

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
5.1. Naïve Bayes Classification Algorithm	9
5.2. Support Vector Machine	10
5.3. Random Forest Classifier	12
6. Evaluation	14
6.1. Naïve Bayes Classification Algorithm	14
6.2. Support Vector Machine	14
6.3. Random Forest Classifier	14
6.4. Results	14
7. Conclusion	15
8. Appendix	17
8.1. Data Preparation Script (Python)	17
8.2. R Shiny App Script (R)	21
9. References	43

1. Introduction

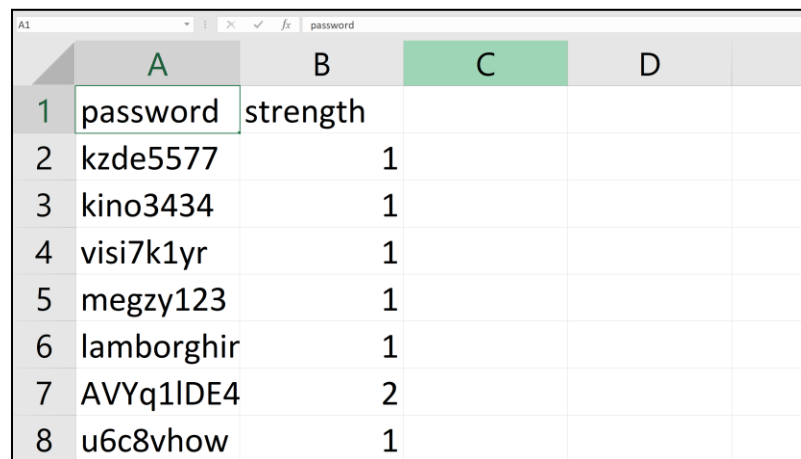
For every organization, security is the primary concern and to ensure security they must ensure there is no way an unauthorized personnel or entity can access their system. Each employee working in that organization is responsible to ensure safety. An outside entity obtaining the password of an employee 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.

2. 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).

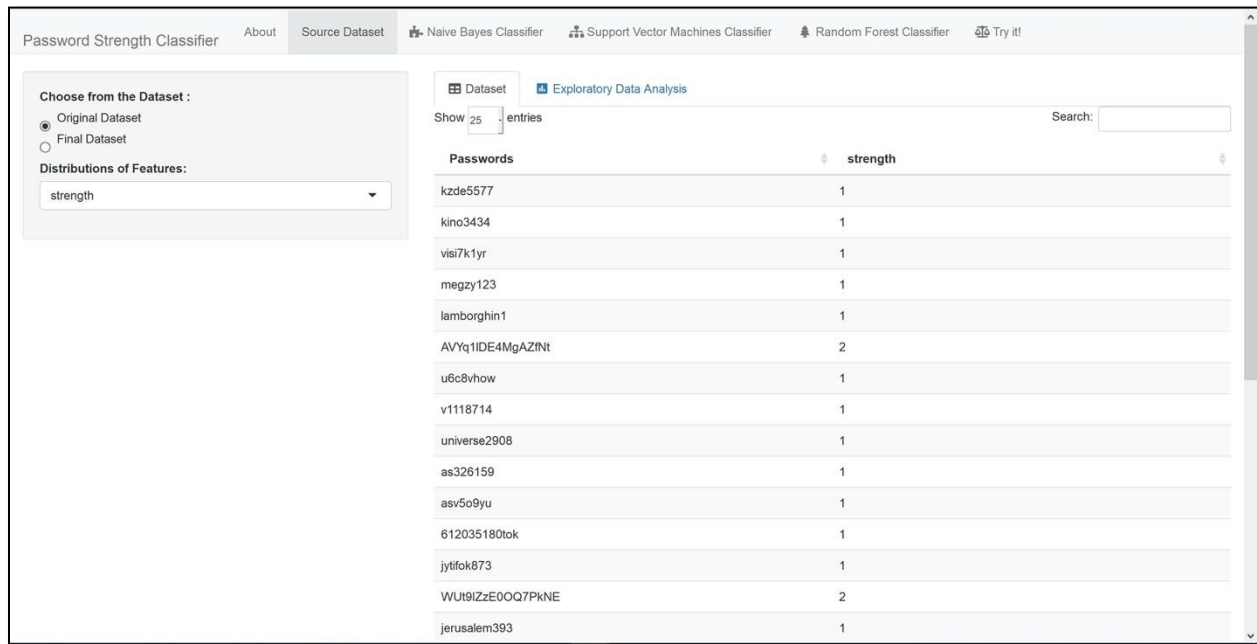
On the homepage of PARS the creators describe the tool as follows: “In this module, leveraging a large corpus (~115M) of real-world passwords, we develop several analytical functions to characterize password datasets. The strength distribution of passwords in terms of specific metrics or meters can also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



