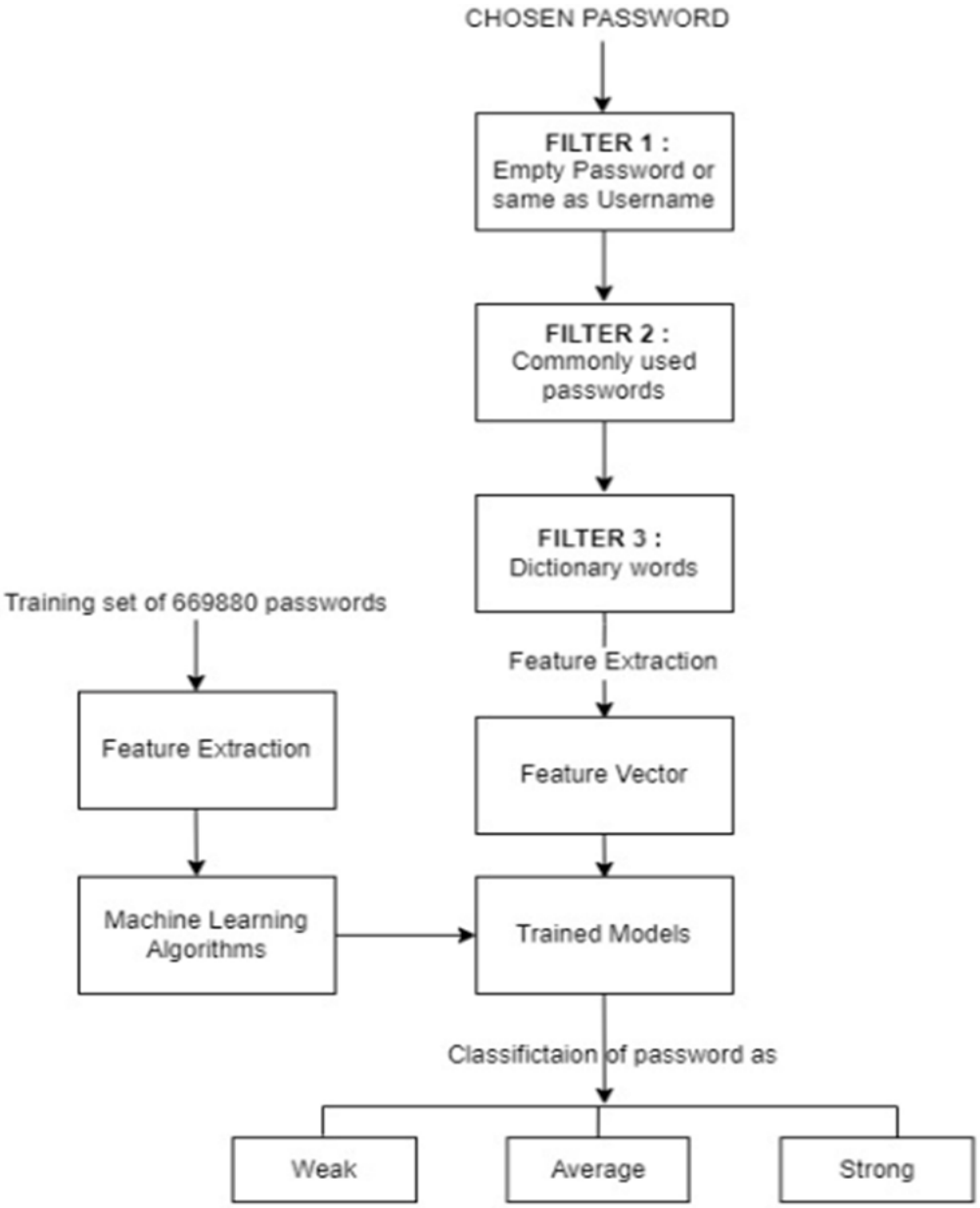
3.2.3 ACTIVITY DIAGRAM

Activity Diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.



