NATIONAL INSTITUTE OF TECHNOLOGY

HAZRATBAL, SRINAGAR

PROJECT REPORT

ON

A HYBRID PASSWORD STRENGTH ANALYSIS TOOL BASED ON MACHINE LEARNING

Submitted in partial fulfilment of the requirement
For the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Aquif Reza Mir, Jamyang Lotus

Under the guidance of

Dr Shaima Qureshi

CERTIFICATE

This is to certify that the Project titled "**A Hybrid Password Strength Analysis Tool Based on Machine Learning and Rule-Based Pattern Tests**" has been prepared by **Aquif Reza Mir**, Enrollment No: **2017BCSE023** and **Jamyang Lotus**, Enrollment No: **2017BCSE017** under my guidance in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering, National Institute of Technology Srinagar during the academic session June 2021. The work has not been submitted in any form for the award of a degree in any other institute.

Dr Shaima Qureshi
Project Guide
Assistant Professor
Department of Computer Science & Engineering
National Institute of Technology, Srinagar

STUDENT DECLARATION

We, **Aquif Reza Mir** Enrollment No: **2017BCSE023** and **Jamyang Lotus** Enrollment No: **2017BCSE017**, declare that the work which is being presented in the Project titled "**A Hybrid Password Strength Analysis Tool Based on Machine Learning and Rule-Based Pattern Tests**" in the partial fulfilment of the requirements for the award of "Bachelor of Technology in Computer Science and Engineering" degree in the session June 2021, is an authentic record of our own work carried out under the supervision of **Dr Shaima Qureshi**, Assistant Professor, Department of Computer Science and Engineering, National Institute of Technology, Srinagar.

The matter embodied in this project has not been submitted by us for the award of any other degree.

Dated: 9 June 2021

ACKNOWLEDGEMENT

We would like to take this opportunity to express our sincere gratitude to our mentor for this project, **Dr Shaima Qureshi**, Asst. Prof. Department of Computer Science and Engineering for their patience while guiding us and providing us with advice and constructive criticism throughout the course of our project.

Furthermore, we would also like to extend our thanks to our **H.O.D Dr Roohie Naaz** for their active support and guidance.

ABSTRACT

With the advent of digital-age passwords have become the most prevalent authentication scheme. This is because of their superior versatility and deployability in comparison to other authentication mechanisms. Unfortunately, humans still struggle with creating strong passwords that are per the guidelines listed in standard password policies. Users, in general, are reluctant to follow these policies and thus end up creating passwords that contain predictable patterns or reuse the same password across multiple platforms which in turn significantly reduces the work of hackers. This behaviour establishes the need for password analysis tools that users can utilize to determine the strength of their passwords. This project report explores the use of machine learning algorithms to reduce the computation needs of traditional rule-based password analysis tools and to make them more flexible.

CONTENTS

1 Introduction

In today's world, with the ever-increasing computation power of processors, utilization of graphical processing units for password cracking, and development of efficient and effective password cracking algorithms, the secureness of password policies which were initially designed to be resistant to simple brute-force attacks is questionable. It is a fact that many tech giants with sufficient resources have dealt with this issue by using advanced slow hashing algorithms such as bcrypt, PBKDF2 e.t.c to keep their password database secure. However, this does not solve the issue at hand as smaller firms and startups are usually the ones at the receiving end of such password cracking hacks. These companies due to lack of sufficient resources still use common MD5-based hashing algorithms(e.g SHA1). Therefore, there is a need for the development of password analysis tools that can quantify the strength of passwords. People can use these tools to make their passwords secure in case the company's password database is hacked and then leaked by the hacker.

Many rule-based password strength classifying tools have already been developed and are available on the internet. These tools check for various patterns in a given password to determine its strength(e.g What is the length of password?, What is the key-space of password?, Does the password contain any dictionary word? e.t.c). However, with each advance in password cracking algorithms used by the hackers, new rules need to be added to these rule-based password analyzers to ensure the measurement of password strength given by them is valid. This repeated addition of pattern testing rules has already made modern rule-based password analysis tools computationally intensive and will continue to do so as long as the passwords keep on getting more and more complex which is certain. One way to solve this issue is to introduce Machine Learning components into these password analysis tools. These Artificial Intelligence components will cut the computational needs of these tools and prevent them from further increasing provided they are trained with the latest passwords.

This project aims to develop a hybrid tool based on this philosophy which combines the flexibility of Machine Learning algorithms with traditional rule-based pattern testing algorithms to create a password analyzer that is capable of improving itself provided it is trained with the latest passwords.

2 Background and Motivation

In recent years many Machine Learning based password strength analysis tools have been developed. Most of these software tools use various classification algorithms such as Logistic regression, Ensemble Learning(Decision tree learning) e.t.c to create models that classify a given password by assigning a specific label value to it based on its complexity. The class prediction models are created by training these classification algorithms using word-lists of real user passwords. These password lists are leaked over the internet by hackers after successfully hacking into various company databases. E.g's of these password lists include rockyou(2009 breach of online games service RockYou, 14 million leaked passwords), hashkiller, 000webhost e.t.c.

After analysis it has been found that these prediction models used factors such as length of password, key-space, intermixing of characters e.t.c to define the complexity of passwords. An example of a password class prediction model using Logistic regression[1] is as follows:

```
import numpy as np
import random
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

def getTokens(inputString): #custom tokenizer. ours tokens are
characters rather than full words
  tokens = []
  for i in inputString:
    tokens.append(i)
  return tokens


filepath = 'data.csv' #path for password file
data = pd.read_csv(filepath,',',error_bad_lines=False)

data = pd.DataFrame(data)
passwords = np.array(data)
```

```
random.shuffle(passwords) #shuffling randomly for robustness
y = [d[1] for d in passwords] #labels
allpasswords= [d[0] for d in passwords] #actual passwords

vectorizer = TfidfVectorizer(tokenizer=getTokens) #vectorizing
X = vectorizer.fit_transform(allpasswords.astype('U'))

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)  #splitting

lgs = LogisticRegression(penalty='l2',multi_class='ovr')  #our
logistic regression classifier
lgs.fit(X_train, y_train) #training
print((lgs.score(X_test, y_test)))  #testing

#more testing
X_predict =
['faizanahmad123','faizanahmad##','ajd1348#28t**','fffffffff'
,'kuiqwasdi','uiquiuiiuiuiuiuiuiuiuiuiui','mynameisfaizan','my
nameis123faizan#','faizan','123456','abcdef']
X_predict = vectorizer.transform(X_predict)
y_Predict = lgs.predict(X_predict)
print(y_Predict)
```