	A	B	C	D
1	password	strength		
2	kzde5577	1		
3	kino3434	1		
4	visi7k1yr	1		
5	megzy123	1		
6	lamborghir	1		
7	AVYq1lDE4	2		
8	u6c8vhow	1		

Figure 2.1 Original Dataset

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



Choose from the Dataset :

☒ Original Dataset
☐ Final Dataset

Distributions of Features:

strength

Dataset [Exploratory Data Analysis](#)

Show 25 entries Search:

Passwords	strength
kzde5577	1
kino3434	1
visi7k1yr	1
megzy123	1
lamborghini1	1
AVYq1IDE4MgAZfNt	2
u6c8vhow	1
v1118714	1
universe2908	1
as326159	1
asv5o9yu	1
612035180tok	1
jytfok873	1
WUit9IzZE0OQ7PkNE	2
jerusalem393	1

Figure 2.2 The original dataset in the app.

3. 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.

The screenshot shows the 'Password Strength Classifier' application. The 'Source Dataset' tab is active. On the left, under 'Choose from the Dataset', the 'Final Dataset' is selected. Below it, 'Distributions of Features' shows 'strength' as the selected feature. The main area displays a table of 25 entries. The table has columns: Passwords, strength, wordCount, FChar, LChar, LowerChar, UpperChar, NumChar, and SpaceChar. The data rows show various passwords and their corresponding feature values.

Passwords	strength	wordCount	FChar	LChar	LowerChar	UpperChar	NumChar	SpaceChar
kzde5577	1	0	L	N	4	0	4	0
kino3434	1	1	L	N	4	0	4	0
visi7k1yr	1	1	L	L	7	0	2	0
megzy123	1	1	L	N	5	0	3	0
lamborghini1	1	2	L	N	10	0	1	0
AVYq1IDE4MgAZINt	2	0	U	L	5	9	2	0
u6c8vhow	1	1	L	L	6	0	2	0
v1118714	1	0	L	N	1	0	7	0
universe2908	1	1	L	N	8	0	4	0
as326159	1	0	L	N	2	0	6	0
asv5o9yu	1	0	L	L	6	0	2	0
612035180tok	1	0	N	L	3	0	9	0
jytfok873	1	0	L	N	7	0	3	0
WUti9IzZE00Q7PkNE	2	1	U	U	3	10	3	0
tanualem303	1	1	L	N	0	0	3	0

Fig 3.1 Screenshot of Dataset after Feature Engineering

4. 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.