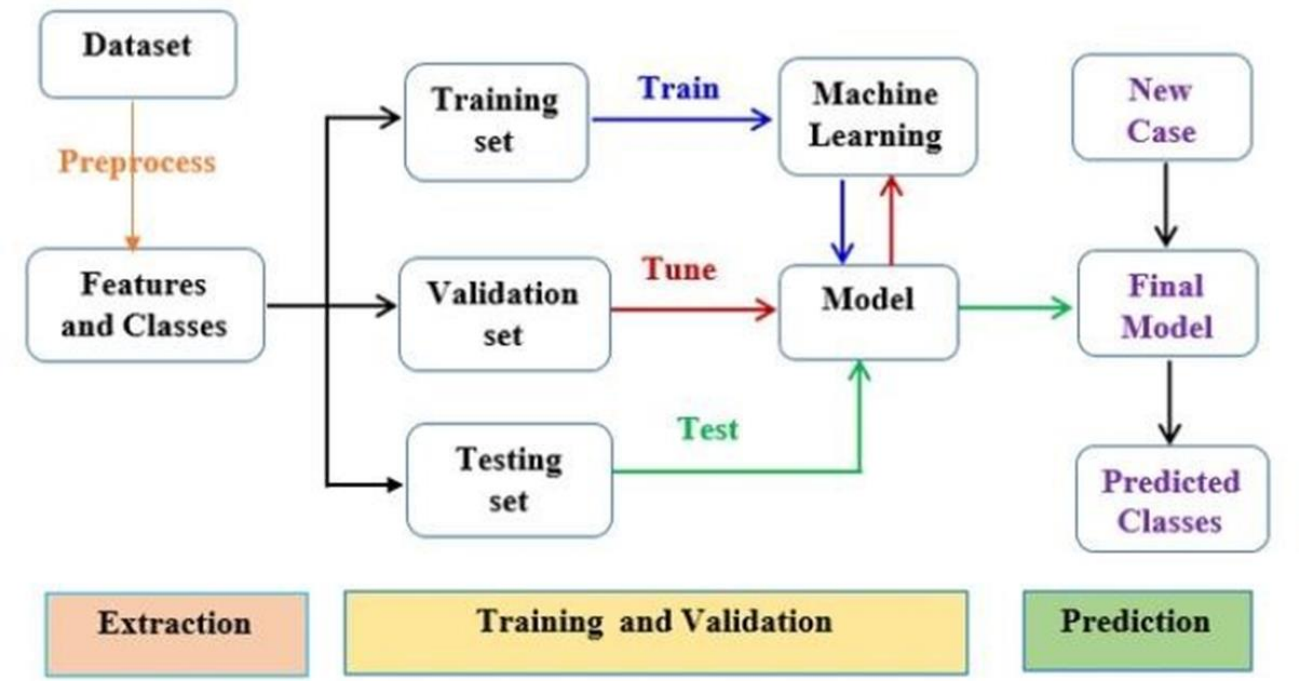
3.4 PROCESS FLOW DIAGRAM

A sequential representation of a process and its components, including operations, timelines, and people involved, and resources needed. The main objective of process flows is to help you standardize and optimize your processes and help your team better understand how your business works. Process Flow Diagrams (PFDs) are a graphical way of describing a process, its constituent tasks, and their sequence. A PFD helps with the brainstorming and communication of the process design.



3.5 WORKFLOW DIAGRAM

A workflow diagram (also known as a workflow) provides a graphic overview of the business process. Using standardized symbols and shapes, the workflow shows step by step how your work is completed from start to finish. It also shows who is responsible for work at what point in the process.



3.6 SUMMARY

The UML diagrams for the system are drafted for a better understanding of the system's objects, functions and classes.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 FUNCTIONAL REQUIREMENTS

4.1.1 Software Requirements

4.1.1.1 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands. Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It has more than 1500 Python/R data science packages. Anaconda simplifies package management and deployment. It has tools to easily collect data from sources using machine learning and AI. It creates an environment that is easily manageable for deploying any project. Using the Jupyter lab we build ML models like Naive bayes, Multi-Layer Perceptron, Support Vector Machine and Decision tree.Navigator is available for Windows, macOS, and Linux.

Python Packages

Packages or additional libraries help in scientific computing and computational modeling. In Python, the packages are not part of the Python standard library. Few major packages are –

numpy (NUMeric Python): matrices and linear algebra

scipy (SCIentific Python): many numerical routines

matplotlib: (PLOTtingLIBrary) creating plots of data

sympy (SYMbolic Python): symbolic computation

pytest (Python TESTing): a code testing framework.

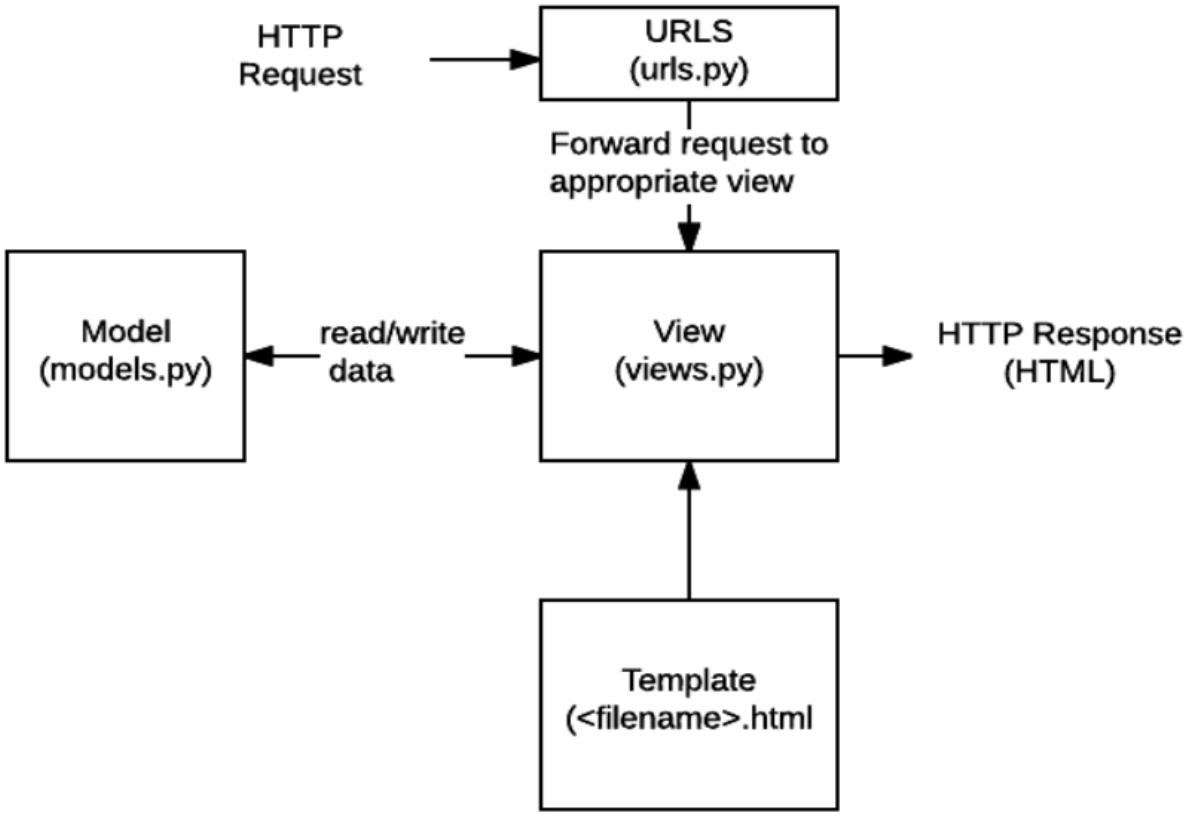
Together with a list of Python packages, tools like editors, Python distributions include the Python interpreter. Anaconda is one of several Python distributions.