The prediction results of this classifier are as follows:

```
   faizanahmad123 [medium]
   faizanahmad## [medium]
   ajd1348#28t** [strong]
   fffffffff [easy]
   kuiqwasdi [medium]
   uiquiuiiuiuiuiuiuiuiuiuiui [easy]
   mynameis123faizan# [strong]
   faizan [easy]
   abcdef [easy]
```

These password classifiers based purely on Machine Learning classification algorithms have been very successful(> 75% success rate) in classifying passwords. However, there is an inherent issue with these models. They can only assign a strength class to a given password i.e they can only tell us whether our password is strong/weak or strong/satisfactory/weak. These models cannot give us the measurement of the strength of a password in terms of real numbers or integers. To deal with this issue many attempts at regression-based prediction models have been made but due to the amount of randomness, a password can exhibit these models are prone to overfitting and hence give bad prediction results.

In addition to these classification models, some attempts have been made to develop password strength classifiers based on clustering algorithms(Kernel Density Estimation(KDE) [7]). This model explores the clustering of passwords in the space whose dimensions are defined by structural features extracted from each password, which are transformed and down-selected using Principal Component Analysis (PCA).

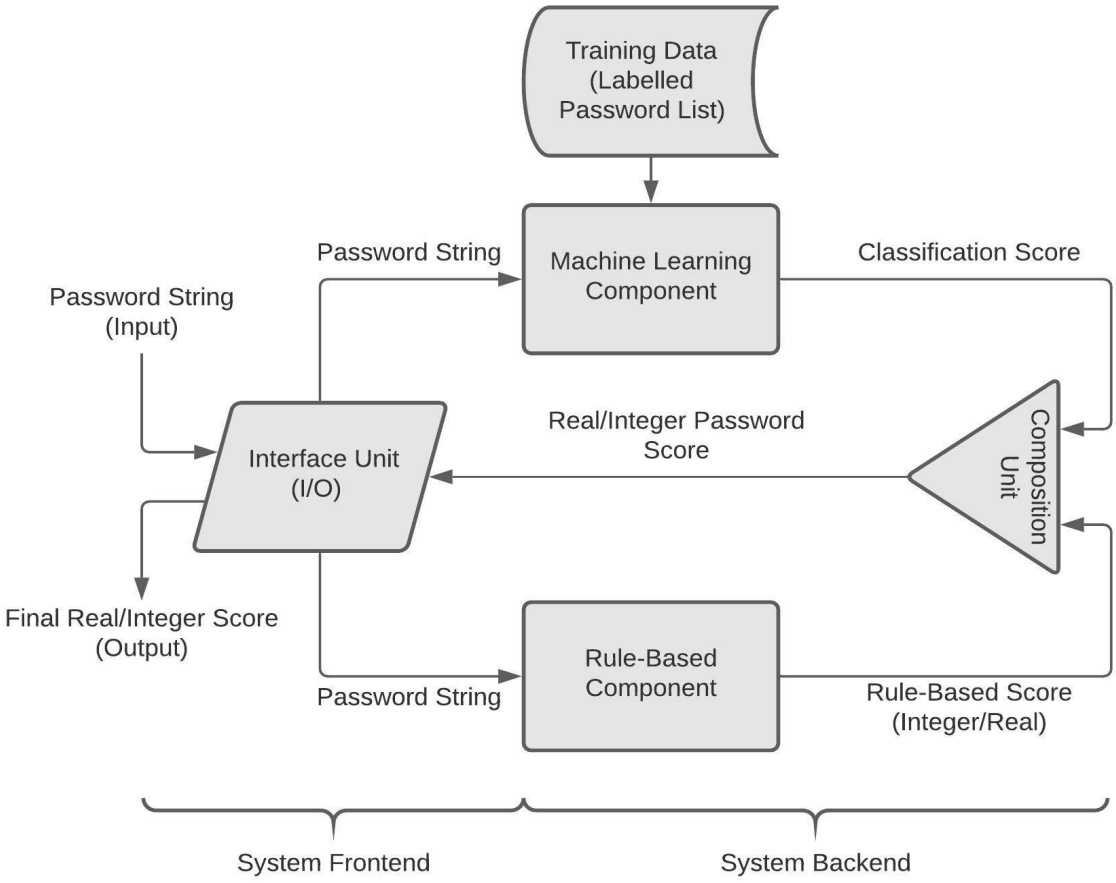
Considering the above-mentioned issues, it is clear the problem statement is to effectively inculcate the classification algorithms into our analysis tool without compromising the success rate of prediction results.

3 Approach

This project report investigates the possibility of leveraging machine learning classification algorithms in the future to cut short most computation-intensive rule-based pattern testing algorithms in traditional password strength analyzers. This hybrid amalgam will at least comprise of the following four components:

- A Machine Learning component utilizing classification algorithms to assign a base class to a password based on a pre-trained model.
- A light-weight Rule-based component using specific pattern testing algorithms to assign a real-value/integer score to a password.
- A Composition/Integration unit that combines the scores of the aforementioned two components using a specific strategy to generate the final normalized score allocated to the password.
- An Interface unit for password input and score output.

3.1 Block-Diagram



3.2 Machine Learning Component

The machine learning component essentially comprises class prediction models created by training various classification algorithms using password lists leaked over the internet. Some of these classification algorithms are as follows:

- Logistic Regression
- K-Nearest Neighbour(KNN) classification
- Support Vector Machine(SVM)
- Naive Bayes
- Decision Tree classification
- Ensemble Learning(Random Forest classification)

These classification algorithms use different strategies to classify passwords with each strategy being based on or more classification parameters. Some of these parameters are Cross-Entropy, Cosine Similarity, Information Gain, Gini Index, Gini Ratio, Chi-Square values e.t.c.

