**Final project report**

**on**

**PASSWORD STRENGTH CLASSIFIER**

*Syracuse University*

*IST707 – Data Analytics*

***GROUP 7***

***Professor In-charge:*** *Prof. Ying Lin*

***Authors:*** *Trina Ghosh*

*Harmeet Singh Lamba*

*Jack Dunn*

**TABLE OF CONTENTS**

1. **Introduction ……………………………………………………………… 3**
2. **Data Description ……………………………………………………… 4**
3. **Data Wrangling ……………………………………………………… 6**
4. **Data Distribution and Sampling ……………………………………… 8**
5. **Classification Algorithms ……………………………………………… 9**
   1. Naïve Bayes Classification Algorithm ……………………………… 9
   2. Support Vector Machine ……………………………………… 10
   3. Random Forest Classifier ……………………………………… 12
6. **Evaluation ……………………………………………………………… 14**
   1. Naïve Bayes Classification Algorithm ……………………………… 14
   2. Support Vector Machine ……………………………………… 14
   3. Random Forest Classifier ……………………………………… 14
   4. Results ……………………………………………………………… 14
7. **Conclusion ……………………………………………………………… 15**
8. **Appendix ……………………………………………………………… 17**
   1. Data Preparation Script (Python) ……………………………… 17
   2. R Shiny App Script (R) …………………………………….... 21
9. **References ……………………………………………………………… 43**
10. ***Introduction***

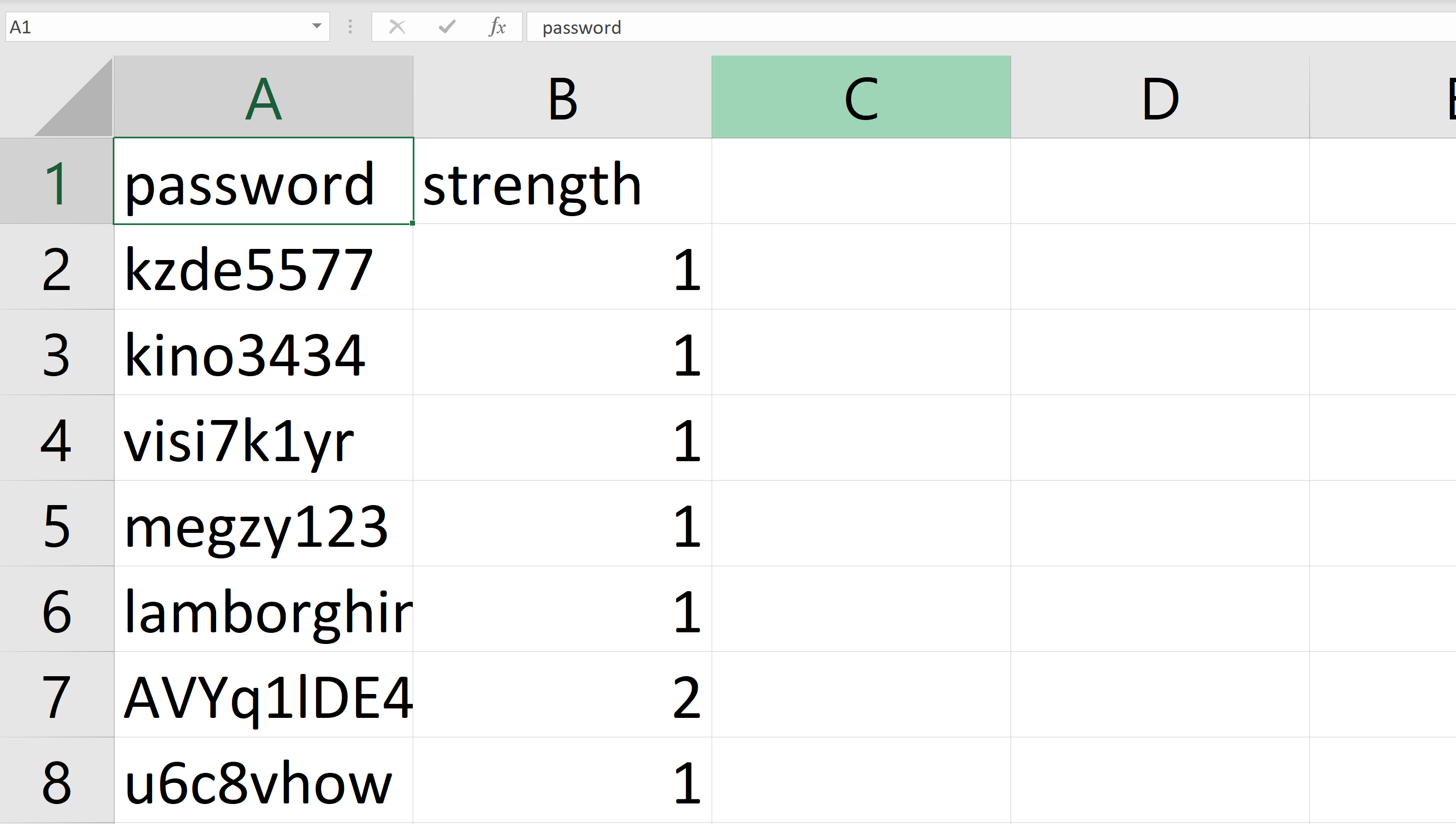
For every organization, security is the primary concern and to ensure security they must ensure there is no way an unauthorized personnel or entity can access their system. Each employee working in that organization is responsible to ensure safety. An outside entity obtaining the password of an employee is a nightmare for any organization. So, it is the responsibility of each employee to make their password unique and hard to guess. This will help them as well as the organization to maintain their level of customer services they provide.

To do this, employees must know whether their password is strong. Therefore, we used several machine learning algorithms to predict the strength of a given password based on traits like length and whether it contains words. These models could then be used to predict the strength of new passwords.

1. ***Data Description***

This Data was downloaded from “*Password Strength Classifier Dataset*” Kaggle (Kaggle, 2019). It consisted of approximately 700K observations, with only two columns passwords and their respective strengths ranging from 0-2, with “0” being a weak password. These strengths were provided by Georgia Tech University. They did this by running an array of passwords through a tool called “*Password Analysis and Research System*” (developed and maintained by Georgia Tech University).

