Phishing Email Detection and Accuracy Comparison of different Machine Learning (ML) Algorithms

Practical 1: Phishing Email Detection Using Logistic Regression ML Algorithm

1. Ensure that Python environment is setup in your Operating System. Here we are using Kali Linux, so Python 3 comes pre-loaded into the environment.

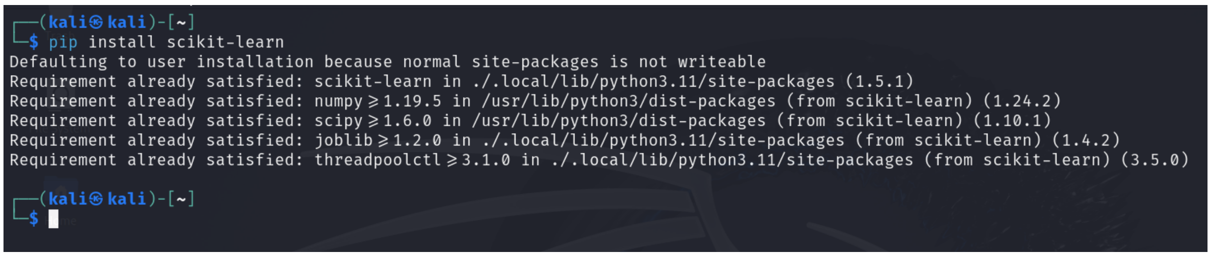
Also ensure all the required libraries are installed.

Use the command:

$ **pip install library_name**

For installing scikit-learn use the command: -

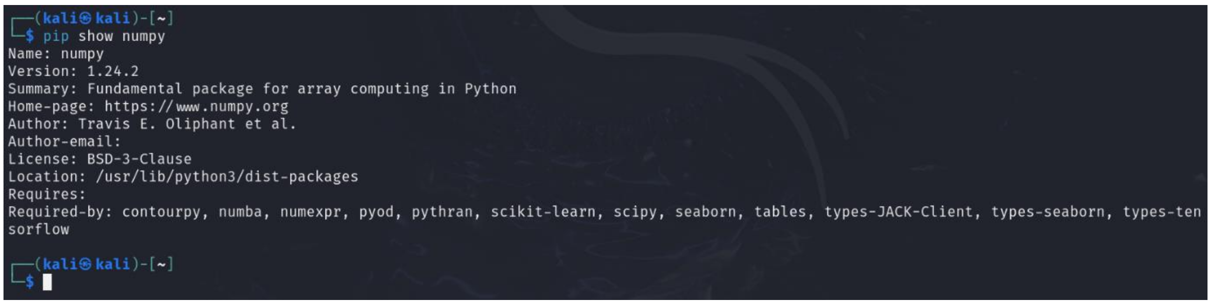
$ **pip install scikit-learn**



```
┌──(kali㉿kali)-[~]
└─$ pip install scikit-learn
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scikit-learn in ./.local/lib/python3.11/site-packages (1.5.1)
Requirement already satisfied: numpy≥1.19.5 in /usr/lib/python3/dist-packages (from scikit-learn) (1.24.2)
Requirement already satisfied: scipy≥1.6.0 in /usr/lib/python3/dist-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: joblib≥1.2.0 in ./.local/lib/python3.11/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl≥3.1.0 in ./.local/lib/python3.11/site-packages (from scikit-learn) (3.5.0)

┌──(kali㉿kali)-[~]
└─$ 
```

You can verify if a Python library is installed using: -

$ **Pip show library_name**



```
┌──(kali㉿kali)-[~]
└─$ pip show numpy
Name: numpy
Version: 1.24.2
Summary: Fundamental package for array computing in Python
Home-page: https://www.numpy.org
Author: Travis E. Oliphant et al.
Author-email:
License: BSD-3-Clause
Location: /usr/lib/python3/dist-packages
Requires:
Required-by: contourpy, numba, numexpr, pyod, pythran, scikit-learn, scipy, seaborn, tables, types-JACK-Client, types-seaborn, types-ten
sorflow

┌──(kali㉿kali)-[~]
└─$ 
```

