Password Strength Evaluator - Team 24

1st Hao-Wei Chen
Computer Science of TSING HUA
108062302

2nd Han-Lin Lai
Computer Science of TSING HUA
108062118

3rd Yu-Cheng Lin
Computer Science of TSING HUA
108062128

4th Tung-Yeh Wu
Computer Science of TSING HUA
108062129

***Abstract*—This paper is an introduction about our final project - ”Password Strength Evaluator”. We will introduce our methods, models, results and our discussion about how we implement our project.**

I. Introduction

Our final project is to evaluate the strength of the input password. Not only evaluating the strength of the password, we also calculate the time a hacker needs to crack the input password by searching a list of passwords corresponding to the time a hacker needs to crack it.

II. Methods

A. Dataset

For the dataset we used, we download the dataset from Kaggle whose name is ”Password Strength Classifier Dataset”. It provides 670,000 passwords and each of them with a corresponding strength 0(weak), 1(medium) and 2(strong).

However, we find out that the dataset is quite imbalance, there are too much medium strength password which the strength is 1. So we append 10,000 more passwords which are the most common password from another 10 million password dataset. We gave those 10,000 passwords with strength 0(weak). By this, we can slightly improve the imbalance of our dataset and can get a better model instead of getting a model which gives every input password a strength with 1(medium). Now we can start finding the features.

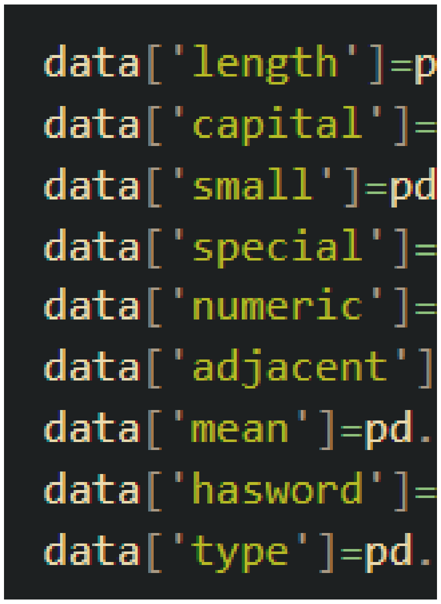
B. Features

The picture below is the features we defined. ”Lenght” is the length of the password. ”Capital” is the number of the capital letters in the password. ”Small” is the letters which are not capital. ”Special” is to show that if this password include any special characters. ”Numeric” is the number of the numbers in this password. ”Type” is the number of how many kinds of character userd in this password.

For some special features, such as ”Adjacent”. We defined a function which gives a character a coordinate and by calculating the Manhattan distance between each character. By calculating this, we can detect the password which seems stochastic but in fact is just typing through the keyboard such as: qwertyui, asdfghjk. We can also detect the same character password such as: aaaaaaaa, bbbbbbbb.

And for ”Mean”, it means the continuity of character of a word. We simply discuss only alphabets and numbers. For example, when the input password is: abcdefgh, we will get its mean equals to 7. By this feature, we can prevent some lazy password such as: 12345678 or hijklmnop. For passwords mixed with alphabets and numbers: a1p2p3l4e5, we seperate it as apple and 12345 to calculate their mean and add them together.

”Hasword” is implemented by importing nltk library and use the dictionary inside to find out if this password contains any meaningful word. For some passwords such as a1p2p3l4e5, we list the alphabets out and we get apple. Now we know there is an meaningful word inside the password.



```
data['length']=p
data['capital']=
data['small']=pd
data['special']=
data['numeric']=
data['adjacent']
data['mean']=pd.
data['hasword']=
data['type']=pd.
```

Fig. 1. Features of a password.

III. Method

A. Steps

To implement our project, we first turn the dataset into dataframes of features. Then we try some models we wanted

Identify applicable funding agency here. If none, delete this.

to use and test each accuracy. At last we write a website and import our model into it and now we can evaluate any input passwords.