On the homepage of PARS the creators describe the tool as follows: “In this module, leveraging a large corpus (~115M) of real-world passwords, we develop several analytical functions to characterize password datasets. The strength distribution of passwords in terms of specific metrics or meters can also be computed by interacting with the corresponding module in PARS” (Georgia Tech University, n.d.). The original Dataset snapshot could be seen below.



**Figure 2.1 Original Dataset**

The original dataset is shown below in the screen shot from the application.

A screenshot of a social media post

Description automatically generated

**Figure 2.2 The original dataset in the app.**

1. ***Data Wrangling***

During the process of data cleaning and preprocessing we found approx. 15 observations with email-ids in the strength column which made zero sense. Hence, we deleted them before moving further.

Furthermore, as the dataset had only two columns, we had to perform extensive feature engineering to get attributes we could use to build models. The attributes extracted were:

* Type of first & last characters.
* Number of uppercase, lowercase, space, special ($, %, &, \*, #, etc.) and numerical characters.
* If there are any dictionary words in the password.
* If there are any consecutively repeated characters. ex “aaa” or “111”.
* Number of times the type of character switches to another type.

|  |  |
| --- | --- |
| Password: **kzde5577** | |
| First Char | L |
| Last Char | N |
| Not Lower | 4 |
| Not Upper | 0 |
| Not Special | 0 |
| Not Space | 0 |
| Not Number | 4 |
| Repeat? | 0 |
| Word? | 0 |
| Switches | 1 |

**Table 3.1 An example of the feature engineering**

The screenshot shown below from the application shows the final dataset with feature engineering.

A screenshot of a social media post

Description automatically generated

**Fig 3.1 Screenshot of Dataset after Feature Engineering**

1. ***Data Distribution and Sampling***

The data across the strength column was biased with 74% of the data for strength “1”, 11% for strength “0” and the remaining for strength “2”. The same can be seen in the fig 4.1.

A screenshot of a social media post

Description automatically generated

**Figure 4.1 Distribution of the Strength**

To overcome this issue, we experimented with sampling techniques. During research for solutions we came across two of the widely used sampling techniques, Under Sampling and Over Sampling (SMOTE).

The first technique we played with was under-sampling where we only get a certain amount of the majority category of data so that it does not over saturate the training data.

Furthermore, we also tried Synthetic Minority Over-sampling Technique or SOMTE to balance out the data. According to a 2002 paper published in the *Journal of Artificial Intelligence*, “A combination of our method of over-sampling the minority (abnormal) class and under-sampling the majority (normal) class can achieve better classifier performance (in ROC space) than only under-sampling the majority class.”(Chawla eta all, 2002). This shows that it is a good method for balancing skewed data such as ours.

1. ***Classification Algorithms***

To find the reasoning for each password to be classified as 0, 1 or 2, we used our extracted features to build our own classification system.

Scikit-Learn, a common machine learning package for Python was used for building all models. We also used the Shiny Package to allow easy understanding of the data and real time hyperparameters tuning for our models. Different pages of our app gave the user to tune their model with ease.

* 1. **Naive Bayes Classification Algorithm**

The first model we tested was the Naïve Bayes Model. We tested all three types of Naïve Bayes models namely Gaussian, which is often used for continuous attributes, Bernoulli, which is often used for binary attributes, and Multinomial which is often used for discrete data. We also looked at how the soothing parameter effected the outcome. Shown in figure 5.1.1 and figure 5.1.2 are the application pages that were used for the exploration.

A screenshot of a social media post

Description automatically generated

**Figure 5.1.1 Naïve Bayes with no Sampling**

A screenshot of a social media post

Description automatically generated

**Figure 5.1.2 Naïve Bayes with Sampling**

* 1. **Support Vector Machine**

Support Vector Machine is a very powerful machine learning tool but is often very computationally heavy. Because of this when ever running this model, we had to take a random subset of the data to ensure that the program could run.

We experimented with four different kernels:

* Linear
* Polynomial
* Radial basis function (RBF)
* Sigmoid

We also explored what effect the cost parameter C had on the models.

Again, in figure 5.2.1 and figure 5.2.2 are the application pages that were used for the exploration.

A screenshot of a cell phone

Description automatically generated

**Figure 5.2.1 SVM With No Sampling**

A screenshot of a cell phone

Description automatically generated

**Figure 5.2.2 SVM With Sampling**

* 1. **Random Forest**

The last model we built was a random forest model.

We experimented with hyperparameters maximum depth of the trees, maximum number of features for each tree and the maximum number of trees.

We also tried the model with and without bootstrapping and what type of evaluation criterion we used. The two options were Gini or Entropy.

Just as before figure 5.3.1 and figure 4.3.2 are the application pages that were used for the exploration

A screenshot of a cell phone

Description automatically generated

**Figure 5.3.1 Random Forest with No Sampling**

A screenshot of a social media post

Description automatically generated

**Figure 5.3.2 Random Forest with Sampling**

1. ***Evaluations***

After our testing, we found the best parameters for each model.

* 1. **Naïve-Bayes**

We found that the Gaussian Naïve-Bayes model was the best sub-type. We also found that a smoothing factor of 0.021 was ideal. This granted us an accuracy of 75.15% for the training data.

* 1. **Support Vector Machine**

The Radial basis function (RBF) turned out to be the best kernel. The cost of 0.026 was the best and the gamma was set to the ‘auto’ feature. This gave an accuracy of 84.00%.

* 1. **Random Forest**

The final hyperparameters were a max depth of 5, max number of features as 4 and 25 trees. There was no bootstrapping and the evaluation criteria was entropy. This resulted in an accuracy of 91.08%.

* 1. **Results**

Table 6.4.1 shows all the evaluation metrics for our final models.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ­­ | Accuracy | Precision | Recall | F1-Score |
| Naïve Bayes | 75.15% | 0.70 | 0.83 | 0.76 |
| Support Vector Machine | 84.00% | 0.79 | **0.91** | 0.83 |
| Random Forest | **91.08%** | **0.94** | 0.77 | **0.85** |

**Table 6.4.1 Evaluation Metrics for all the models used**

After running multiple models, we observed that Random Forest and SVM had th­­­­e best at classifying passwords correctly. Due to the better accuracy of the random forest as well as the good balance of precision, recall and f1 score, we deemed that Random Forest was the best model. Especially since it has a faster run time than Support Vector Machine.