3.2.1 Data-set Description

In this project, as has been already mentioned the data-sets we are using are leaked password lists. However, there is a big issue with these lists that have not been addressed. These lists comprise millions of passwords but these passwords have not been classified. In other words, our current data-set comprises passwords without any strength label/class assigned to them.
Now the question remains, What medium should we use to classify these passwords in our data-set such that the results of classification are acceptable by current standards. After all, if the passwords in the training data-set have not been properly labelled then the prediction models created from these data-sets will also give bad results. Here we have used a tool called PARS(Password Analysis and Research System)[2][3] developed at Georgia Tech University. This tool has most of the commercial **Rule-based** password strength classifiers integrated into it. After feeding it the password list we got separate classification results of each of the classifiers integrated into this tool. Each of the classifiers sorted the passwords into three classes(0-Weak, 1-Average, 2-Strong). We then took the intersection of these classification results to obtain our final data-set. In other words, we only took those passwords for which all classifiers were in agreement (i.e

passwords which got the same label from all classifiers). An example of the resulting data-set is as shown in fig(i).

password	strength
kzde5577	1
kino3434	1
visi7k1yr	1
megzy123	1
lamborghin1	1
AVYq1lDE4MgAZfNt	2
u6c8vhow	1
v1118714	1
universe2908	1
as326159	1
asv5o9yu	1
612035180tok	1
jytifok873	1
WUt9IZzE0OQ7PkNE	2
jerusalem393	1
g067057895	1
52558000aaa	1
idofo673	1
6975038lp	1
sbl571017	1
elyass15@ajilent-ci	2
intel1	0
klara-tershina3H	2
czuodhj972	1
faranumar91	1
cigicigi123	1

fig(i)

3.2.2 Feature Matrix Modification

As of now, our data-set looks as shown in fig(i). The feature matrix contains 1 column(passwords) and a dependent vector(strength) which classifies passwords in 3 labels(0-Weak, 1-Average, 2-Strong). Now if we were to use this data-set to train our classification model, we would get bad prediction results. This is because our only column in the feature matrix(passwords) has a huge domain. This will continue to be true even if we restrict the length of passwords to a specific range because of the permutation of characters in the string.

Now to deal with this issue, instead of taking the entire string of passwords as a token, we take individual characters inside the password string as tokens[1]. In this way, we have a feature matrix containing multiple columns with each column containing only a single character. Thereby reducing its domain of values. After carrying out this modification we have our final data-set as shown in fig(ii).

Password								Strength
k	z	d	e	5	5	7	7	1

fig(ii)

The code of the Machine Learning component developed in this project is shown in **APPENDIX-1**.

3.2.3 Technologies used for implementation

- Python Libraries:
    1. Sci-kit-Learn
    2. Numpy
    3. Pandas

- Mysql for manipulating database containing training data-set

3.3 Rule-Based Component

The rule-based component is a light-weight unit that contains a collection of tests that have been designed and manually programmed by the developer. Each of these tests check a given password based on certain criteria. Some of these criteria are listed as follows:

- What is the length of the password?
- Does the password contain letters/numbers only?
- How many repeated characters are there in the password?
- How many consecutive uppercase/lowercase letters are there in the password?
- How many consecutive numbers/special characters are there in the password?
- In addition to these common pattern tests, there are specific tests that check whether the given password is based on a specific mask.

The password string is evaluated by these tests and then assigned a real/integer score based on the number and type of tests in which it gets the desired results. The type of tests plays a role because not all tests have the same contribution towards the overall score of the password i.e different tests have different weightage.

There is a reason for having this light-weight rule-based component in addition to the machine learning model. The machine learning model assigns a specific class to a password based on its complexity. However, there are password strings such as “momokids@15”, “momof3g8kids” e.t.c that might seem complex based on string length, key-space e.t.c but these passwords can be easily cracked using Mask Attacks[4], Rule-based Attacks[5], Combinator Attacks[6] e.t.c. The machine learning model will sort these passwords in Average or Strong classes which will certainly be a wrong prediction. Therefore we need this rule-based component to perform pattern-based tests aimed to detect these types of passwords. The code of the Machine Learning component developed in this project is shown in APPENDIX-2.

3.3.1 Technologies used for implementation

- Python Libraries:
    1. Numpy

3.4 Composition/Integration Unit

The Composition unit takes the classification score of a given password allocated by the machine learning unit and the integer/real score of the rule-based unit as its inputs. It combines these scores using a specific strategy to generate the final real/integer normalized score assigned to the given password. The code of the Machine Learning component developed in this project is shown in **APPENDIX-3**.

3.4.1 Technologies used for implementation

- Python Libraries:
    1. Numpy

3.5 Interface Unit

The function of the interface unit is to take password inputs from users and output final scores assigned to the password. In this project we are using a web application interface i.e we are hosting our analysis tool on a web server. The user will have access to this tool via a website URL. The users will input the password strings in the appropriate section of the web-page which will be captured and then forwarded to the back-end of the website where the running password analysis tool will assign a score to this string. This score will then be transmitted to the website front-end and will be displayed on the web-page. The code of the Machine Learning component developed in this project is shown in **APPENDIX-4**.

3.5.1 Technologies used for implementation

- Website Backend:
    1. Flask framework
- Website Frontend:
    1. HTML
    2. CSS and CSS framework (Bootstrap)
    3. Javascript

4 Project Results

At the end of this project, we can try to analyse the results of this project along three major axes:

- Accuracy
- Flexibility
- Processing Time

4.1 Accuracy

During our research, the Machine Learning based password analysis tools[1] that we came across had an accuracy of about 80-81% and that too against pre-processed test data-set. In comparison to test the accuracy of our password analysis tool we took two major steps:

1. Firstly we used two standard password analysis tools available on the market(NordPass[8], LastPass[9]).
2. Secondly, we used a third party reliable random password generator tool(PasswordGenerator+[10]).

Since each of the standard password analyzers that we used works slightly differently, we carried out a separate comparative analysis of each of them against our tool.
In our trials, we generated 100 random passwords using the aforementioned password generator and then compared the results.