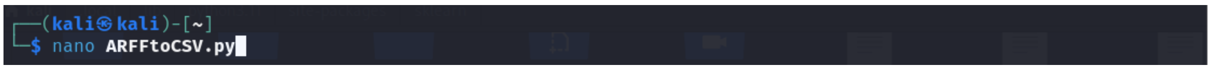
2. The email dataset has been taken from the UCI Machine Learning Repository. The “Phishing Website Features” word file downloaded from the above link shows what features of emails are taken into consideration and how they are converted to .arff format. This part is called as **Feature Extraction** and this is very important for analyzing real world data. The .arff file dataset is also downloaded from the above link. The Attribute-Relation File Format (ARFF) is an ASCII text file format that is essentially a CSV file with a header that describes the metadata (Features of the email). We need to convert this .arff file to .csv file so that our model can read the file.

The converted csv file is available at https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%204/Training%20Dataset.arff.csv

Download the file by clicking on the download arrow symbol (Download raw file)

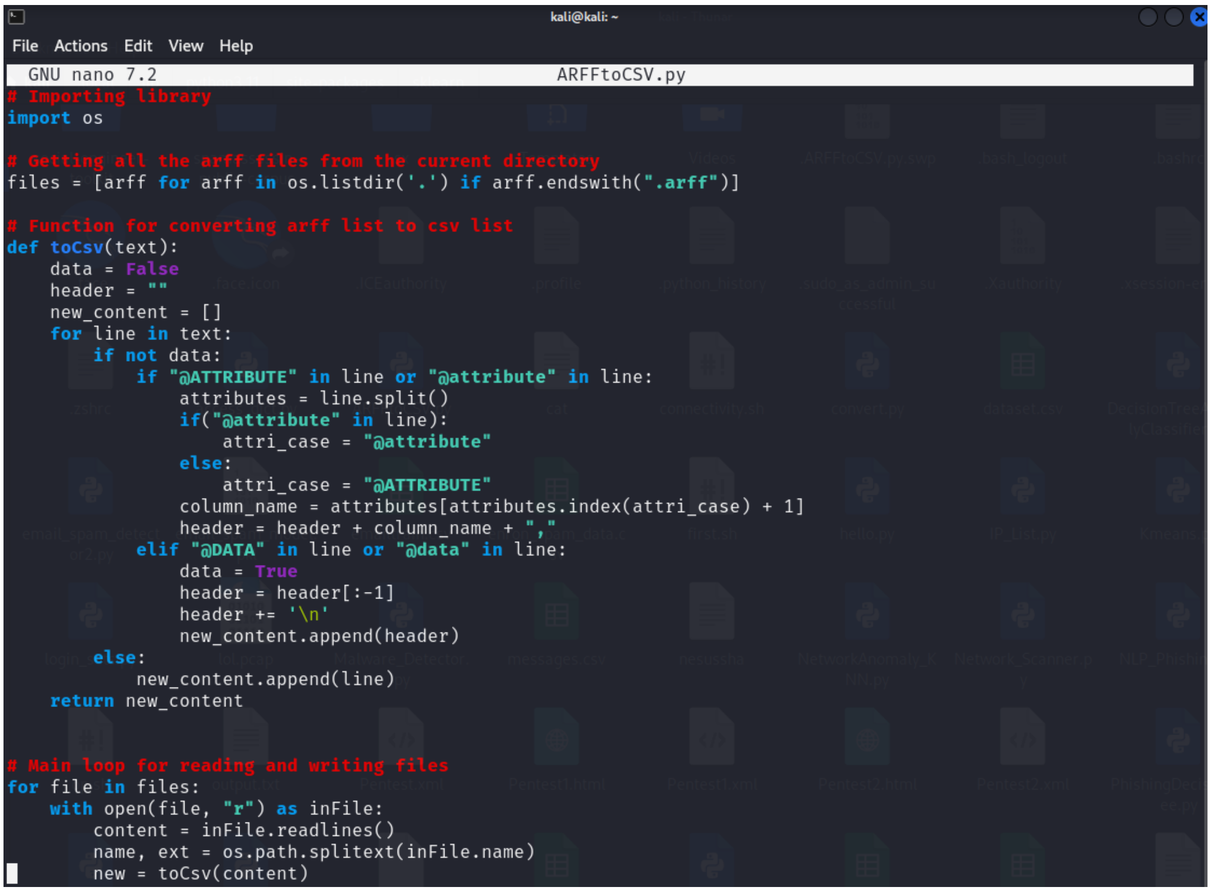
(Skip the next part of converting .arff to .csv if you have already downloaded the converted csv file above. Directly move to step 3)