1. ***Conclusion***

We were able to determine the strength of a password with very high accuracy. All three models were well within error limits that would be reasonable for this application, allowing some room for misclassification. Furthermore, having 91.08% from random forest is definitely a model that can used by people to predict the strength of their password. The accuracy is not too high that the models risks overfitting.

Since our models were able to predict the strength, we built an application to allow people to predict the strength of their password. Users must only input their password; it will then find all of the attributes needed and then run our models on it to predict the strength of the inputted password. Figure 7.1 shows the input screen and figure 7.2 shows the attributes calculated. Figure 7.3 shows the outcome of the application.

A screenshot of a cell phone

Description automatically generated

**Figure 7.1 Input Screen for Password Strength Classifier**

A screenshot of a social media post

Description automatically generated

**Figure 7.2 Attributes for Example Password**

**A screenshot of a cell phone

Description automatically generated**

**Figure 7.3 Output of Example Password Classification**

As of now, we can only predict the strength of a password. In the future, we would like to create a model that not only describes the strength of the password but also which attributes about it need to be changed. For example, tell someone their password is too short or contains too few types of characters.

1. ***Appendix***
   1. ***Data Preparation Script (Python)***

import pandas as pd

import numpy as np

#english\_vocab = set(w.lower() for w in nltk.corpus.words.words())

#english\_vocab = [item for item in english\_vocab if len(item)>2]#only keep words longer than 3 letters

nltk.corpus.english.words()

Password = pd.read\_excel("Password.xlsx")

Password.drop(['Unnamed: 2', 'Unnamed: 3', "Unnamed: 4", "Unnamed: 5", "Unnamed: 6"], axis=1, inplace = True)

def n\_lower\_chars(string):

return sum(map(str.islower, string))

def n\_upper\_chars(string):

return sum(map(str.isupper, string))

def n\_num\_chars(string):

return sum(map(str.isdigit, string))

def n\_space\_chars(string):

return sum(map(str.isspace, string))

def split(word):

return [char for char in word]

def char\_detect(charx):

char\_type = "S"

if charx.isupper():

char\_type = "U"

elif charx.islower():

char\_type = "L"

elif charx.isdigit():

char\_type = "N"

elif charx.isspace():

char\_type = "s"

else:

char\_type = "S"

return char\_type

def count\_switches(string):

string = split(str(string))

count = 0

if len(string) > 1:

i = 0

j = 1

for y in range(0,len(string) -1 ):

if char\_detect(string[i]) != char\_detect(string[j]):

count = count + 1

i, j = i+1,j+1

return count

def reps(string):

temp = split(str(string))

i = 0

j = 1

k = 2

flag = 0

for y in range(2,len(temp)-1):

if temp[i] == temp[j] and temp[j] == temp[k]:

flag = 1

y = len(temp) + 1

else:

i += 1

j += 1

k += 1

return flag

# Word Search

def breakStr(password):

#Set up inital variables

word = []

words = []

let = False #this keeps track of if the last character was a letter

password = str(password)

password1 = password +"1"

for i in password1:

if i.isalpha() == False and let == False:

continue

elif i.isalpha() == True:

word.append(i)

let = True

elif i.isalpha() == False and let == True:

wordStr = ''.join(word)

wordStr = wordStr.lower()

if len(wordStr) >= 3:

words.append(wordStr)

word = []

let = False

return (words)

# Word Search

#Now we will check if a string has a word

def checkWord(word):

result = 0

w1 = ""

#Check if full string is a word

if word in english\_vocab:

result = 1

#print(word)

#Check if substrings are words

else:

#check going forward

k = 2

while k < len(word):

if (word[0:k]) in english\_vocab:

result += 1

w1 = word[0:k]

end = -(len(word) - k+1)

k = len(word)

else:

end = -len(word)

k += 1

#Check going backward

k = -3

while k > end:

if word[k:] in english\_vocab:

result += 1

k = -len(word) -1

else:

k -= 1

return(result)

# now we will sum up how many words are in a string

ex1 = ['rst', 'word', "worda"]

#count = 0

def sumWords(password):

sumWords = 0

words = breakStr(password)

print(password)

for word in words:

#print(checkWord(word))

sumWords += checkWord(word)

return (sumWords)

Password["FChar"] = Password.password.astype(str).str[0].apply(char\_detect)

Password["LChar"] = Password.password.astype(str).str[-1].apply(char\_detect)

Password['TotalLength'] = Password.password.astype(str).apply(len)

Password['LowerChar'] = Password.password.astype(str).apply(n\_lower\_chars)

Password['UpperChar'] = Password.password.astype(str).apply(n\_upper\_chars)

Password['NumChar'] = Password.password.astype(str).apply(n\_num\_chars)

Password['SpaceChar'] = Password.password.astype(str).apply(n\_space\_chars)

Password['SpecialChar'] = Password.TotalLength - (Password.LowerChar + Password.UpperChar + Password.SpaceChar + Password.NumChar)

Password['TotalSwitches'] = Password.password.astype(str).apply(count\_switches)

Password['Repeats'] = Password.password.astype(str).apply(reps)

* 1. ***R Shiny App Script (R)***

#

# This is a Shiny web application. You can run the application by clicking

# the 'Run App' button above.

#

# Find out more about building applications with Shiny here:

#

# http://shiny.rstudio.com/

#

#library(shiny)

library(shinyjs)

library(reticulate)

#library(tm)

#options(scipen = 999)

#py\_available()

#virtualenv\_create(envname = "python\_environment",python= "venv/bin/python")

use\_python("C:\\Users\\trina\\Anaconda3\\python.exe", required = T)

#reticulate::use\_virtualenv("python\_environment", required = TRUE)

#virtualenv\_create(envname = "python\_environment",python= python3)

#virtualenv\_install(envname, packages, ignore\_installed = FALSE)