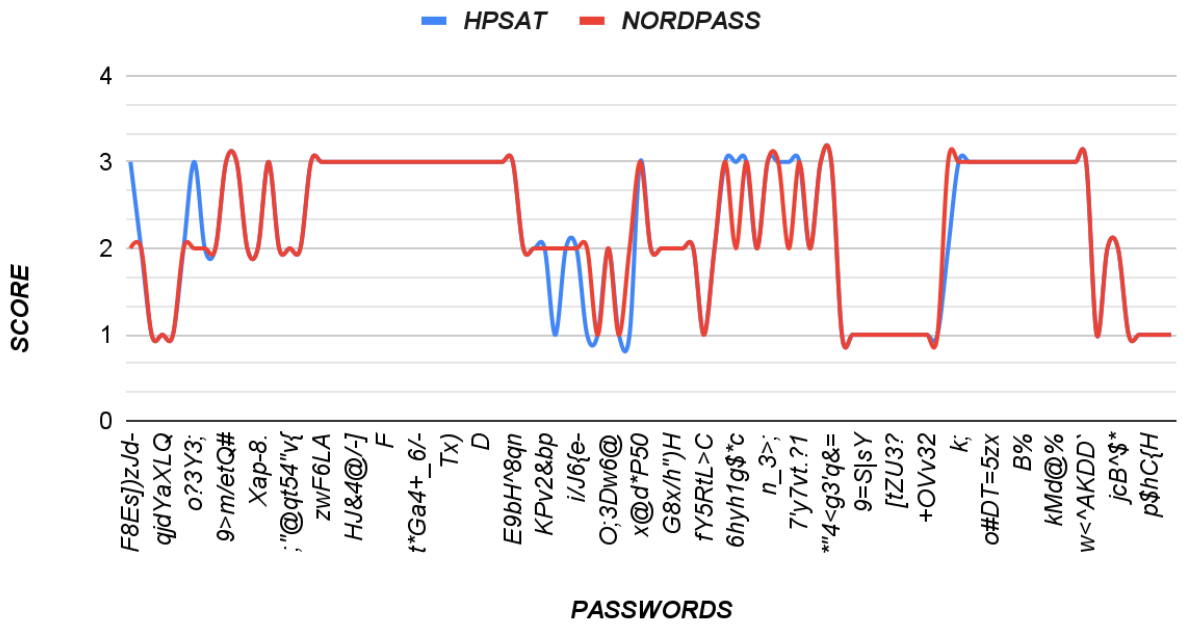
4.1.1 HPSAT VS NordPass

PASSWORD	HPSAT	NORDPASS
F8Es])zJd-	3	2
hM-J2ZmJbB	2	2
cpadintsqb	1	1
qjdYaXLQWh	1	1
wTZbxeNbCL	1	1
M287feXaHG	2	2
o?3Y3;c<Na	3	2
c0&c&vF,W7	2	2
Hn!XvcH9L@	2	2
9>m/etQ#F`RY	3	3
&YB[eY=F*2-d	3	3
R3jSh9Mr>)	2	2
Xap-8.dKB^	2	2
z4NnVSM$+DY	3	3
2#/=[a+,	2	2
;"@qt54"v{	2	2
6c;m(y!q[r	2	2
q!msJTCA2r"j^X{V	3	3
zwF6LA[HD+2;qfSy	3	3
bXfn<.4[7;-y&2m+	3	3
zRD&)g4MWPSx.dup	3	3
HJ&4@/-]ajWY6.S	3	3
EuS&Xqz@xT"J52CP	3	3
qZGD^8k"Q7XdY~LW	3	3
F{~m/CSrt9w]&^Ug	3	3
T(y7GugP53=DHeLz	3	3
QTj`>?qS);b!U9Vp	3	3
t*Ga4+_6/-e39mC`	3	3
J4f-dH7QgL$&<9?	3	3
Xt6UgH=8<{4;Ly95	3	3
Tx)nVFm*fhs?%c2&	3	3
M!/sh;6R=>VPq{xc	3	3
CR}c9J]Muxp=2(^t	3	3
D(KcH=yvQF:&-9`Z	3	3
W)C~T&9pS+j5]=H	3	3
T5prc2)8B"{<'wY[	3	3
E9bH^8qnSGDA~+jv	3	3
AR9q!|#	2	2
J7n)4X>	2	2
KPv2&bp	2	2
C5%Ma:R	1	2
h;SA"2^	2	2

i/J6{e-	2	2
X{f20kz	1	2
zB4ot"R	1	1
O;3Dw6@	2	2
KO!j2"t	1	1
PUIS<8O	1	2
x@d*P50%	3	3
mf'5+JPZ	2	2
PwOF1#vs	2	2
G8x/h")H	2	2
W`1m(&4H	2	2
Lv-lw2=0	2	2
fY5RtL>C	1	1
CZ4;6qHs	2	2
L%+)3fn0	3	3
6hyh1g$*c6	3	2
m*;fkb&53[	3	3
j>>d_pm9}a	2	2
n_3>;ayozm	3	3
r)b;-5qm=x	3	3
0klg9&y:)d	3	2
7'y7vt.?1	3	3
9t9o%)lt2e	2	2
($4_]6aw8g	3	3
*"4<g3'q&=	3	3
7FA`uO	1	1
2Msu'g	1	1
9=S|sY	1	1
;8puJu	1	1
X|4llb	1	1
[tZU3?	1	1
h2Dx!3	1	1
m&O6w{	1	1
+OVv32	1	1
J?t)6}	1	1
Qt`l{3qQF7*<	2	3
k;J=lZ=2*WZ;	3	3
xi"rt>7(iK<i	3	3
i.@3=&j7GNA+	3	3
o#DT=5zx<i7q	3	3
dc'bX0=<g3[-	3	3
XIG0vp21xN^]	3	3
B%5A>7^w+"~r	3	3

g5?T!46u^W0Q	3	3
g*tW;[(4Rs#A	3	3
kMd@%bOC&%k.	3	3
VOr_=x*^]ks	3	3
udQgRW%<=fHm	3	3
w<^AKDD`u=M)	3	3
N>bHfF	1	1
)S(!l&	2	2
jcB^$*	2	2
vhCl#F	1	1
Kqs$r[	1	1
p$hC{H	1	1
IR[hGX	1	1
r)UXra	1	1

HPSAT VS NORDPASS



Upon analysis, we found out our password analyzer had an accuracy of about 92% when compared against the NordPass password analysis tool.