Jupyter Notebook

Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text and multimedia resources in a single document. Jupyter Notebook provides you with an easy-to-use, interactive data science environment across many programming languages that doesn't only work as an IDE, but also as a presentation or education tool. The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

4.1.1.2 Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support. Django web applications typically group the code that handles each of these steps into separate files:



- **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
- **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates.
- **Models**: Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data from a model. A template can be used to define the structure of any type of file; it

doesn't have to be HTML.We use Django because it is complete, versatile, Secure, Scalable, Maintainable and Portable.

4.2 Non-Functional Requirements

4.2.1 Performance Requirements

Dynamic numerical requirements:

- 90% of the operations shall be processed in less than 10s.

Static numerical requirements:

- The number of simultaneous users: Many

4.3 SUMMARY

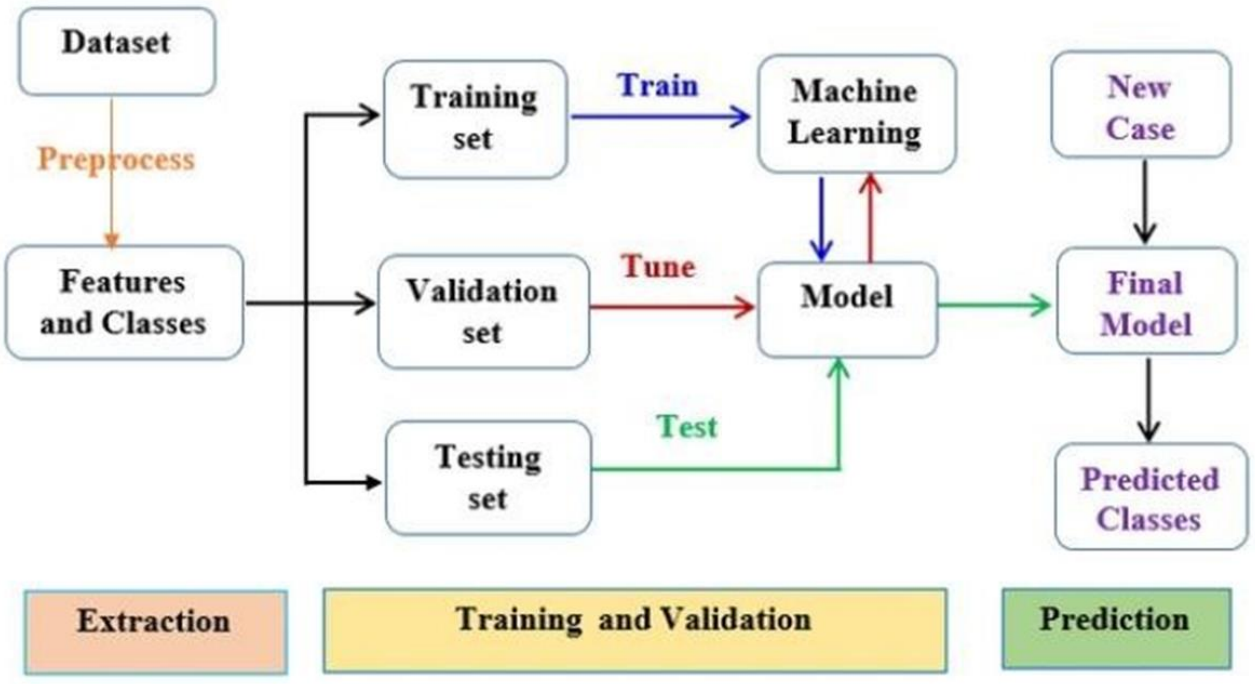
Hence, when compared with the other available models, the technique which is proposed here, is much advantageous in effectively monitoring the password strength of the user and categorizing it into “weak”, ”medium” and “strong” which helps the user to secure their applications from attackers.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 SYSTEM ARCHITECTURE

A system architecture or systems architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system. This system provides majorly four modules, which are mentioned below. The system provides user friendly GUI and data flows to all modules in dynamic way.