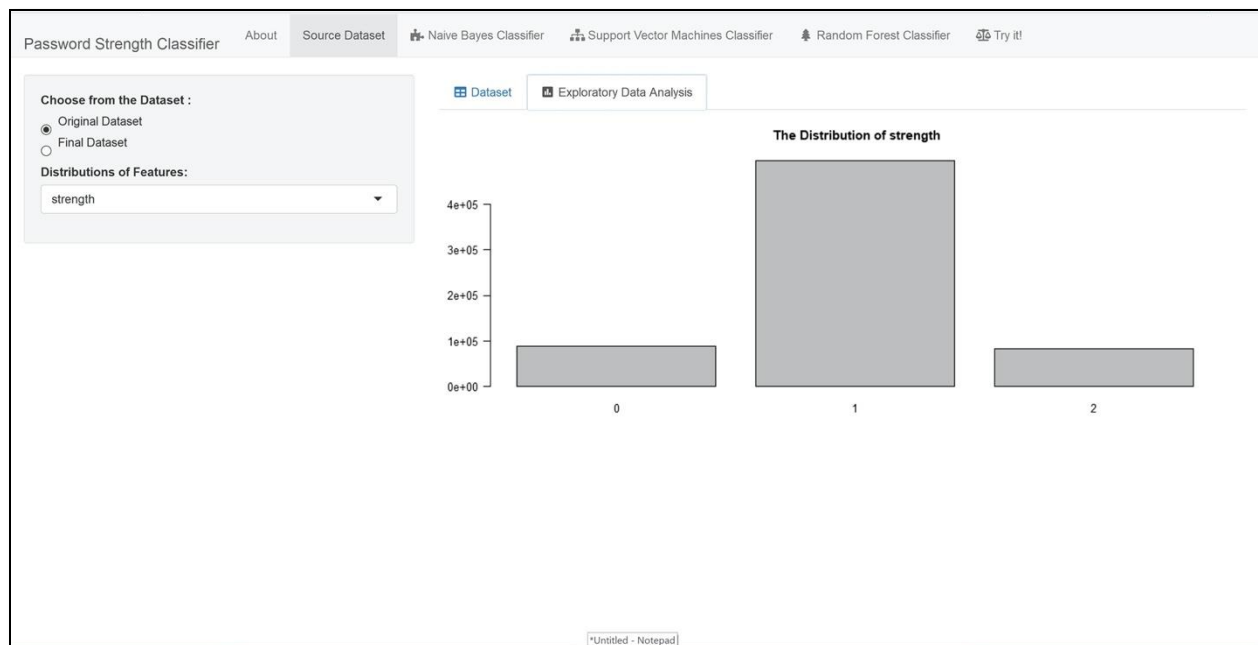


Figure 4.1 Distribution of the Strength

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 et al, 2002). This shows that it is a good method for balancing skewed data such as ours.

5. 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.

5.1. 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 smoothing 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.