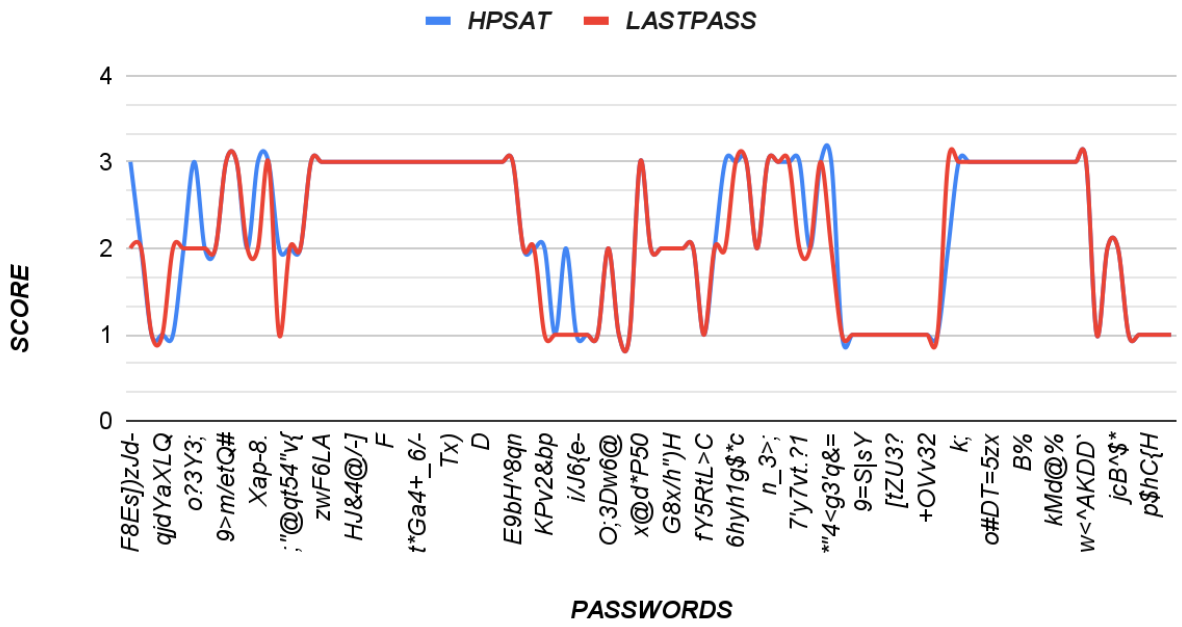
4.1.2 HPSAT VS LastPass

PASSWORD	HPSAT	LASTPASS
F8Es])zJd-	3	2
hM-J2ZmJbB	2	2
cpadintsqb	1	1
qjdYaXLQWh	1	1
wTZbxeNbCL	1	2
M287feXaHG	2	2
o?3Y3;c<Na	3	2
c0&c&vF,W7	2	2
Hn!XvcH9L@	2	2
9>m/etQ#F`RY	3	3
&YB[eY=F*2-d	3	3
R3jSh9Mr>)	2	2
Xap-8.dKB^	3	2
z4NnVSM$+DY	3	3
2#/=[a+,	2	1
;"@qt54"v{	2	2
6c;m(y!q[r	2	2
q!msJTCA2r"j^X{V	3	3
zwF6LA[HD+2;qfSy	3	3
bXfn<.4[7;-y&2m+	3	3
zRD&)g4MWPSx.dup	3	3
HJ&4@/-]ajWY6.S	3	3
EuS&Xqz@xT"J52CP	3	3
qZGD^8k"Q7XdY~LW	3	3
F{~m/CSrt9w]&^Ug	3	3
T(y7GugP53=DHeLz	3	3
QTj`>?qS);b!U9Vp	3	3
t*Ga4+_6/-e39mC`	3	3
J4f-dH7QgL$&<9?	3	3
Xt6UgH=8<{4;Ly95	3	3
Tx)nVFm*fhs?%c2&	3	3
M!/sh;6R=>VPq{xc	3	3
CR}c9J]Muxp=2(^t	3	3
D(KcH=yvQF:&-9`Z	3	3
W)C~T&9pS+j5]=H	3	3
T5prc2)8B"{<'wY[	3	3
E9bH^8qnSGDA~+jv	3	3
AR9q!|#	2	2
J7n)4X>	2	2
KPv2&bp	2	1
C5%Ma:R	1	1
h;SA"2^	2	1

i/J6{e-	1	1
X{f20kz	1	1
zB4ot"R	1	1
O;3Dw6@	2	2
KO!j2"t	1	1
PUIS<8O	1	1
x@d*P50%	3	3
mf'5+JPZ	2	2
PwOF1#vs	2	2
G8x/h")H	2	2
W`1m(&4H	2	2
Lv-lw2=0	2	2
fY5RtL>C	1	1
CZ4;6qHs	2	2
L%+)3fn0	3	2
6hyh1g$*c6	3	3
m*;fkb&53[	3	3
j>>d_pm9}a	2	2
n_3>;ayozm	3	3
r)b;-5qm=x	3	3
0klg9&y:)d	3	3
7'y7vt.?1	3	2
9t9o%)lt2e	2	2
($4_]6aw8g	3	3
*"4<g3'q&=	3	2
7FA`uO	1	1
2Msu'g	1	1
9=S|sY	1	1
;8puJu	1	1
X|4llb	1	1
[tZU3?	1	1
h2Dx!3	1	1
m&O6w{	1	1
+OVv32	1	1
J?t)6}	1	1
Qt`l{3qQF7*<	2	3
k;J=lZ=2*WZ;	3	3
xi"rt>7(iK<i	3	3
i.@3=&j7GNA+	3	3
o#DT=5zx<i7q	3	3
dc'bX0=<g3[-	3	3
XIG0vp21xN^]	3	3
B%5A>7^w+"~r	3	3

g5?T!46u^W0Q	3	3
g*tW;[(4Rs#A	3	3
kMd@%bOC&%k.	3	3
VOr_=x*^]|ks	3	3
udQgRW%<=fHm	3	3
w<^AKDD`u=M)	3	3
N>bHfF	1	1
)S(!l&	2	2
jcB^$*	2	2
vhCI#F	1	1
Kqs$r[	1	1
p$hC{H	1	1
IR[hGX	1	1
r)UXra	1	1

HPSAT VS LASTPASS



Upon analysis, we found out our password analyzer had an accuracy of about 89% when compared against the LastPass password analysis tool.