WORKFLOW DIAGRAM

5.2 MODULE DESCRIPTION

5.2.1 Data Preprocessing

Data Preprocessing transforms the raw data into a useful and efficient format by removing noise and inconsistent data. So, it creates the reliable consistent data that improves the efficiency of the training data for analysis and also enables accurate decision – making.

- Handling missing values
- Dataset Shuffling
- Adding new features/labels
- Tokenization

5.2.2 Feature Extraction

Feature extraction refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set.

Initializing CountVectorizer

```
In [51]: from sklearn.feature_extraction.text import CountVectorizer
         vectorizer_state = CountVectorizer()

In [52]: #allpasswords = np.array(allpasswords).reshape(-1,1)

In [53]: X = vectorizer_state.fit_transform(allpasswords)
```

Splitting vectorized dataset

```
In [55]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, ylabels, test_size=0.2, random_state=42)

In [56]: print(X_train.shape, X_test.shape) #y_train.shape, y_test.shape)
         (16000, 10105) (4000, 10105)
```

5.2.3 ML modeling

In modelling, we are splitting the data as train and test. This is to estimate the performance of machine learning algorithms i.e., used to make predictions on the data not used to train the model. The training data is fit into the model and test data is used to predict. It is a fast and easy procedure that allows us to compare the performance of machine learning algorithms for predictive-based modelling problems. Supervised Machine Learning Algorithms used here are.

DECISION TREE

Decision trees contain a hierarchically organized model. So that an attribute can be classified, deciding in each node the condition that it fulfills. Describing a path from the root to the leaves. In each node, decisions are exclusive. It is assumed that the classes are disjoint, that is, an instance is of class a or class b, but cannot be at the same time of classes a and b. The classes are exhaustive. The decision tree is constructed in the learning phase and then used to predict the instances. The more types of conditions allowed, the more possibilities we will have to find the patterns behind the data. The question in a good decision tree learning algorithm is to find the appropriate balance between expressivity and efficiency. The C4.5 algorithm proposed by Quinlan (1993) was the implemented. Decision tree is based on the divide and conquer method and partition criteria (Gain Ratio), including rule based pruning and other more sophisticated mechanisms

1.Decision Tree

```
from scipy.stats import randint
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

# Setup the parameters and distributions to sample from: param_dist
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}

# Instantiate a Decision Tree classifier: tree
tree = DecisionTreeClassifier()

# Instantiate the RandomizedSearchCV object: tree_cv
tree_cv = RandomizedSearchCV(tree, param_dist, cv=5)

# Fit it to the data
tree_cv.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
```

```
Tuned Decision Tree Parameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': 7, 'min_samples_leaf': 4}
Best score is 0.7394375
```

```
from sklearn.metrics import accuracy_score
dect_model = DecisionTreeClassifier(criterion = 'gini', max_depth = 3, max_features = 1, min_samples_leaf = 2)
dect_model.fit(X_train, y_train)
y_test_pred_dect = dect_model.predict(X_test)
y_train_pred_dect = dect_model.predict(X_train)
print(accuracy_score(y_train, y_train_pred_dect))
print(accuracy_score(y_test, y_test_pred_dect))
```

```
0.739375
0.7305
```

In [45]:
```
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
print(classification_report(y_train,y_train_pred_dect))
print(classification_report(y_test,y_test_pred_dect))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2220
           1       0.74      1.00      0.85     11830
           2       0.00      0.00      0.00      1950

    accuracy                           0.74     16000
   macro avg       0.25      0.33      0.28     16000
weighted avg       0.55      0.74      0.63     16000

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       557
           1       0.73      1.00      0.84      2922
           2       0.00      0.00      0.00       521

    accuracy                           0.73      4000
   macro avg       0.24      0.33      0.28      4000
weighted avg       0.53      0.73      0.62      4000
```

MULTI-LAYER PERCEPTRON

Multilayer perceptron is one of the most used supervised algorithms of neural networks. Its architecture is based on an input layer, another output layer, and at least one hidden layer. Activation is propagated in the network through the weights from the input layer to the intermediate layer where some activation function is applied to the incoming inputs. Then activation is propagated through the weights to the output layer (Witten, Frank, Hall and Pal, 2016). The learning capacity of the neural network is determined by the number of hidden layers and the number of units in each layer. The variables to be used must be chosen carefully: the aim is to include in the model the predictor variables that actually predict the dependent or exit variable, but which in turn do not have relations with each other since this may cause unnecessary overfitting in the model.