B. Models

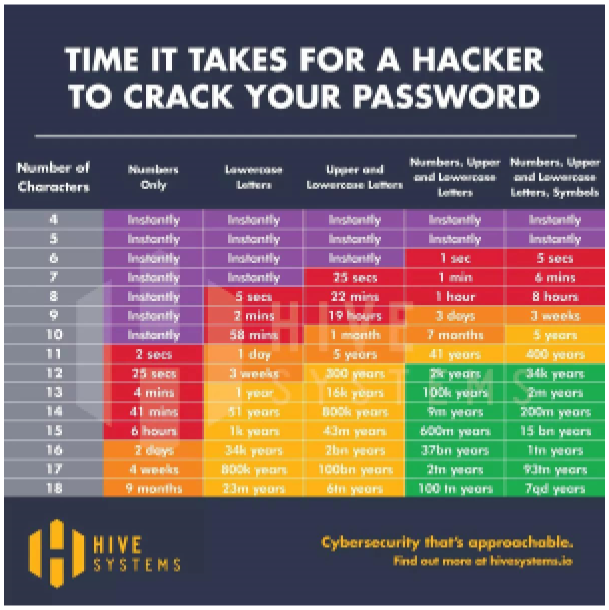
For the models we tried, we tried two kinds of models: **LinearRegression** and **Neural Network**.

Since we first search for some similar projects as ours. We find out that most of the project used **LinearRegression**, so we chose it as a model we are going to try. We train a model with high mean accuracy as we split our dataset with 20 percent of validation test, and we get the score close to one hundred percent.

For Neural Network, we try different activation functions such as *ReLU*, *sigmoid* and *tanh*. After training, the results of three activations has almost the same accuracy and all of them are close to a hundred percent. At last, we decided to use sigmoid as our hidden layers' activation function since we are going to classify our passwords.

C. Website

We build the website via Flask and Django template, and we design the website blueprint through figma. We import our model into the website and we can easily evaluate the input password and show the strength on the website.



TIME IT TAKES FOR A HACKER TO CRACK YOUR PASSWORD

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 tn years	7qd years

HIVE SYSTEMS

Cybersecurity that's approachable.
Find out more at hivesystems.io

Fig. 2. Hacker crack time list.

We calculate the time a hacker can crack the password through the list above. Since it doesn't list all possibilities and we are not hackers, we don't know the actually time we need to crack the passwords aren't listed. So we calculate the relation between the types and the length of the password. At last, we find out the relation below:

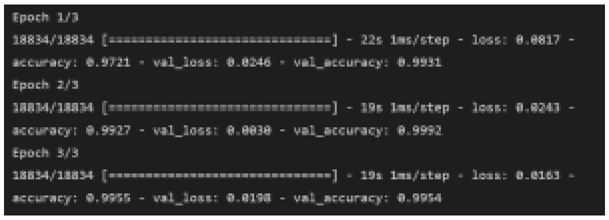
$$pico = pow(mini, (length.item() - 1)) * 0.6 \tag{1}$$

"Mini" is the possibility of a single character. If the character is a number, mini will plus 10, if it is an alphabet, it will plus 26. Length is the length of the password and 0.6 represents 600 picoseconds to crack a character. After calculating out the picosecond, we can then calculate the actual time by turning picosecond to second or minutes or even days.

However, the model size is too big and we also import NLP language package. We can't find any free domain for us to setup our website. We tried another method to finish the implement. We setup the website on our localhost through anaconda/flask, then we use ngrok. We can connect to the ngrok cloud server and it will make our localhost public and we can access the request transmitted to our designated port via ngrok. Then we can finally get the password input by users.

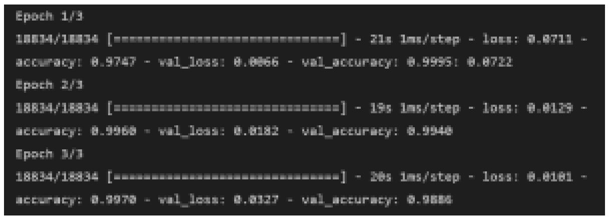
D. Results

- Result of tanh activation function with accuracy = 0.9954.