#py\_install(c("pandas", "numpy"))

py\_run\_string("import pandas as pd")

py\_run\_string("import numpy as np")

py\_run\_string("from sklearn.naive\_bayes import GaussianNB, BernoulliNB, MultinomialNB")

py\_run\_string("from imblearn.over\_sampling import SMOTE")

py\_run\_string("from imblearn.under\_sampling import RandomUnderSampler")

py\_run\_string("from sklearn.svm import SVC")

py\_run\_string("from sklearn.ensemble import RandomForestClassifier")

py\_run\_string("from sklearn.model\_selection import train\_test\_split")

py\_run\_string("from sklearn import metrics")

py\_run\_string("from sklearn.metrics import confusion\_matrix, f1\_score, classification\_report")

py\_run\_string("passwords = pd.read\_csv('Final\_file.csv')")

#py\_run\_string("passwords = passwords[passwords.columns[1:24]]")

passwords <- py$passwords

x <- 0

# Define UI for application that draws a histogram

ui <- navbarPage(

tabPanel("Welcome Page", h4("Password Strength Classifier")),

#---------------------------------------------------------------------------------------------------------

tabPanel("About",

h2("What is this application used for?"),

h4("This application comprises of predicting the strength of passwords based on their underlying features, such as length and different characters. The project was built using the public dataset Password Strength Classifier available on Kaggle. The dataset was built by running an array of passwords through a tool called \"PARS\" (developed by Georgia Tech University) which contains inbuilt features to determine the strength of any password. The dataset contains approximately 700,000 passwords and their respective strengths."),

br(),

h2("Feature Engineering and Oversampling:"),

h4("The original dataset contained only 2 columns: password and strength. In order to build more effective machine learning models, we engineered more features that would help us predict their strengths better. The attributes we engineered were:"),

HTML("<ul><li>Number of uppercase, lowercase, special, numeric characters and spaces</li><li>Type of first and last characters</li><li>If there are any consecutively repeated characters</li><li>Number of switches from one type of character to another, i.e., from uppercase to lowercase or numeric to special character</li><li>If there are any dictionary words in the password</li></ul>"),

h4("We then found out most of the data had a strength of 1 (about 74%). In order to reduce biasness, we decided to run Smote to over-sample our data. This would produce redundant data points for the under-represented classes."),

br(),

h2("Classification Algorithms: "),

h4("Three machine learning models where used for this classification problem:"),

HTML("<ul><li>Naive Bayes Classifier</li><li>Support Vector Machines</li><li>Random Forest Classifier</li></ul>")

),

tabPanel("Source Dataset",

sidebarLayout(

sidebarPanel(

radioButtons("dataset", "Choose from the Dataset :"

, c("Original Dataset" = "og",

"Final Dataset" = "fi")),

selectInput("cols", "Distributions of Features:",

choices=colnames(passwords)[2:22])

),

mainPanel(

tabsetPanel(

tabPanel("Dataset", icon = icon("table"),

dataTableOutput("data")),

tabPanel("Exploratory Data Analysis", icon = icon("poll"),

plotOutput("barplot"))

#tabPanel("Relationship", icon = icon("project-diagram"),

# plotOutput("corrplot"))

)

)

)

),

tabPanel("Naive Bayes Classifier", icon = icon("puzzle-piece"),

sidebarLayout(

sidebarPanel(

radioButtons("nb", "Which algorithm to run :"

, c("Gaussian Naive Bayes : GaussianNB()" = "gaussian",

"Bernoulli Naive Bayes : BernoulliNB()" = "bernoulli",

"Multinomial Naive Bayes : MultinomialNB()" = "multi")),

sliderInput("testperc", "What should be your Testset size ?",

min = 0.1, max = 0.95, value = 0.2, step = 0.05),

conditionalPanel(

condition = "input.nb == \"gaussian\"",

sliderInput("gauslider", "Set Smoothing parameter (var\_smoothing) ?",

min = 0.001, max = 0.99, value = 0.1, step = 0.005)),

conditionalPanel(

condition = "input.nb == \"bernoulli\"",

sliderInput("balphaslider", "Set Smoothing parameter (alpha) ?",

min = 0.1, max = 10, value = 1, step = 0.05)),

conditionalPanel(

condition = "input.nb == \"multi\"",

sliderInput("malphaslider", "Set Smoothing parameter (alpha) ?",

min = 0.1, max = 10, value = 1, step = 0.05)),

checkboxInput("sample\_check", "Change Sampling Technique ? ", F),

conditionalPanel(

condition = "input.sample\_check == true",

radioButtons("sample", "Which algorithm to run :"

, c("Under-Sampling : RandomUnderSampler()" = "rand",

"Over-Sampling : SMOTE()" = "smote")))

),

mainPanel(

h4("Accuracy of Naive Bayes Model"),

verbatimTextOutput("acc"),

h4("Confusion Matrix"),

verbatimTextOutput("conf"),

h4("Evaluation Report"),

verbatimTextOutput("classif")

)

)),

tabPanel("Support Vector Machines Classifier", icon = icon("sitemap"),

sidebarLayout(

sidebarPanel(

sliderInput("sampling", "To reduce computational time, select the sample size :",

min = 1000, max = 15000, value = 2000, step = 500),

sliderInput("testperc2", "What should be your Testset size ?",

min = 0.1, max = 0.95, value = 0.2, step = 0.05),

sliderInput("cost", "Select Penalty term (C : Cost) :",

min = 0.001, max = 10, value = 0.1, step = 0.001),

radioButtons("kern", "Select a Kernel :"

, c("Linear" = "l",

"Poly" = "p",

"rbf" = "r",

"sigmoid" = "s")),

radioButtons("gamma", "Select a kernal coefficient (gamma) :"

, c("Auto" = "a"

, "Scale" = "s")),

checkboxInput("sample\_check\_svc", "Change Sampling Technique ? ", F),

conditionalPanel(

condition = "input.sample\_check\_svc == true",

radioButtons("sample\_svc", "Which algorithm to run :"

, c("Under-Sampling : RandomUnderSampler()" = "rand",

"Over-Sampling : SMOTE()" = "smote")))

),

mainPanel(

h4("Accuracy of Support Vector Machines Model"),

verbatimTextOutput("acc\_sv"),

h4("Confusion Matrix"),

verbatimTextOutput("conf\_sv"),

h4("Evaluation Report"),

verbatimTextOutput("classif\_sv")

)

)),

tabPanel("Random Forest Classifier", icon = icon("tree"),

sidebarLayout(

sidebarPanel(

sliderInput("testperc1", "What should be your Testset size ?",

min = 0.1, max = 0.95, value = 0.2, step = 0.05),

sliderInput("md", "Select the Maximum Depth of a tree ?",

min = 1, max = 10, value = 2, step = 1),

sliderInput("mf", "Select the Maximum number of Features ?",

min = 1, max = 10, value = 2, step = 1),

sliderInput("nest", "Select the Number of Trees ?",

min = 1, max = 60, value = 10, step = 1),

radioButtons("bootstrap", "Enable Bootstrap (resampling) ?"