MLPClassifier

In [35]:
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(max_iter=100)
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
clf.fit(X_train, y_train)

#print the tuned parameters
print("Tuned MLPClassifier Parameters: {}".format(clf.best_params_))
print("Best score is {}".format(clf.best_score_))
```

```
Tuned MLPClassifier Parameters: hidden_layer_sizes=(6,5), random_state=5, verbose=True, learning_rate='adaptive', act
ivation='relu', solver='adam', alpha=0.05
```

In [30]:
```
# Create model object
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=True,
                    learning_rate='adaptive',
                    activation='relu',
                    solver='adam',
                    alpha=0.05)

# Fit data onto the model
clf.fit(X_train,y_train)
y_test_pred_MLP = clf.predict(X_test)
y_train_pred_MLP = clf.predict(X_train)
print(accuracy_score(y_train, y_train_pred_MLP))
print(accuracy_score(y_test, y_test_pred_MLP))
```

```
Iteration 113, loss = 0.04361594
Iteration 114, loss = 0.04358863
Iteration 115, loss = 0.04354080
Iteration 116, loss = 0.04348845
Iteration 117, loss = 0.04338801
Iteration 118, loss = 0.04335436
Iteration 119, loss = 0.04329096
Iteration 120, loss = 0.04338493
Iteration 121, loss = 0.04339499
Iteration 122, loss = 0.04326376
Iteration 123, loss = 0.04318888
Iteration 124, loss = 0.04313876
Iteration 125, loss = 0.04314115
Iteration 126, loss = 0.04314795
Iteration 127, loss = 0.04314126
Iteration 128, loss = 0.04321860
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
0.99975
0.91325
```

In [46]:
```
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
print(classification_report(y_train,y_train_pred_MLP))
print(classification_report(y_test,y_test_pred_MLP))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2220
           1       1.00      1.00      1.00     11830
           2       1.00      1.00      1.00      1950

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

              precision    recall  f1-score   support

           0       0.99      0.69      0.81       557
           1       0.89      1.00      0.94      2922
           2       1.00      0.66      0.80       521

    accuracy                           0.91      4000
   macro avg       0.96      0.79      0.85      4000
weighted avg       0.92      0.91      0.91      4000
```

NAIVE BAYES

Probability theory and Bayesian methods are one of the techniques that have been used the most in artificial intelligence problems and, therefore, in machine learning and data mining. They are a practical method to make inferences from the data, inducing probabilistic models that will later be used to reason (formulate hypotheses) about new observed values (Cichosz, 2015). They allow us to explicitly calculate the probability associated to each of the possible hypotheses, which constitutes a great advantage over other techniques. In addition, Naïve Bayes provides a useful framework for the understanding and analysis of numerous data mining and learning techniques that do not work explicitly with probabilities. The Naïve Bayes algorithm is based on the assumption that the attributes are independent. Although this assumption is assumed, it is quite strong in most cases. Based on this assumption, the study performed an exhaustive analysis of the variables that were taken from the instrument.

Naive bayes

In [31]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB

nb_classifier = GaussianNB()

params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs_NB = GridSearchCV(estimator=nb_classifier,
                     param_grid=params_NB,
                     verbose=1,
                     scoring='accuracy')
gs_NB.fit(X_train.todense(), y_train)

#print the tuned parameters
print("Tuned MLPClassifier Parameters: {}".format(gs_NB.best_params_))
print("Best score is {}".format(gs_NB.best_score_))
```

In [32]:
```
from sklearn.naive_bayes import GaussianNB
naive_model = GaussianNB()
naive_model.fit(X_train.toarray(), y_train)
y_test_pred_naive = naive_model.predict(X_test.toarray())
y_train_pred_naive = naive_model.predict(X_train.toarray())
print(accuracy_score(y_train, y_train_pred_naive))
print(accuracy_score(y_test, y_test_pred_naive))
```

```
0.9996875
0.728
```

```
In [47]: from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
         print(classification_report(y_train,y_train_pred_naive))
         print(classification_report(y_test,y_test_pred_naive))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2220
           1       1.00      1.00      1.00     11830
           2       1.00      1.00      1.00      1950

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

              precision    recall  f1-score   support

           0       0.34      1.00      0.51       557
           1       1.00      0.69      0.81      2922
           2       1.00      0.67      0.80       521

    accuracy                           0.73      4000
   macro avg       0.78      0.79      0.71      4000
weighted avg       0.91      0.73      0.77      4000
```

SVM

Support Vector Machine a new approach to supervised pattern classification that has been successfully applied to a wide range of pattern recognition problems. Support vector machine is a training algorithm for learning classification and regression rules from data. SVM is most suitable for working accurately and efficiently with high dimensionality feature spaces. SVM is based on strong mathematical foundations and results in simple yet very powerful algorithms [6-9]. The standard SVM algorithm builds a binary classifier. A simple way to build a binary classifier is to construct a hyperplane separating class members from non-members in the input space. SVM also finds a nonlinear decision function in the input space by mapping the data into a higher dimensional feature space and separating it there by means of a maximum margin hyperplane. The system automatically identifies a subset of informative points called support vectors and uses them to represent the separating hyperplane, which is sparsely a linear combination of these points. Finally SVM solves a simple convex optimization problem.