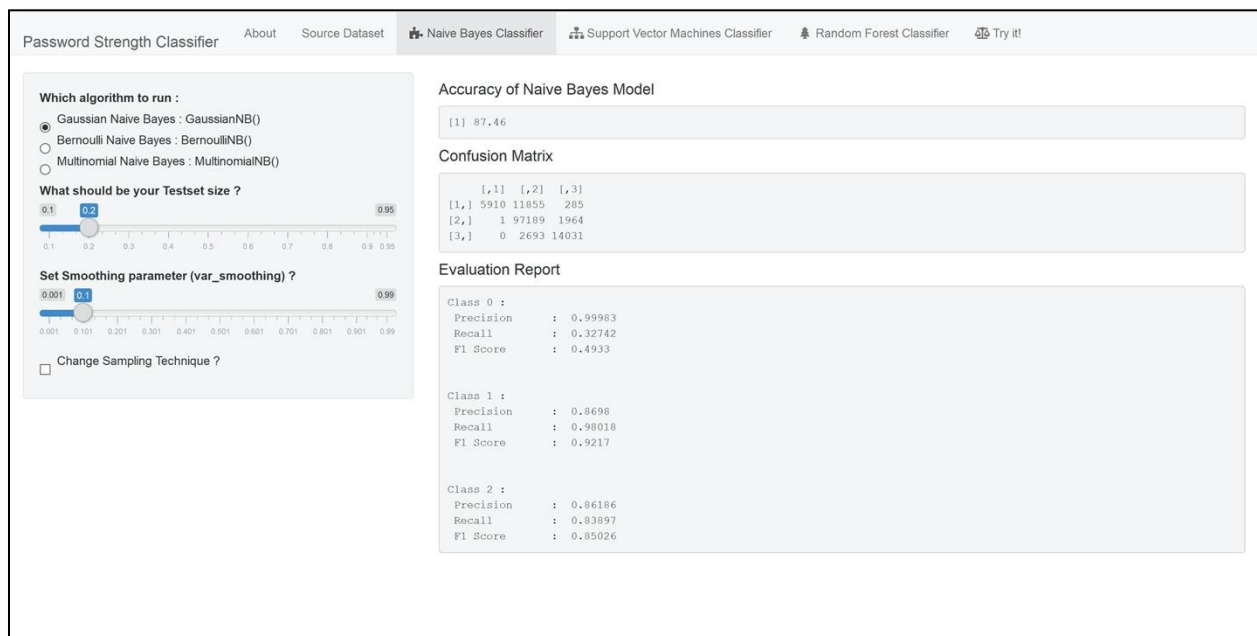


Figure 5.1.1 Naïve Bayes with no Sampling

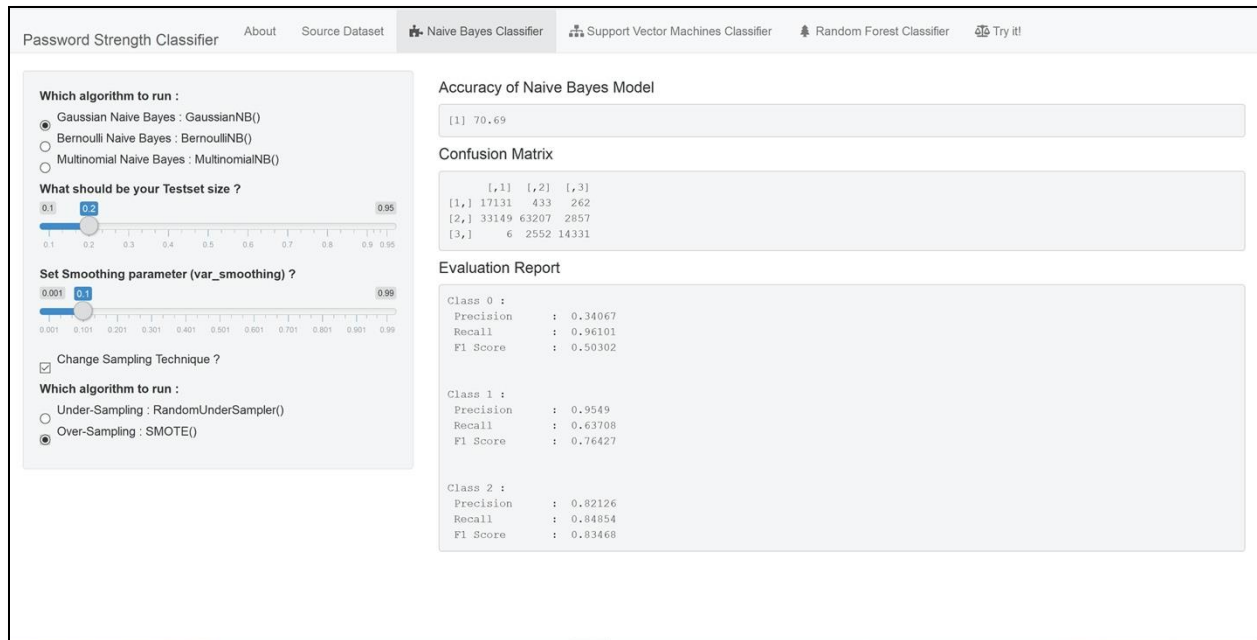


Figure 5.1.2 Naïve Bayes with Sampling

5.2. 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.