Thus overall accuracy of our password strength analysis tool based on these trials comes out to be 90.5%.

4.2 Flexibility

One of the main achievements of our password strength analysis tool is its flexibility. For e.g: Suppose a hacker develops a new technique of cracking passwords, now in a traditional password analyzer new pattern tests would have to be introduced to act as a filter to look for passwords matching the ones crackable by the newly developed technique. However, in our case, all we have to do is to feed new knowledge data(i.e a number of passwords that are cracked by the new technique) into the Machine Learning component and our tool will adjust its model accordingly. Thus we do not have to increase the size of our tool each time a password cracking technique is developed.

4.3 Processing Time

In a traditional password strength analysis tool, to determine the strength of a given password, it is tested against numerous rule-based pattern tests to ascertain its strength. However, our password analysis tool utilizes a somewhat different strategy. Initially, the machine learning component takes the password and carries out a class-based prediction based on its model. Next, a few custom-designed pattern tests are run to create a final normalized score of password. The absence of a large number of pattern tests greatly reduces the processing time of determining the strength of the password. Although while training our machine learning component takes a large amount of time, it only makes simple prediction while it is testing a given password for strength estimation.

5 Conclusions and Future Work

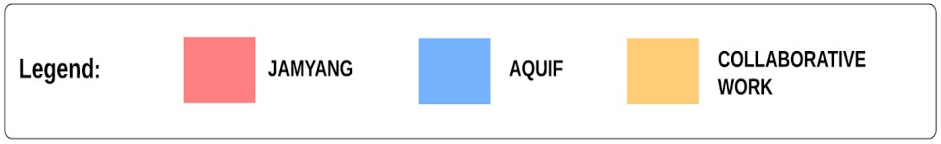
With all the accumulated effort invested in this project we summarize the conclusions with respect to the main objectives of the project, namely, accuracy, flexibility and processing time.

- **Accuracy:** This is a major obstacle for the acceptance of this technique in production environments. However through rigorous comparative analysis we have shown that our password analysis tool has an average accuracy of about **90.5%** which is much higher than the accuracy of ML based password analysis tools**(80-81%)**[1] that we encountered in our research.
- **Flexibility:** We have already shown it in our project results that our password analyser is far more flexible than any traditional password analyser because all we need to do to modify our tool is to train the ML component with the updated knowledge data.
- **Processing Time:** Since we have replaced most of the computation intensive pattern tests in traditional password analyzers with a Machine Learning component which only carries out class based predictions while testing passwords. This cuts short most of the processing time and makes our tool very light-weight.

Now in the domain of this project there is an active chance of future work to further increase the **accuracy** of the analyzer. This can be done by further experimenting with various classification and clustering techniques to create efficient machine learning components which give better classification results on their own and rely less on rule-based tests.

Project-Timeline Group-9

	October	November	December	January	February	March	April	May	June
Project Research & Planning									
Machine Learning Component Development									
Rule-Based Component Development									
Composition Unit Development & Testing									
Interface Unit Develpment & Unit Testing									
Interface Integration									
Overall Project Testing & Debugging									



REFERENCES

[1] Ahmed, Faizan (2016). Machine Learning based Password Strength Classification. Available from World Wide Web: https://medium.com/@faizann20/machine-learning-based-password-strength-classification-7b2a3c84b1a6

[2] E.g of PARS processed dataset. Available from World Wide Web: https://www.kaggle.com/bhavikbb/password-strength-classifier-dataset

[3] Ji, S., Yang, S., Wang, T., Liu, C., Lee, W., & Beyah, R. (2015). PARS: A Uniform and Open-source Password Analysis and Research System. ACSAC 2015. Available from World Wide Web: https://www.semanticscholar.org/paper/PARS%3A-A-Uniform-and-Open-source-Password-Analysis-Ji-Yang/99b9bff925fdab9c4680f5667e7e6bc93cde2980

[4] Mackay, W. (2016). How to Perform a Mask Attack Using hashcat. Available from World Wide Web: https://www.4armed.com/blog/perform-mask-attack-hashcat/

[5] Mackay, W. (2016). How to Perform a Rule-Based Attack Using hashcat. Available from World Wide Web: https://www.4armed.com/blog/hashcat-rule-based-attack/

[6] Mackay, W. (2016). How to Perform a Combinator Attack Using hashcat. Available from World Wide Web: https://www.4armed.com/blog/hashcat-combinator-attack/

[7] Todd, M. (2016). An Investigation of Machine Learning for Password Evaluation. Available from World Wide Web: https://www.semanticscholar.org/paper/An-

Investigation-of-Machine-Learning-for-Password-Todd/d40dee322446d8a3e45a622e1cf80f990976627c

[8] NordPass password strength checker.
Available from World Wide Web:
https://nordpass.com/secure-password/

[9] LastPass password strength checker.
Available from World Wide Web:
https://lastpass.com/howsecure.php

[10] Secure Password Generator(PasswordGenerator+)
Available from World Wide Web:
https://passwordsgenerator.net/plus/