Support Vector Machines(SVM)

In [36]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# define model and parameters
model = SVC()
kernel = ['poly', 'rbf', 'sigmoid', 'linear']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']

# define grid search
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train, y_train)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Tuned SVC Parameters: C = 50, gamma = 'scale', kernel = 'rbf'
```

In [34]:
```
from sklearn.svm import SVC
svm_model = SVC(C = 50, gamma = 'scale', kernel = 'rbf')
svm_model.fit(X_train, y_train)
y_test_pred_SVM = svm_model.predict(X_test)
y_train_pred_SVM = svm_model.predict(X_train)
print(accuracy_score(y_train, y_train_pred_SVM))
print(accuracy_score(y_test, y_test_pred_SVM))
```

```
0.999875
0.9155
```

In [48]:
```
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
print(classification_report(y_train,y_train_pred_SVM))
print(classification_report(y_test,y_test_pred_SVM))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2220
           1       1.00      1.00      1.00     11830
           2       1.00      1.00      1.00      1950

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

              precision    recall  f1-score   support

           0       1.00      0.69      0.82       557
           1       0.90      1.00      0.95      2922
           2       0.99      0.69      0.81       521

    accuracy                           0.92      4000
   macro avg       0.96      0.79      0.86      4000
weighted avg       0.92      0.92      0.91      4000
```

5.2.4 Deployment using Django framework(Heroku)

Django is full of shortcuts to make Web developers' lives easier, but all those tools are of no use if you can't easily deploy your sites. Since Django's inception, ease of deployment has been a major goal.

There are many options for deploying your Django application, based on your architecture or your particular business needs, but that discussion is outside the scope of what Django can give you as guidance.

Django, being a web framework, needs a web server in order to operate. And since most web servers don't natively speak Python, we need an interface to make that communication happen.

Django currently supports two interfaces: WSGI and ASGI.

- WSGI is the main Python standard for communicating between Web servers and applications, but it only supports synchronous code.
- ASGI is the new, asynchronous-friendly standard that will allow your Django site to use asynchronous Python features, and asynchronous Django features as they are developed.

Why Heroku?

Heroku is one of the longest running and popular cloud-based PaaS services. It originally supported only Ruby apps, but now can be used to host apps from many programming environments, including Django!

We are choosing to use Heroku for several reasons:

- Heroku has a free tier that is really free.

- As a PaaS, Heroku takes care of a lot of the web infrastructure for us. This makes it much easier to get started, because you don't worry about servers, load balancers, reverse proxies, or any of the other web infrastructure that Heroku provides for us under the hood.
- Heroku provides only short-lived storage so user-uploaded files cannot safely be stored on Heroku itself.
- The free tier will sleep an inactive web app if there are no requests within a half hour period. The site may then take several seconds to respond when it is woken up.
- The free tier will sleep an inactive web app if there are no requests within a half hour period. The site may then take several seconds to respond when it is woken up.
- Mostly it just works, and if you end up loving it, scaling your app is very easy.
- While Heroku is perfect for hosting, it makes things easy to set up and scale, at the cost of being less flexible, and potentially a lot more expensive once you get out of the free tier.

How does Heroku work?

Heroku runs Django websites within one or more "Dynos", which are isolated, virtualized Unix containers that provide the environment required to run an application. The dynos are completely isolated and have an ephemeral file system (a short-lived file system that is cleaned/emptied every time the dyno restarts). The only thing that dynos share by default are application configuration variables. Heroku internally uses a load balancer to distribute web traffic to all "web" dynos. Since nothing is shared between them, Heroku can scale an app horizontally by adding more dynos (though of course you may also need to scale your database to accept additional connections).

Because the file system is ephemeral you can't install services required by your application directly (e.g. databases, queues, caching systems, storage, email services, etc). Instead Heroku web applications use backing services provided as independent

"add-ons" by Heroku or 3rd parties.

Once attached to your web application, the dynos access the services using information contained in application configuration variables. In order to execute your application Heroku needs to be able to set up the appropriate environment and dependencies, and also understand how it is launched. For Django apps we provide this information in a number of text files:

- runtime.txt: the programming language and version to use.
- requirements.txt: the Python component dependencies, including Django.
- Procfile: A list of processes to be executed to start the web application. For Django this will usually be the Gunicorn web application server (with a .wsgi script).
- wsgi.py: WSGI configuration to call our Django application in the Heroku environment.

Developers interact with Heroku using a special client app/terminal, which is much like a Unix Bash shell. This allows you to upload code that is stored in a git repository, inspect the running processes, see logs, set configuration variables and much more! In order to get our application to work on Heroku we'll need to put our Django web application into a git repository, add the files above, integrate with a database add-on, and make changes to properly handle static files. Once we've done all that we can set up a Heroku account, get the Heroku client, and use it to install our website.

5.3 SUMMARY

By implementing multiple machine learning algorithms such as Decision Tree (DT), Naive Bayes (NB), Multi-layer perceptron, and Support Vector Machine on a web application in real time, the proposed model provides a common and efficient way to defend against these online and offline attacks by forcing users to choose a strong password. This results in the user's account being logged into only if the password strength from all algorithms is strong.

CHAPTER 6

EXPERIMENTAL RESULTS

6.1 Performance Evaluation

Evaluation metrics are used to measure the quality of the machine learning model. Evaluating machine learning models or algorithms is essential for any project. By using evaluation metrics, we can ensure that the model is operating correctly and optimally.

It is very important to use multiple evaluation metrics to evaluate your model. This is because a model may perform well using one measurement from one evaluation metric, but may perform poorly using another measurement from another evaluation metric. There are many different types of evaluation metrics available to test a model. They are confusion matrix, accuracy, precision, recall, F1-score,false positive rate, Receiver operator characteristics curve (ROC), Precision- Recall (PR) curve, Logarithmic loss, mean absolute error and root mean squared error.

In this project, we include confusion matrix and classification accuracy and classification report.

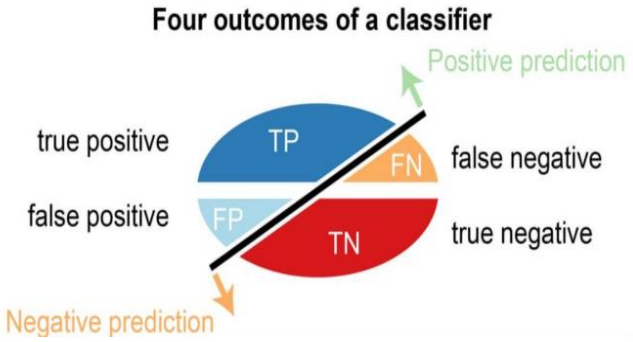
A confusion matrix gives us a matrix as output and describes the complete performance of the model. Classification of a test dataset produces four outcomes – true positive, false positive, true negative, and false negative.

True positive (TP): correct positive prediction

False positive (FP): incorrect positive prediction

True negative (TN): correct negative prediction

False negative (FN): incorrect negative prediction