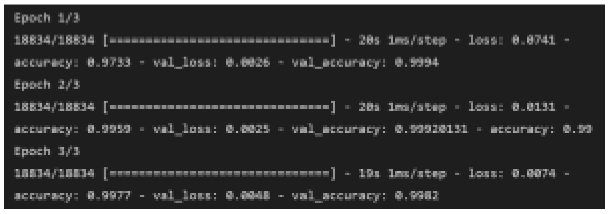
```
Epoch 1/3
18834/18834 [==============================] - 22s 1ms/step - loss: 0.0817 -
accuracy: 0.9721 - val_loss: 0.0246 - val_accuracy: 0.9931
Epoch 2/3
18834/18834 [==============================] - 19s 1ms/step - loss: 0.0243 -
accuracy: 0.9927 - val_loss: 0.0030 - val_accuracy: 0.9992
Epoch 3/3
18834/18834 [==============================] - 19s 1ms/step - loss: 0.0163 -
accuracy: 0.9955 - val_loss: 0.0198 - val_accuracy: 0.9954
```

- Result of ReLU activation function with accuracy = 0.9886.



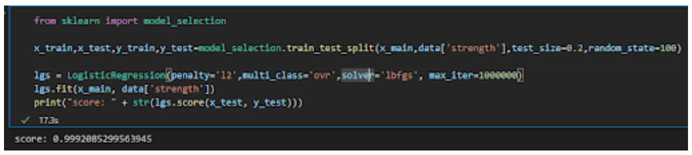
```
Epoch 1/3
18834/18834 [==============================] - 21s 1ms/step - loss: 0.0711 -
accuracy: 0.9747 - val_loss: 0.0066 - val_accuracy: 0.9995: 0.0722
Epoch 2/3
18834/18834 [==============================] - 19s 1ms/step - loss: 0.0129 -
accuracy: 0.9960 - val_loss: 0.0182 - val_accuracy: 0.9940
Epoch 3/3
18834/18834 [==============================] - 20s 1ms/step - loss: 0.0101 -
accuracy: 0.9970 - val_loss: 0.0327 - val_accuracy: 0.9886
```

- Result of sigmoid activation function with accuracy = 0.9982.



```
Epoch 1/3
18834/18834 [==============================] - 20s 1ms/step - loss: 0.0741 -
accuracy: 0.9733 - val_loss: 0.0026 - val_accuracy: 0.9994
Epoch 2/3
18834/18834 [==============================] - 20s 1ms/step - loss: 0.0131 -
accuracy: 0.9959 - val_loss: 0.0025 - val_accuracy: 0.99920131 - accuracy: 0.99
Epoch 3/3
18834/18834 [==============================] - 19s 1ms/step - loss: 0.0074 -
accuracy: 0.9977 - val_loss: 0.0048 - val_accuracy: 0.9982
```

- Result of **LinearRegression** with score = 0.9992.

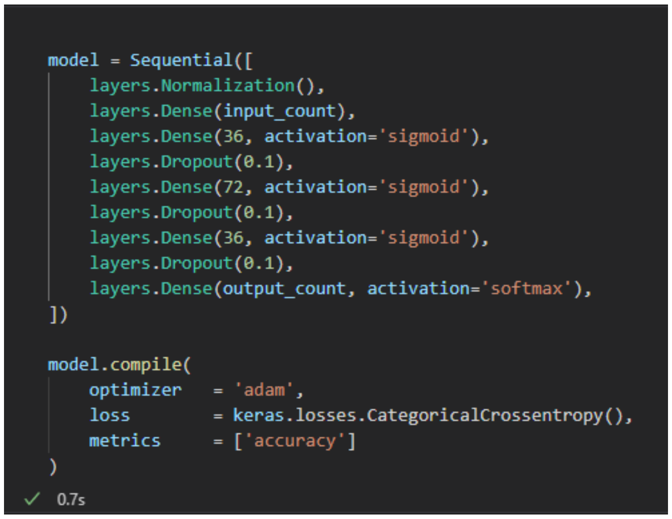


```
from sklearn import model_selection

x_train,x_test,y_train,y_test=model_selection.train_test_split(x_main,data['strength'],test_size=0.2,random_state=100)

lgs = LogisticRegression(penalty='l2',multi_class='ovr',solver='lbfgs', max_iter=1000000)
lgs.fit(x_main, data['strength'])
print("score: " + str(lgs.score(x_test, y_test)))
✓ 17.3s
score: 0.9992085299563945
```