APPENDIX

1 - Machine Learning Component(Code-Snippet):

```
import numpy as np
import random
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

def getTokens(inputString):
    tokens = []
    for i in inputString:
        tokens.append(i)
    return tokens

filepath = '../ml/data_5.csv' #path for password file
data = pd.read_csv(filepath,',',error_bad_lines=False)
data.isnull().any()

data = pd.DataFrame(data)

X = data.iloc[:, 0].values
y = data.iloc[:, 1].values

vectorizer = TfidfVectorizer(tokenizer=getTokens) #vectorizing
X = vectorizer.fit_transform(X)

lgs = LogisticRegression(penalty='l2',multi_class='ovr', max_iter=1000)
#our logistic regression classifier
lgs.fit(X, y) #training


class ml_passwordstrength:
```

```
    def __init__(self, password):
        self.password = []
        self.password.append(password)

        self.score = self.__classification_score()[0] + 1


    def __classification_score(self):
        vect_password = vectorizer.transform(self.password)
        return lgs.predict(vect_password)

    def get_ml_score(self):
        return self.score
```

2 - Rule-Based Component(Code-Snippet):

```
import os
import string
# Only load it once.
with open(os.path.join(os.pardir, "english"), 'r') as f:
    words = set(f.read().split('\n'))

class passwordstrength:
    def __init__(self, password):
        self.words = words

        self.password = password
        self.password_length = len(password)
        self.strength_flags = {
            "length": 0,
            "upper_case": 0,
            "lower_case": 0,
            "digits": 0,
            "special_chars": 0,
            "middle_num_symbol": 0,
            "extra": 0,
            "letters_only": 2,
            "numbers_only": 2,
            "repeating_chars": 2,
            "consecutive_upper_case": 2,
            "consecutive_lower_case": 2,
            "consecutive_digits": 2,
            "sequential_letters": 2,
            "sequential_nums": 2,
            "dictionary_word": 2
        }

        #Calculating the score
        self.score = sum((
            self.__length_score(),
```

```
            self.__lower_upper_case_score(),
            self.__digits_score(),
            self.__special_score(),
            self.__middle_score(),
            self.__letters_only_score(),
            self.__numbers_only_score(),
            self.__repeating_chars_score(),
            self.__consecutive_case_score(),
            self.__sequential_letters_score(),
            self.__sequential_numbers_score(),
            self.__dictionary_words_score(),
            self.__extra_score()
            ))

    def __chars_of_type(self, chartype):
        chartype_count = 0
        for char in self.password:
            if char in chartype:
                chartype_count += 1

        return chartype_count

    def __length_score(self):
        if self.password_length > 8:
            self.strength_flags["length"] = 3
        elif self.password_length == 8:
            self.strength_flags["length"] = 2

        return self.password_length * 4 #The way WA works.

    def __lower_upper_case_score(self):
        #This will return the number of upper and lower case characters,
        #if at least one of each is available.
        n_upper = self.__chars_of_type(string.ascii_uppercase)
        n_lower = self.__chars_of_type(string.ascii_lowercase)

        # upper_score = 0 if not n_upper else (self.password_length -
n_upper)*2
        # lower_score = 0 if not n_lower else (self.password_length -
n_lower)*2
```

```
        if not n_upper:
            upper_score = 0
        else:
            upper_score = (self.password_length - n_upper)*2

            if n_upper == 1:
                self.strength_flags["upper_case"] = 2
            else:
                self.strength_flags["upper_case"] = 3

        if not n_lower:
            lower_score = 0
        else:
            lower_score = (self.password_length - n_lower)*2

            if n_lower == 1:
                self.strength_flags["lower_case"] = 2
            else:
                self.strength_flags["lower_case"] = 3

        if lower_score and upper_score:
            return lower_score + upper_score

        return 0

    def __digits_score(self):
        digit_count = self.__chars_of_type(string.digits)

        if digit_count == 1:
            self.strength_flags["digits"] = 2
        elif digit_count > 1:
            self.strength_flags["digits"] = 3

        return digit_count * 4 #The way WA works.

    def __special_score(self):
        special_char_count = self.__chars_of_type(string.punctuation)

        if special_char_count == 1:
```

```
            self.strength_flags["special_chars"] = 2
        elif special_char_count > 1:
            self.strength_flags["special_chars"] = 3

        return special_char_count * 6 #The way WA works.

    def __middle_score(self):
        #WA doesn't count special characters in this score (which probably
        #is a mistake), but we will.
        middle_chars_count = 0

        middle_chars = set(string.punctuation + string.digits)

        for char in self.password[1:-1]:
            if char in middle_chars:
                middle_chars_count += 1

        if middle_chars_count == 1:
            self.strength_flags["middle_num_symbol"] = 2
        elif middle_chars_count >1:
            self.strength_flags["middle_num_symbol"] = 3

        return middle_chars_count * 2 #The way WA works


    def __letters_only_score(self):
        letter_count = self.__chars_of_type(string.ascii_lowercase +
string.ascii_uppercase)

        if self.password_length == letter_count:

            self.strength_flags["letters_only"] = 1
            #If the password is all letters, return the negative
            #of the password length. The way WA works.
            return -self.password_length
        #If the password contains something else than letters,
        #don't affect the score.
        return 0

    def __numbers_only_score(self):
```

```
        digit_count = self.__chars_of_type(string.digits)

        if self.password_length == digit_count:

            self.strength_flags["numbers_only"] = 1

            #If the password is all numbers, return the negative
            #form of the password length. The way WA works.
            return -self.password_length

        #If the password contains something else than numbers,
        #don't affect the score.
        return 0

    def __repeating_chars_score(self):
        repeating_char_count = self.password_length -
len(set(self.password))

        if repeating_char_count:
            self.strength_flags["repeating_chars"] = 1

        return -repeating_char_count #The way WA works

    def __consecutive_case_score(self):
        char_types = {
            string.ascii_lowercase: 0,
            string.ascii_uppercase: 0,
            string.digits   : 0,
            }

        for c1, c2 in zip(self.password, self.password[1:]):
            #zips match the length of the shorter by default

            for char_type in char_types:
                if c1 in char_type and c2 in char_type:
                    char_types[char_type] += 1

        if char_types[string.ascii_uppercase]:
            self.strength_flags["consecutive_upper_case"] = 1
```

```
        if char_types[string.ascii_lowercase]:
            self.strength_flags["consecutive_lower_case"] = 1

        if char_types[string.digits]:
            self.strength_flags["consecutive_digits"] = 1

        return -2*sum(char_types.values()) #The way WA works

    def __sequential_numbers_score(self):
        sequential_number_count = 0

        for i in range(1000):
            if str(i) + str(i + 1) in self.password:
                sequential_number_count += 2

        if sequential_number_count:
            self.strength_flags["sequential_nums"] = 1

        return -sequential_number_count * 2

    def __sequential_letters_score(self):
        password = self.password.lower()

        sequential_letter_count = 0

        seeing = False
        for c1, c2 in zip(password, password[1:]):
            #zips match the length of the shorter by default
            if ord(c1)+1 == ord(c2) and c1 in string.ascii_lowercase[:-1]:
                sequential_letter_count += 1

                if not seeing:
                    #until now, we weren't in a pocket of sequential
letters
                    sequential_letter_count += 1
                    seeing = True
            else:
                #we've seen a whole pocket, and counted everything
                seeing = False
```

```
        sequential_letter_count -= 2 #The way WA works

        if sequential_letter_count > 0:
            self.strength_flags["sequential_letters"] = 1
            return sequential_letter_count * -2 #The way WA works


        return 0

    def __substrings(self, word):
        #generate all substrings of length at least 3
        for i in range(len(word)):
            for j in range(i+3, len(word)+1):
                yield word[i:j]

    def __dictionary_words_score(self):
        password_substrings = self.__substrings(self.password)

        if self.words.intersection(password_substrings):
            self.strength_flags["dictionary_word"] = 1
            return -20
        return 0

    def __extra_score(self):
        if self.password_length < 8:
            return 0

        upper = False
        lower = False
        special = False
        number = False

        for char in self.password:
            if char in string.ascii_uppercase:
               upper = True
            elif char in string.ascii_lowercase:
                lower = True
            elif char in string.punctuation:
                special = True
            elif char in string.digits:
```

```
                number = True

        if upper and lower and (special or number):
            if special and number:
                self.strength_flags["extra"] = 3
            else:
                self.strength_flags["extra"] = 2

            return 8

        return 0

    def get_score(self):
        return self.score

    def get_readable_score(self):
        if self.score < 0:
            return 'Very weak'
        elif self.score < 30:
            return 'Weak'
        elif self.score < 60:
            return 'OK'
        elif self.score < 80:
            return 'Strong'

        return 'Very strong'

    def get_results(self):
        results = {
            "length_score" : self.__length_score(),
            "lower_upper_case_score" : self.__lower_upper_case_score(),
            "digits_score" : self.__digits_score(),
            "special_score" : self.__special_score(),
            "middle_score" : self.__middle_score(),
            "letters_only_score" : self.__letters_only_score(),
            "numbers_only_score" : self.__numbers_only_score(),
            "repeating_chars_score" : self.__repeating_chars_score(),
            "consecutive_case_score" : self.__consecutive_case_score(),
            "sequential_letters_score" :
self.__sequential_letters_score(),
```

```
            "sequential_numbers_score" :
self.__sequential_numbers_score(),
            "dictionary_words_score" : self.__dictionary_words_score(),
            "extra_score" : self.__extra_score()
            }
        return results

    def get_strength_flags(self):
        return self.strength_flags
```