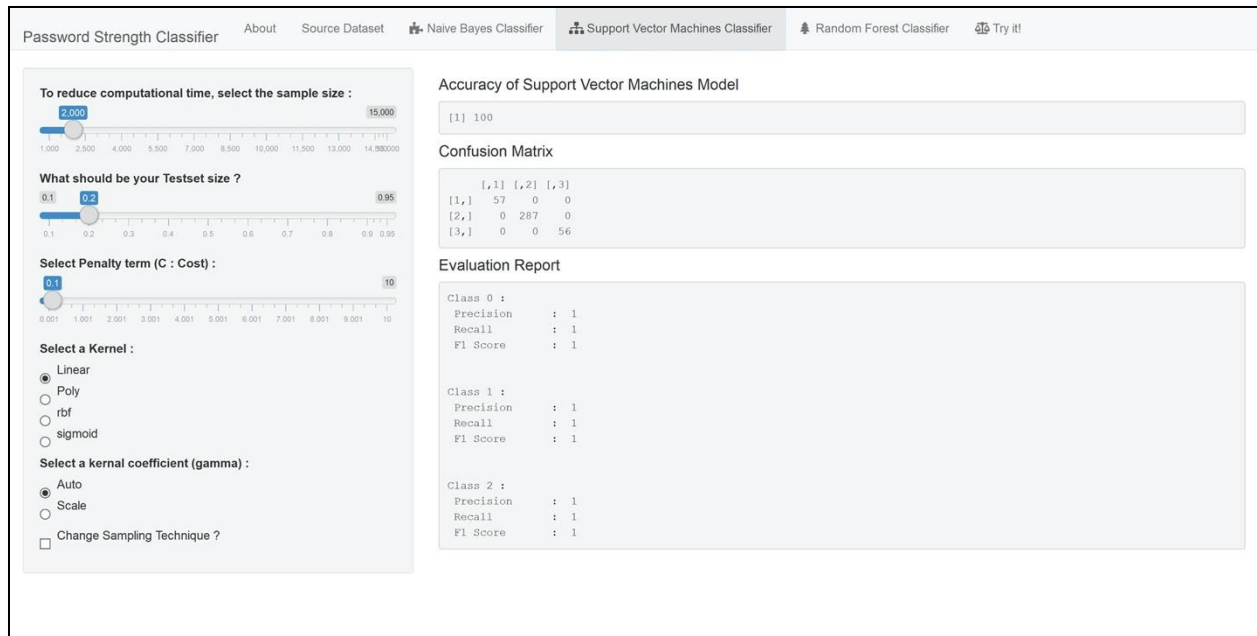


Figure 5.2.1 SVM With No Sampling

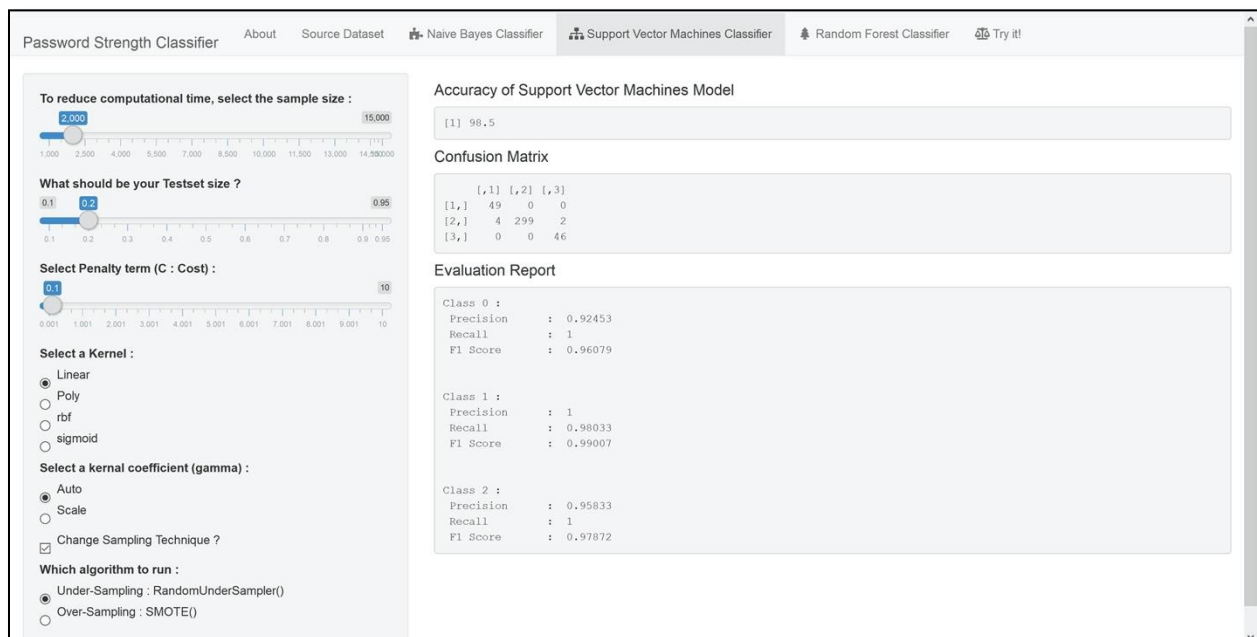


Figure 5.2.2 SVM With Sampling

5.3. 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

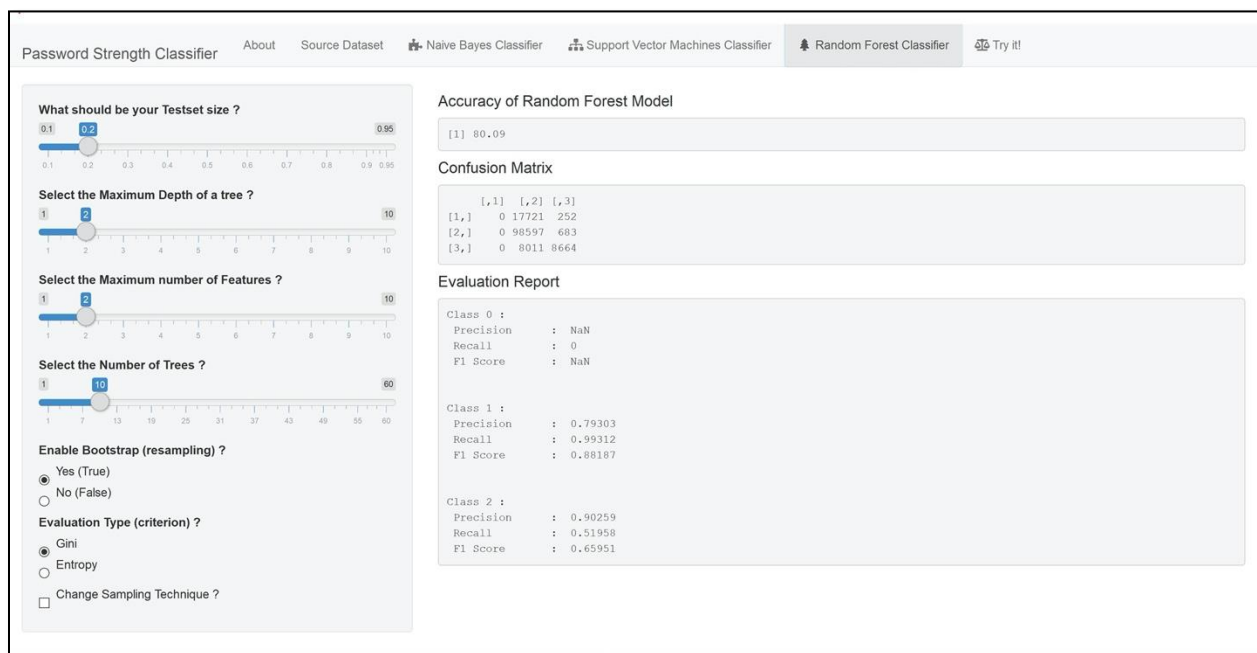


Figure 5.3.1 Random Forest with No Sampling