, c("Yes (True)" = "t",

"No (False)" = "f")),

radioButtons("criterion", "Evaluation Type (criterion) ?"

, c("Gini" = "g",

"Entropy" = "e")),

checkboxInput("sample\_check\_rfc", "Change Sampling Technique ? ", F),

conditionalPanel(

condition = "input.sample\_check\_rfc == true",

radioButtons("sample\_rfc", "Which algorithm to run :"

, c("Under-Sampling : RandomUnderSampler()" = "rand",

"Over-Sampling : SMOTE()" = "smote")))

),

mainPanel(

h4("Accuracy of Random Forest Model"),

verbatimTextOutput("acc\_rf"),

h4("Confusion Matrix"),

verbatimTextOutput("conf\_rf"),

h4("Evaluation Report"),

verbatimTextOutput("classif\_rf")

)

)),

tabPanel("Try it!", icon = icon("balance-scale"),

textInput("pwd", "Enter the dummy password :"

, placeholder = "Enter the password Here!"),

h4("Features Engineering:"),

dataTableOutput("test\_data"),

actionButton("gobutton", "Run Predictions!"),

h4("Predicted strength by Gaussian Naive Bayes :"),

verbatimTextOutput("gnb\_pred"),

h4("Predicted strength by Bernoulli Naive Bayes :"),

verbatimTextOutput("bnb\_pred"),

h4("Predicted strength by Multinomial Naive Bayes :"),

verbatimTextOutput("mnb\_pred"),

h4("Predicted strength by Support Vector Machine :"),

verbatimTextOutput("svm\_pred"),

h4("Predicted strength by Random Forest Classifier :"),

verbatimTextOutput("rfc\_pred")

)

)

# Define server logic required to draw a histogram

server <- function(input, output, session) {

predict\_strength <- eventReactive(input$gobutton, {

p()

test\_break()

py$test1 <- py$test[,c(3:20)]

gauslider\_pred <- 0.05

string <- paste0("gnbc\_model = GaussianNB(var\_smoothing =",gauslider\_pred,")")

py\_run\_string(string)

py\_run\_string("gnbc\_model.fit(X\_train, y\_train)")

py\_run\_string("results\_gnbc\_pred = gnbc\_model.predict(test1)")

balphaslider\_pred <- 0.01

string <- paste0("bnbc\_model = BernoulliNB(alpha =",balphaslider\_pred,")")

py\_run\_string(string)

py\_run\_string("bnbc\_model.fit(X\_train, y\_train)")

py\_run\_string("results\_bnbc\_pred = bnbc\_model.predict(test1)")

malphaslider\_pred <- 0.01

string <- paste0("mnbc\_model = MultinomialNB(alpha =",malphaslider\_pred,")")

py\_run\_string(string)

py\_run\_string("mnbc\_model.fit(X\_train, y\_train)")

py\_run\_string("results\_mnbc\_pred = mnbc\_model.predict(test1)")

md\_pred <- 3

mf\_pred <- 5

nest\_pred <- 20

bootstrap\_pred <- "True"

criterion\_pred <- "entropy"

string <- paste0("rfc\_model = RandomForestClassifier(max\_depth = ",md\_pred,", max\_features = ",mf\_pred,", n\_estimators = ",nest\_pred,", bootstrap = ",bootstrap\_pred,", criterion = \"",criterion\_pred,"\")")

py\_run\_string(string)

py\_run\_string("rfc\_model.fit(X\_train, y\_train)")

py\_run\_string("results\_rfc\_pred = rfc\_model.predict(test1)")

cost\_pred <- 0.01

kern\_pred <- "linear"

gamma\_pred <- "auto"

string <- paste0("svm\_model = SVC(C = ",cost\_pred,", kernel = \"",kern\_pred,"\", gamma = \"",gamma\_pred,"\")")

py\_run\_string(string)

py\_run\_string("svm\_model.fit(X\_train, y\_train)")

py\_run\_string("results\_svm\_pred = svm\_model.predict(test1)")

})

output$gnb\_pred <- renderPrint({

predict\_strength()

boom <- py$results\_gnbc\_pred

if (boom == 0) {

boomer <- "Weak"

}else{

if (boom == 1) {

boomer <- "Average"

}else{

if (boom == 2) {

boomer <- "Strong"

}else{

boomer <- "error"

}

}

}

boomer

})

output$bnb\_pred <- renderPrint({

predict\_strength()

#py$results\_bnbc\_pred

boom <- py$results\_bnbc\_pred

if (boom == 0) {

boomer <- "Weak"

}else{

if (boom == 1) {

boomer <- "Average"

}else{

if (boom == 2) {

boomer <- "Strong"

}else{

boomer <- "error"

}

}

}

boomer

})

output$rfc\_pred <- renderPrint({

predict\_strength()

#py$results\_rfc\_pred

boom <- py$results\_rfc\_pred

if (boom == 0) {

boomer <- "Weak"

}else{

if (boom == 1) {

boomer <- "Average"

}else{

if (boom == 2) {

boomer <- "Strong"

}else{

boomer <- "error"

}

}

}

boomer

})

output$mnb\_pred <- renderPrint({

predict\_strength()

#py$results\_mnbc\_pred

boom <- py$results\_mnbc\_pred

if (boom == 0) {

boomer <- "Weak"

}else{

if (boom == 1) {

boomer <- "Average"

}else{

if (boom == 2) {

boomer <- "Strong"

}else{

boomer <- "error"