```
In [84]: #predicting user input
         print("Give atleast 8 characters")
         inp = [input()]
         #print(len(inp[0]))
         if(len(inp[0]) > 7):

             inp = vectorizer_state.transform(inp)
             res=clf.predict(inp.todense())
             if res[0] == 0:
                 print("your password was : Weak")
                 print("Warning!!!! Please updtae the password with Special characters alpha numericals :(")
             elif res[0] ==1:
                 print("Your password was : Moderate")
                 print("Meets minimum stands advised to have a strong password :)")
             else :
                 print("Your password was : Strong :>")
         else:
             print("password length was less than 8 characters please check")
```

```
Give atleast 8 characters
saikumar23
Your password was : Moderate
Meets minimum stands advised to have a strong password :)
```

Displaying all test and train accuracies

```
In [44]: from prettytable import PrettyTable
         x = PrettyTable()
         x.field_names = ["S.no", "Model", "Train_accuracy", "Test_accuracy"]
         x.add_row(['1', 'Naive Bayes', accuracy_score(y_train, y_train_pred_naive), accuracy_score(y_test, y_test_pred_naive)])
         x.add_row(['2', 'SVM', accuracy_score(y_train, y_train_pred_SVM), accuracy_score(y_test, y_test_pred_SVM)])
         x.add_row(['3', 'MLP', accuracy_score(y_train, y_train_pred_MLP), accuracy_score(y_test, y_test_pred_MLP)])
         x.add_row(['4', 'Decision Tree', accuracy_score(y_train, y_train_pred_dect), accuracy_score(y_test, y_test_pred_dect)])
         print(x)
```