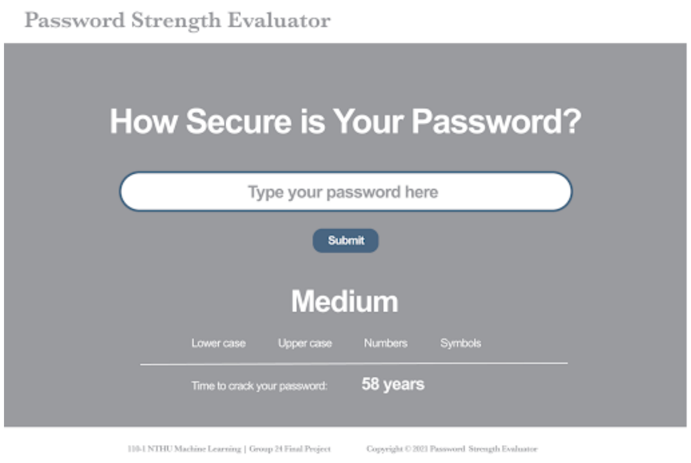
- Model of our **Neural Network**.
- Website designed by Figma.



```
model = Sequential([
    layers.Normalization(),
    layers.Dense(input_count),
    layers.Dense(36, activation='sigmoid'),
    layers.Dropout(0.1),
    layers.Dense(72, activation='sigmoid'),
    layers.Dropout(0.1),
    layers.Dense(36, activation='sigmoid'),
    layers.Dropout(0.1),
    layers.Dense(output_count, activation='softmax'),
])

model.compile(
    optimizer   = 'adam',
    loss        = keras.losses.CategoricalCrossentropy(),
    metrics     = ['accuracy']
)
```

```
✓ 0.7s
```


Password Strength Evaluator
How Secure is Your Password?
Type your password here
Submit
Medium
Lower case
Upper case
Numbers
Symbols
Time to crack your password:
58 years
110-1 NTHU Machine Learning | Group 24 Final Project
Copyright © 2021 Password Strength Evaluator

E. *Discussion*

We kept discussing how far are we going to do at the beginning. We first wanted to try to give a better password after evaluating the password of the user inputs. However, we don't actually know what kind of password given out is better. We also face some problem while training our model since our dataset are quite large but imbalanced. We always get a medium or low while trying different kinds of input.

At last we find out some format of our model. We need to type across the keyboard to get a strong password and we need to avoid to type any meaningful words in a password because of our dataset. The dataset contains too many medium or weak strength passwords including meaningful words but stochastic password with strong strength. We didn't have that much time to trace all 670,000 passwords so we can't prune the dataset better. This is the difficulties we face and it is also the reason we add 10,000 more weak passwords to fix the imbalance so the model won't give every password a medium strength.

F. *Conclusion*

To be honest, we didn't train a model same as some website strength evaluator because of the lack of data. We seek for many datasets and finally find the one we used. It is also the best dataset we considered because of the scale and the passwords we briefly traced. Since no features are given at first, we defined the features ourselves and we can also test the importance of each feature. Although some input password didn't get the strengh we expected, but the strength of the password is quite subjective so we did try our best to finish this project.

G. *Author Contribution*

Hao-Wei Chen(25%): study design, Linear Regression Training, final presentation, project report Han-Lin Lai(25%): study design, website design and implement, writing Yu-Chen Lin(25%): study design, Neural Network Training, Django(import model to website) Tung-Yeh Wu(25%): study design, data collection, data preprocess, feature define

H. *Reference*

REFERENCES

[1] https://www.kaggle.com/bhavikbb/password-strength-classifier-dataset
[2] https://www.security.org/how-secure-is-my-password/
[3] https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-10000.txt
[4] https://www.kaggle.com/wjburns/common-password-list-rockyoutxt
[5] https://www.uic.edu/apps/strong-password/