}

}

}

boomer

})

output$svm\_pred <- renderPrint({

predict\_strength()

#py$results\_svm\_pred

boom <- py$results\_svm\_pred

if (boom == 0) {

boomer <- "Weak"

}else{

if (boom == 1) {

boomer <- "Average"

}else{

if (boom == 2) {

boomer <- "Strong"

}else{

boomer <- "error"

}

}

}

boomer

})

d <- reactive({

x <- as.numeric(input$testperc)

py\_run\_string("cub = [1,2,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]")

py\_run\_string("xpass = passwords[passwords.columns[cub]]")

py\_run\_string(paste0("X\_train, X\_CV, y\_train, y\_CV = train\_test\_split(xpass[xpass.columns[1:24]], xpass[xpass.columns[0]], test\_size = ",x,")"))

if (input$sample\_check == TRUE) {

if (input$sample == "smote") {

py\_run\_string("sam = SMOTE(n\_jobs = -1, k\_neighbors=3)")

}else{

py\_run\_string("sam = RandomUnderSampler(random\_state=0)")

}

py\_run\_string("X\_train, y\_train = sam.fit\_sample(X\_train, y\_train)")

}

})

f <- reactive({

x <- as.numeric(input$testperc1)

py\_run\_string("cub = [1,2,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]")

py\_run\_string("xpass = passwords[passwords.columns[cub]]")

py\_run\_string(paste0("X\_train, X\_CV, y\_train, y\_CV = train\_test\_split(xpass[xpass.columns[1:24]], xpass[xpass.columns[0]], test\_size = ",x,")"))

if (input$sample\_check\_rfc == TRUE) {

if (input$sample\_rfc == "smote") {

py\_run\_string("sam = SMOTE(n\_jobs = -1, k\_neighbors=3)")

}else{

py\_run\_string("sam = RandomUnderSampler(random\_state=0)")

}

py\_run\_string("X\_train, y\_train = sam.fit\_sample(X\_train, y\_train)")

}

})

s <- reactive({

x <- as.numeric(input$testperc2)

py\_run\_string("cub = [1,2,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]")

passwords <- py$passwords

py$samp <- passwords[sample(1:dim(passwords)[1],input$sampling, replace = T),]

py\_run\_string("xpass = samp[samp.columns[cub]]")

py\_run\_string(paste0("X\_train, X\_CV, y\_train, y\_CV = train\_test\_split(xpass[xpass.columns[1:24]], xpass[xpass.columns[0]], test\_size = ",x,")"))

if (input$sample\_check\_svc == TRUE) {

if (input$sample\_svc == "smote") {

py\_run\_string("sam = SMOTE(n\_jobs = -1, k\_neighbors=3)")

}else{

py\_run\_string("sam = RandomUnderSampler(random\_state=0)")

}

py\_run\_string("X\_train, y\_train = sam.fit\_sample(X\_train, y\_train)")

}

})

p <- reactive({

x <- 0.3

py\_run\_string("cub = [1,2,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]")

passwords <- py$passwords

py$samp <- passwords[sample(1:dim(passwords)[1],10000, replace = T),]

py\_run\_string("xpass = samp[samp.columns[cub]]")

py\_run\_string(paste0("X\_train, X\_CV, y\_train, y\_CV = train\_test\_split(xpass[xpass.columns[1:24]], xpass[xpass.columns[0]], test\_size = ",x,")"))

})

test\_break <- reactive({

py\_run\_string("def n\_lower\_chars(string):

return sum(map(str.islower, string))")

py\_run\_string("def n\_upper\_chars(string):

return sum(map(str.isupper, string))")

py\_run\_string("def n\_num\_chars(string):

return sum(map(str.isdigit, string))")

py\_run\_string("def n\_space\_chars(string):

return sum(map(str.isspace, string))")

py\_run\_string("def split(word):

return [char for char in word]")

py\_run\_string("def char\_detect(charx):

char\_type = \"S\"

if charx.isupper():

char\_type = \"U\"

elif charx.islower():

char\_type = \"L\"

elif charx.isdigit():

char\_type = \"N\"

elif charx.isspace():

char\_type = \"s\"

else:

char\_type = \"S\"

return char\_type")

py\_run\_string("def count\_switches(string):

string = split(str(string))

count = 0

if len(string) > 1:

i = 0

j = 1

for y in range(0,len(string) -1 ):

if char\_detect(string[i]) != char\_detect(string[j]):

count = count + 1

i, j = i+1,j+1

return count")

py\_run\_string("def reps(string):

temp = split(str(string))

i = 0

j = 1

k = 2

flag = 0

for y in range(2,len(temp)-1):

if temp[i] == temp[j] and temp[j] == temp[k]:

flag = 1

y = len(temp) + 1

else:

i += 1

j += 1

k += 1

return flag")

py\_run\_string("def breakStr(password):

word = []

words = []

let = False #this keeps track of if the last character was a letter

password = str(password)

password1 = password +\"1\"

for i in password1:

if i.isalpha() == False and let == False:

continue

elif i.isalpha() == True:

word.append(i)

let = True

elif i.isalpha() == False and let == True:

wordStr = ''.join(word)

wordStr = wordStr.lower()

if len(wordStr) >= 3:

words.append(wordStr)

word = []

let = False

return (words) ")

py\_run\_string("def checkWord(word):

result = 0

w1 = \"\"

if word in english\_vocab:

result = 1

else:

k = 2

while k < len(word):

if (word[0:k]) in english\_vocab:

result += 1

w1 = word[0:k]

end = -(len(word) - k+1)

k = len(word)

else:

end = -len(word)

k += 1

k = -3

while k > end:

if word[k:] in english\_vocab:

result += 1

k = -len(word) -1

else:

k -= 1

return(result)")

py\_run\_string("ex1 = ['rst', 'word', \"worda\"]")

py\_run\_string("def sumWords(password):

sumWords = 0

words = breakStr(password)

print(password)

for word in words:

sumWords += checkWord(word)

return (sumWords)")

test\_string <- trimws(input$pwd)

py\_run\_string("test = pd.DataFrame()")

string <- paste0("test = test.append({'password' : \"",test\_string,"\", 'strength' : '' }, ignore\_index=True)")