3 - Composition/Integration Unit(Code-Snippet):

```
import sys
import os
sys.path.append(os.path.join(os.pardir, "passwordstrength"))
sys.path.append(os.path.join(os.pardir, "ml"))
import ml_passwordstrength
import passwordstrength

password = passwordstrength.passwordstrength(input_string)
ml_password = ml_passwordstrength.ml_passwordstrength(input_string)
password_score = password.get_score()
ml_password_score = ml_password.get_ml_score()

password_score = password_score if password_score <= 100 else 100

if ml_password_score == 1:
      ml_password_score = 25
elif ml_password_score == 2:
      ml_password_score = 60
elif ml_password_score == 3:
      ml_password_score = 100

final_score = (password_score + ml_password_score) / 2

if final_score < 30:
      password_complexity = "Very Weak"
elif final_score < 50:
      password_complexity = "Weak"
elif final_score < 70:
      password_complexity = "OK"
elif final_score < 90:
      password_complexity = "Strong"
else:
      password_complexity = "Very Strong"
```

4 - Interface Unit(Code-Snippet):

```
from flask import
Flask,render_template,request,jsonify,send_from_directory
import passwordstrength
import pickle
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer


app = Flask(__name__)


'''
@app.route('/')
def home():
    return render_template('home.html')
'''
def getTokens(inputString):
    tokens = []
    for i in inputString:
    # i = i.astype('u')
        tokens.append(i)
    # print(tokens)
    return tokens


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

filepath = './data_5.csv' #path for password file
data = pd.read_csv(filepath,',',error_bad_lines=False)
data.isnull().any()

data = pd.DataFrame(data)
X = data.iloc[:, 0].values
vectorizer = TfidfVectorizer(tokenizer=getTokens)
X = vectorizer.fit_transform(X)
```

```
class ml_passwordstrength:
    def __init__(self, password):
        self.password = []
        self.password.append(password)

        self.score = self.__classification_score()[0] + 1


    def __classification_score(self):
        vect_password = vectorizer.transform(self.password)
        return model.predict(vect_password)

    def get_ml_score(self):
        return self.score



@app.route('/',methods=['GET','POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    password_value = ""
    if request.method == 'POST':
        password_value = request.form['pwd']
    password = passwordstrength.passwordstrength(password_value)
    ml_password = ml_passwordstrength(password_value)
    password_score = password.get_score()
    ml_password_score = ml_password.get_ml_score()

    results = password.get_results()
    flags = password.get_strength_flags()

    password_score = password_score if password_score <= 100 else 100


    if ml_password_score == 1:
        ml_password_score = 25
    elif ml_password_score == 2:
```

```
        ml_password_score = 60
    elif ml_password_score == 3:
        ml_password_score = 100

    final_score = (password_score + ml_password_score) / 2

    if request.method == 'GET':
        final_score = 0

    return render_template('index.html',password_value =
password_value,final_score = final_score, results = results, flags =
flags)

@app.route('/report')
def report():
    filepath = 'static/Report.pdf'
    return send_from_directory(app.config['UPLOAD_FOLDER'],
filepath,as_attachment = True)


if __name__ == "__main__":
    app.run(debug=False)
```