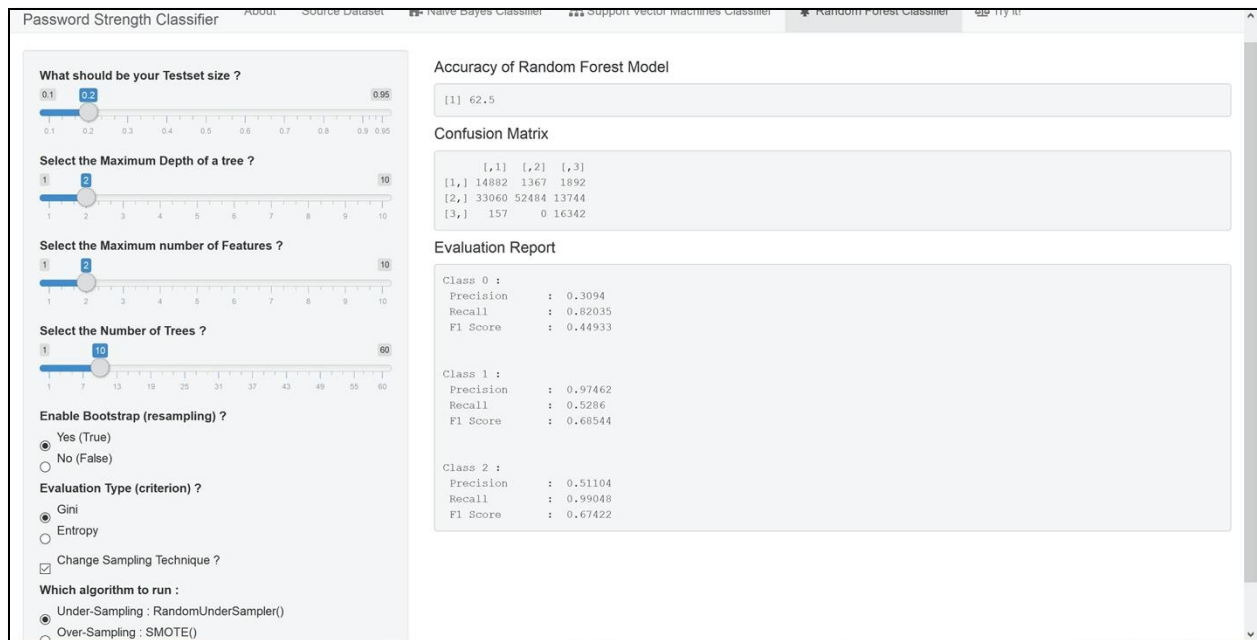


Figure 5.3.2 Random Forest with Sampling

6. Evaluations

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

6.1. 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.

6.2. 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%.

6.3. 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%.