py\_run\_string(string)

py\_run\_string("test['wordCount'] = 0")

py\_run\_string("test['FChar'] = test.password.astype(str).str[0].apply(char\_detect)")

py\_run\_string("test['LChar'] = test.password.astype(str).str[-1].apply(char\_detect)")

py\_run\_string("test['TotalLength'] = test.password.astype(str).apply(len)")

py\_run\_string("test['LowerChar'] = test.password.astype(str).apply(n\_lower\_chars)")

py\_run\_string("test['UpperChar'] = test.password.astype(str).apply(n\_upper\_chars)")

py\_run\_string("test['NumChar'] = test.password.astype(str).apply(n\_num\_chars)")

py\_run\_string("test['SpaceChar'] = test.password.astype(str).apply(n\_space\_chars)")

py\_run\_string("test['SpecialChar'] = test['TotalLength'] - (test['LowerChar'] + test['UpperChar'] + test['SpaceChar'] + test['NumChar'])")

py\_run\_string("test['TotalSwitches'] = test.password.astype(str).apply(count\_switches)")

py\_run\_string("test['Repeats'] = test.password.astype(str).apply(reps)")

py\_run\_string("def check\_U(Char):

x = 0

if Char == \"U\":

x = 1

return x")

py\_run\_string("def check\_L(Char):

x = 0

if Char == \"L\":

x = 1

return x")

py\_run\_string("def check\_N(Char):

x = 0

if Char == \"N\":

x = 1

return x")

py\_run\_string("def check\_S(Char):

x = 0

if Char == \"S\":

x = 1

return x")

py\_run\_string("def check\_s(Char):

x = 0

if Char == \"s\":

x = 1

return x")

py\_run\_string("test['FCharL'] = check\_L(test['FChar'].loc[0])")

py\_run\_string("test['FCharN'] = check\_N(test['FChar'].loc[0])")

py\_run\_string("test['FCharS'] = check\_S(test['FChar'].loc[0])")

py\_run\_string("test['FCharU'] = check\_U(test['FChar'].loc[0])")

py\_run\_string("test['FChars'] = check\_s(test['FChar'].loc[0])")

py\_run\_string("test['LCharL'] = check\_L(test['LChar'].loc[0])")

py\_run\_string("test['LCharN'] = check\_N(test['LChar'].loc[0])")

py\_run\_string("test['LCharS'] = check\_S(test['LChar'].loc[0])")

py\_run\_string("test['LCharU'] = check\_U(test['LChar'].loc[0])")

py\_run\_string("test['LChars'] = check\_s(test['LChar'].loc[0])")

py\_run\_string("test.drop(['TotalLength', 'FChar', 'LChar'], axis=1, inplace = True)")

})

nb\_react <- reactive({

d()

if(input$nb == "gaussian"){

gauslider <- input$gauslider

string <- paste0("nbc = GaussianNB(var\_smoothing =",gauslider,")")

py\_run\_string(string)

}else{

if (input$nb == "bernoulli") {

balphaslider <- input$balphaslider

string <- paste0("nbc = BernoulliNB(alpha =",balphaslider,")")

py\_run\_string(string)

}else{

malphaslider <- input$malphaslider

string <- paste0("nbc = MultinomialNB(alpha =",malphaslider,")")

py\_run\_string(string)

}

}

py\_run\_string("nbc.fit(X\_train, y\_train)")

py\_run\_string("results = nbc.predict(X\_CV)")

py\_run\_string("acc = round(metrics.accuracy\_score(y\_CV,results)\*100, 2)")

py\_run\_string("conf = confusion\_matrix( y\_pred = results,y\_true= np.array(y\_CV), labels = [0,1,2])")

py\_run\_string("classif = classification\_report( y\_pred = results,y\_true= np.array(y\_CV), labels = [0,1,2])")

})

rf\_react <- reactive({

f()

md <- input$md

mf <- input$mf

nest <- input$nest

bootstrap <- "False"

if (input$bootstrap == "t") {

bootstrap <- "True"

}

criterion <- "gini"

if (input$criterion == "e") {

criterion <- "entropy"

}

string <- paste0("rfc = RandomForestClassifier(max\_depth = ",md,", max\_features = ",mf,", n\_estimators = ",nest,", bootstrap = ",bootstrap,", criterion = \"",criterion,"\")")

py\_run\_string(string)

py\_run\_string("rfc.fit(X\_train, y\_train)")

py\_run\_string("results\_rf = rfc.predict(X\_CV)")

py\_run\_string("acc\_rf = round(metrics.accuracy\_score(y\_CV,results\_rf)\*100, 2)")

py\_run\_string("conf\_rf = confusion\_matrix( y\_pred = results\_rf,y\_true= np.array(y\_CV), labels = [0,1,2])")

py\_run\_string("classif\_rf = classification\_report( y\_pred = results\_rf,y\_true= np.array(y\_CV), labels = [0,1,2])")

})

sv\_react <- reactive({

s()

cost <- input$cost

kern <- "linear"

if (input$kern == "p") {

kern <- "linear"

}else{

if (input$kern == "r") {

kern <- "rbf"

}else{

if (input$kern == "s") {

kern <- "sigmoid"

}

}

}

gamma <- "auto"

if (input$gamma == "scale") {

gamma <- "scale"

}

string <- paste0("svmodel = SVC(C = ",cost,", kernel = \"",kern,"\", gamma = \"",gamma,"\")")

py\_run\_string(string)

py\_run\_string("svmodel.fit(X\_train, y\_train)")

py\_run\_string("results\_sv = svmodel.predict(X\_CV)")

py\_run\_string("acc\_sv = round(metrics.accuracy\_score(y\_CV,results\_sv)\*100, 2)")

py\_run\_string("conf\_sv = confusion\_matrix( y\_pred = results\_sv,y\_true= np.array(y\_CV), labels = [0,1,2])")

py\_run\_string("classif\_sv = classification\_report( y\_pred = results\_sv,y\_true= np.array(y\_CV), labels = [0,1,2])")

})

observeEvent(input$gobutton,{

if (input$pwd != '') {

test\_break()

}

})

prd\_react <- reactive({

#p()

if (input$pwd != "") {

test\_break()