```
+------+---------------+----------------+---------------+
| S.no |     Model     | Train_accuracy | Test_accuracy |
+------+---------------+----------------+---------------+
|  1   |  Naive Bayes  |   0.9996875    |     0.728     |
|  2   |      SVM      |    0.999875    |     0.9155    |
|  3   |      MLP      |     0.99975    |    0.91325    |
|  4   | Decision Tree |    0.739375    |     0.7305    |
+------+---------------+----------------+---------------+
```

6.2 Summary

The proposed model is efficient than the other existing methodology in terms of simplicity and the ML models that are used.

CHAPTER 7

CONCLUSION

7.1 Conclusion

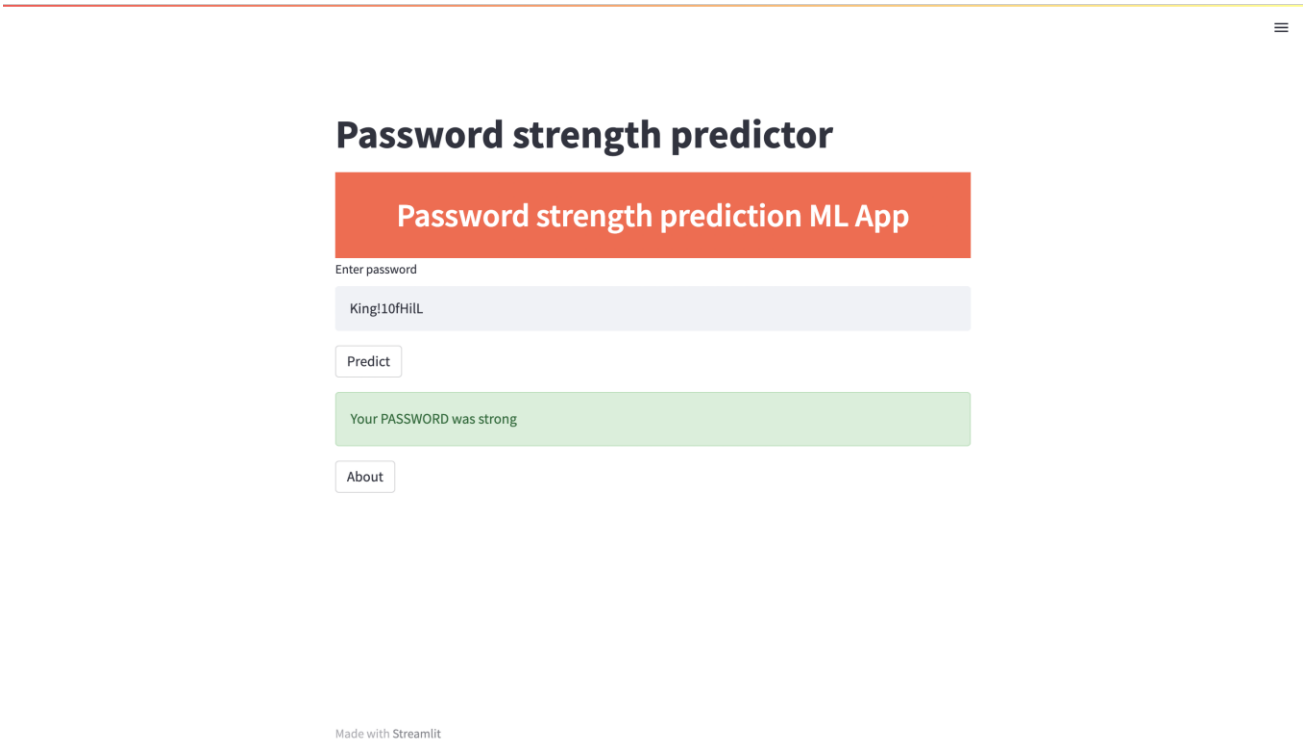
The password strength prediction is modelled as a classification task in this paper, and the machine learning approach for predicting the password strength is described. The password strength analysis model was trained using the Nave Bayes classifier, Decision tree classifier, Multilayer perceptron, and Support Vector Machine. To make training and implementation easier, features were extracted from a database of 669880 passwords from various categories. The model's performance was assessed using 10-fold cross validation, and it was discovered that the SVM classifier trained with the RBF kernel performed effectively. Machine learning learns on its own, using intelligence cues from the training data to produce correct predictions. The outcomes of putting the human-generated data to the test Passwords are checked for strength using a variety of popular tools. The proposed model is also confirmed by thee tools may could be employed for providing a highly secure environment.

APPENDIX I

I SAMPLE CODE

```
--#predicting user input
print("Give atleast 8 characters")
inp = [input()]
#print(len(inp[0]))
if(len(inp[0]) > 7):
    inp = vectorizer_state.transform(inp)
    res=clf.predict(inp.todense())
    if res[0] == 0:
        print("your password was : Weak")
        print("Warning!!!! Please updtae the password with Special characters alpha
numericals :(")
    elif res[0] ==1:
        print("Your password was : Moderate")
        print("Meets minimum stands advised to have a strong password :)")
    else :
        print("Your password was : Strong :>")
else:
    print("password length was less than 8 characters please check")
```

II SNAPSHOTS



III REFERENCE

[1] F Bergadano, B.Crispo, G.Ruffo, “Proactive password checking with decision trees”,Proc. of the 4th ACM conference on computer and communications security, Zurich, Switzerland, 1997, pp 67-77

[2] Giancarlo Ruffo, Francesco Bergadano, “EnFilter : A Password Enforcement and Filter Tool Based on Pattern Recognition Techniques”, Springer Berlin / Heidelberg, 1611-3349 (Online),Volume 3617/2005

[3] Dell'amico, Matteo, Michiardi, Pietro and Roudier, Measuring Password Strength: An Empirical Analysis., Yves. Jul 20, 2009.

[4] Forget,Alain, Improving text passwords through persuasion, et al. s.l. :ACM, 2008. pp. 1--12.

[5] Narayanan, Arvind and Shmatikov, Vitaly. Fast dictionary attacks on passwords using time-space tradeoff. s.l. : ACM,
2005. pp. 364--372.

[6] Ur, P. G. Kelley, S. Komanduri, M. M. J. Lee, M. Mazurek, T. Passaro, R. Shay, T. Vidas and L. Bauer, "How does your password measure up? the effect of strength meters on password creation," in Proc. USENIX SEC 2012, 2012

[7] Wang, Z. Zhang, J. Y. P. Wang and X. Huang, "Targeted online password guessing: an underestimated threat," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security,Vienna Austria, 2016.

[8] Florencio, C. Herley and P. C. V. Oorschot, "Password portfolios and the finite-effort user: Sustainably managing large numbers of accounts," in Proc. USENIX SEC 2014, 2014.

[9] Ma, W. Yang, M. Luo and N. L, "A study of probabilistic password models," in 2014 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2014.

[10] L. Gong-Shen, Q. Wei-Dong and L. Jian-Hua, "Password vulnerability assessment and recovery based on rules mined from large-scale real data," Chinese Journal of Computers, vol. 39, no. 3, p. 454–467, 2016.

[11] D.Wang and P.Wang, "The emperor’s new password creation policies," in in Proc. ESORICS 2015, 2015.

[12] M. Weir, S. Aggarwal, M. Collins and H. Stern, "Testing metrics for password creation policies by attacking large
sets of revealed passwords," in Proc.ACM CCS 2010, 2010.