6.4. 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

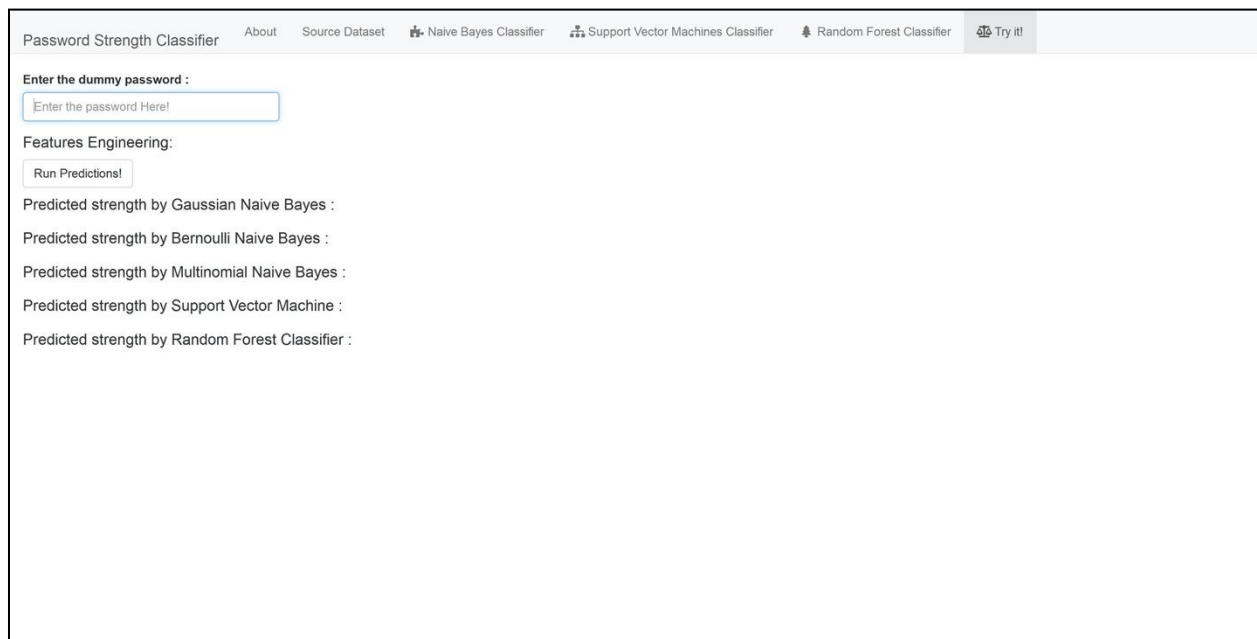
Table 6.4.1 Evaluation Metrics for all the models used

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.

7. 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 be used by people to predict the strength of their password. The accuracy is not too high that the models risk 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.



The screenshot displays the 'Password Strength Classifier' web application. The top navigation bar includes links for 'About', 'Source Dataset', and three classifiers: 'Naive Bayes Classifier', 'Support Vector Machines Classifier', and 'Random Forest Classifier'. A 'Try it!' button is also present. The main content area features a section titled 'Enter the dummy password :' with a text input field containing the placeholder 'Enter the password Here!'. Below this is a 'Features Engineering:' section with a 'Run Predictions!' button. At the bottom, there are five lines of text, each followed by a colon, representing the predicted strength from different models: 'Predicted strength by Gaussian Naive Bayes :', 'Predicted strength by Bernoulli Naive Bayes :', 'Predicted strength by Multinomial Naive Bayes :', 'Predicted strength by Support Vector Machine :', and 'Predicted strength by Random Forest Classifier :'. The results area is currently empty.