}

# cost <- 0.001

# kern <- "linear"

# gamma <- "auto"

#

# string <- paste0("svmodel = SVC(C = ",cost,", kernel = \"",kern,"\", gamma = \"",gamma,"\")")

# py\_run\_string(string)

# py\_run\_string("svmodel.fit(X\_train, y\_train)")

# py\_run\_string("results\_sv\_pred = svmodel.predict(X\_CV)")

# py\_run\_string("acc\_sv\_pred = round(metrics.accuracy\_score(y\_CV,results\_sv)\*100, 2)")

# py\_run\_string("conf\_sv\_pred = confusion\_matrix( y\_pred = results\_sv,y\_true= np.array(y\_CV), labels = [0,1,2])")

# py\_run\_string("classif\_sv\_pred = classification\_report( y\_pred = results\_sv,y\_true= np.array(y\_CV), labels = [0,1,2])")

})

output$test\_data <- renderDataTable({

if (input$pwd != "") {

test\_break()

py$test[,-3]

}

})

output$test\_fe <- renderPrint({

prd\_react()

cat(py$First\_Letter, py$Last\_Letter, py$length)

})

output$barplot <- renderPlot({

if((input$dataset == "og" & input$cols == "strength") | input$dataset != "og"){

main\_print = paste0(" The Distribution of ",input$cols)

barplot(table(passwords[,input$cols]), las = 1, main = main\_print)

}

})

output$data <- renderDataTable({

if (input$dataset == "og") {

passwords[,1:2]

}else{

passwords

}

})

output$acc <- renderPrint({

nb\_react()

py$acc

})

output$conf <- renderPrint({

nb\_react()

py$conf

})

output$classif <- renderPrint({

nb\_react()

rconf <- py$conf

C0P <- round(rconf[1,1]/(sum(rconf[,1])),5)

C0R <- round(rconf[1,1]/(sum(rconf[1,])),5)

f10 <- round((2\*C0P\*C0R)/(C0P+C0R),5)

C1P <- round(rconf[2,2]/(sum(rconf[,2])),5)

C1R <- round(rconf[2,2]/(sum(rconf[2,])),5)

f11 <- round((2\*C1P\*C1R)/(C1P+C1R),5)

C2P <- round(rconf[3,3]/(sum(rconf[,3])),5)

C2R <- round(rconf[3,3]/(sum(rconf[3,])),5)

f12 <- round((2\*C2P\*C2R)/(C2P+C2R),5)

cat(cat("Class 0 : \n Precision\t: ",C0P,"\n Recall\t\t: ",C0R,"\n F1 Score\t: ",f10)

, cat("\n\n\nClass 1 : \n Precision\t: ",C1P,"\n Recall\t\t: ",C1R,"\n F1 Score\t: ",f11)

, cat("\n\n\nClass 2 : \n Precision\t: ",C2P,"\n Recall\t\t: ",C2R,"\n F1 Score\t: ",f12))

})

output$acc\_rf <- renderPrint({

rf\_react()

py$acc\_rf

})

output$conf\_rf <- renderPrint({

rf\_react()

py$conf\_rf

})

output$classif\_rf <- renderPrint({

rf\_react()

rconf <- py$conf\_rf

C0P <- round(rconf[1,1]/(sum(rconf[,1])),5)

C0R <- round(rconf[1,1]/(sum(rconf[1,])),5)

f10 <- round((2\*C0P\*C0R)/(C0P+C0R),5)

C1P <- round(rconf[2,2]/(sum(rconf[,2])),5)

C1R <- round(rconf[2,2]/(sum(rconf[2,])),5)

f11 <- round((2\*C1P\*C1R)/(C1P+C1R),5)

C2P <- round(rconf[3,3]/(sum(rconf[,3])),5)

C2R <- round(rconf[3,3]/(sum(rconf[3,])),5)

f12 <- round((2\*C2P\*C2R)/(C2P+C2R),5)

cat(cat("Class 0 : \n Precision\t: ",C0P,"\n Recall\t\t: ",C0R,"\n F1 Score\t: ",f10)

, cat("\n\n\nClass 1 : \n Precision\t: ",C1P,"\n Recall\t\t: ",C1R,"\n F1 Score\t: ",f11)

, cat("\n\n\nClass 2 : \n Precision\t: ",C2P,"\n Recall\t\t: ",C2R,"\n F1 Score\t: ",f12))

})

output$acc\_sv <- renderPrint({

sv\_react()

py$acc\_sv

})

output$conf\_sv <- renderPrint({

sv\_react()

py$conf\_sv

})

output$classif\_sv <- renderPrint({

sv\_react()

rconf <- py$conf\_sv

C0P <- round(rconf[1,1]/(sum(rconf[,1])),5)

C0R <- round(rconf[1,1]/(sum(rconf[1,])),5)

f10 <- round((2\*C0P\*C0R)/(C0P+C0R),5)

C1P <- round(rconf[2,2]/(sum(rconf[,2])),5)

C1R <- round(rconf[2,2]/(sum(rconf[2,])),5)

f11 <- round((2\*C1P\*C1R)/(C1P+C1R),5)

C2P <- round(rconf[3,3]/(sum(rconf[,3])),5)

C2R <- round(rconf[3,3]/(sum(rconf[3,])),5)

f12 <- round((2\*C2P\*C2R)/(C2P+C2R),5)

cat(cat("Class 0 : \n Precision\t: ",C0P,"\n Recall\t\t: ",C0R,"\n F1 Score\t: ",f10)

, cat("\n\n\nClass 1 : \n Precision\t: ",C1P,"\n Recall\t\t: ",C1R,"\n F1 Score\t: ",f11)

, cat("\n\n\nClass 2 : \n Precision\t: ",C2P,"\n Recall\t\t: ",C2R,"\n F1 Score\t: ",f12))

})

}

# Run the application

shinyApp(ui = ui, server = server)

1. ***References***

Bansal, B. (2019, June 27). Password Strength Classifier Dataset. Retrieved from https://www.kaggle.com/bhavikbb/password-strength-classifier-dataset/version/1.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. doi: 10.1613/jair.953

PARS Home. (n.d.). Retrieved from http://www.pars.gatech.edu/.