Figure 7.1 Input Screen for Password Strength Classifier

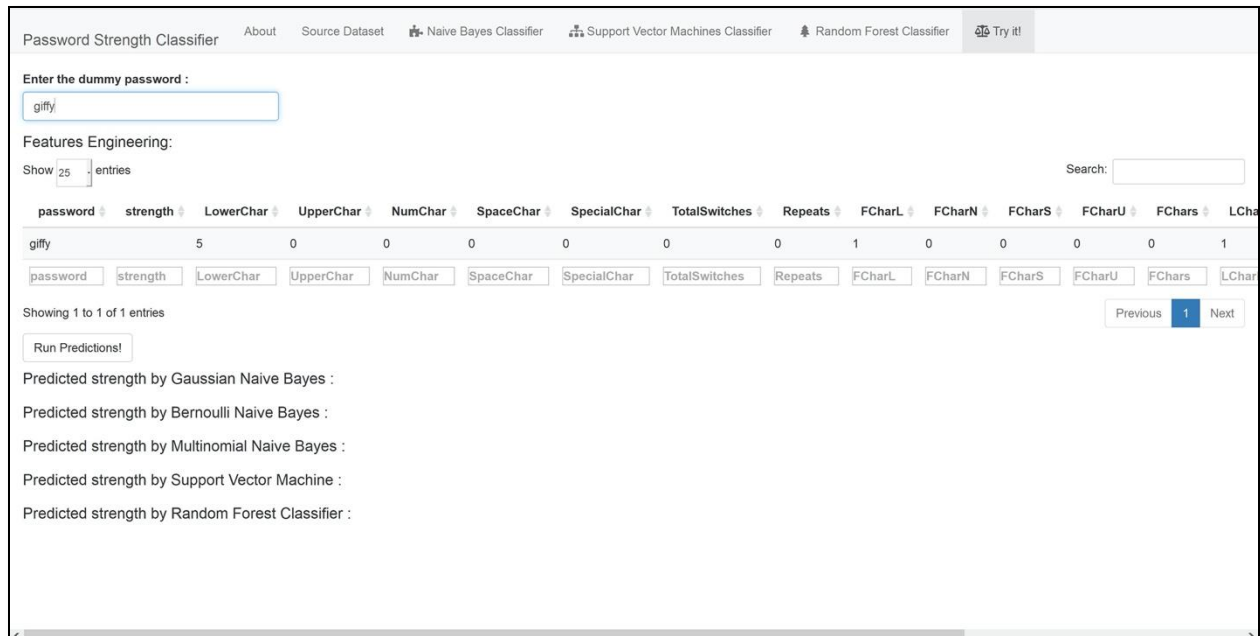


Figure 7.2 Attributes for Example Password

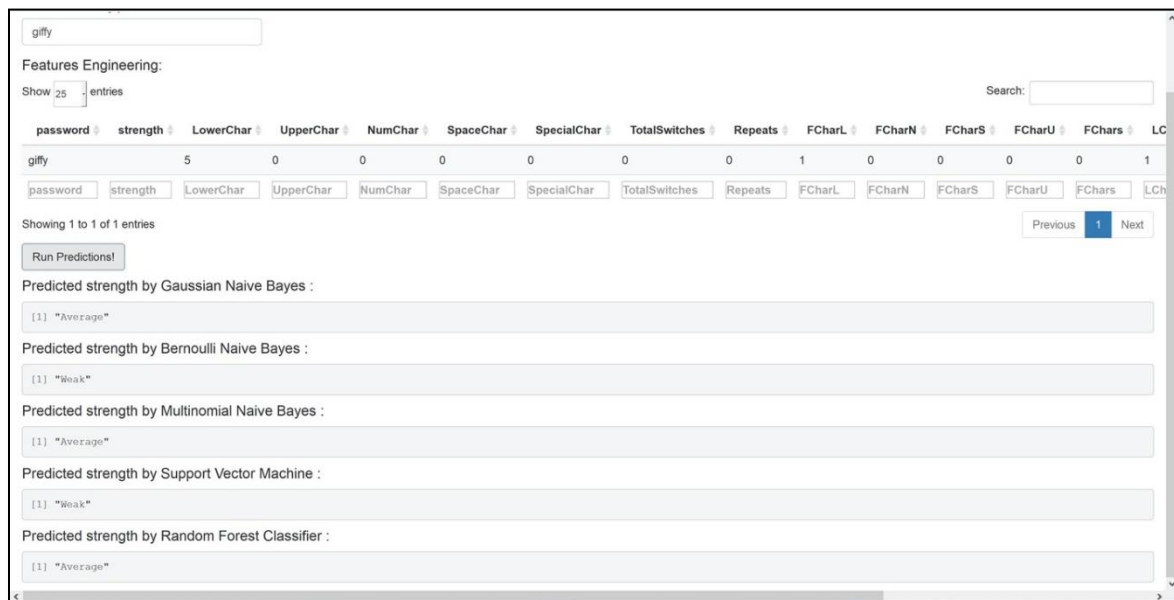


Figure 7.3 Output of Example Password Classification

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.

8. *Appendix*

8.1. *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 initial 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)
```

8.2. *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 = bnb_model.predict(test1)")

malphaslider_pred <- 0.01
string <- paste0("mnb_model = MultinomialNB(alpha =",malphaslider_pred,")")
py_run_string(string)
py_run_string("mnb_model.fit(X_train, y_train)")
py_run_string("results_mnb_pred = mnb_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_bnb_pred  
  boom <- py$results_bnb_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$mnbc_pred <- renderPrint({  
  predict_strength()  
  #py$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\nClass 1 : \n Precision\t: ",C1P,"\n Recall\t\t: ",C1R,"\n F1 Score\t: ",f11)
, cat("\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\nClass 1 : \n Precision\t: ",C1P,"\n Recall\t\t: ",C1R,"\n F1 Score\t: ",f11)
    , cat("\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)

```

9. 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/>.