We have written a Python script for this conversion: -



```
┌──(kali㉿kali)-[~]
└─$ nano ARFFtoCSV.py
```

The code for converting .arff file to .csv is available at https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%204/ARFFtoCSV.py

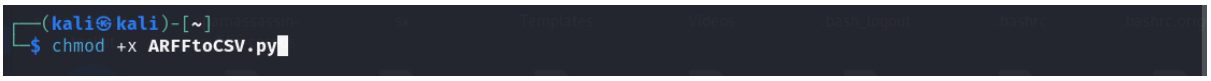


```
GNU nano 7.2                                   ARFFtoCSV.py
# Importing library
import os

# Getting all the arff files from the current directory
files = [arff for arff in os.listdir('.') if arff.endswith(".arff")]

# Function for converting arff list to csv list
def toCsv(text):
    data = False
    header = ""
    new_content = []
    for line in text:
        if not data:
            if "@ATTRIBUTE" in line or "@attribute" in line:
                attributes = line.split()
                if("@attribute" in line):
                    attri_case = "@attribute"
                else:
                    attri_case = "@ATTRIBUTE"
                column_name = attributes[attributes.index(attri_case) + 1]
                header = header + column_name + ","
            elif "@DATA" in line or "@data" in line:
                data = True
                header = header[:-1]
                header += '\n'
                new_content.append(header)
        else:
            new_content.append(line)
    return new_content


# Main loop for reading and writing files
for file in files:
    with open(file, "r") as inFile:
        content = inFile.readlines()
        name, ext = os.path.splitext(inFile.name)
        new = toCsv(content)
```

After entering this code, save the file (using “CTRL+x” followed by “y” and “Enter”)

Change it to an executable file using: -

$ **chmod +x ARFFtoCSV.py**

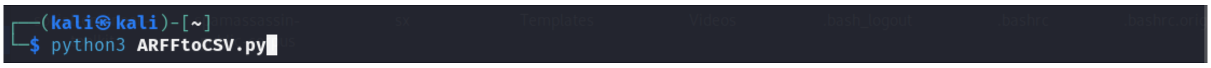


```
┌──(kali㉿kali)-[~]
└─$ chmod +x ARFFtoCSV.py
```

Ensure that the location or path of this file is same as the .arff file to convert it to csv file.

Convert the file by running this program using: -

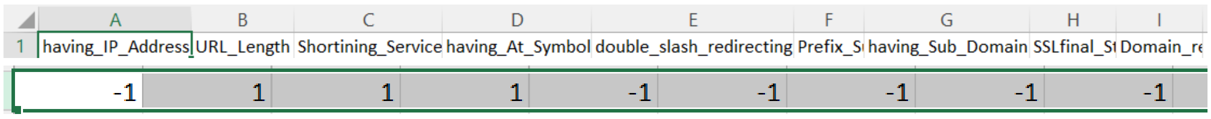
$ **python3 ARFFtoCSV.py**



```
┌──(kali㉿kali)-[~]
└─$ python3 ARFFtoCSV.py
```

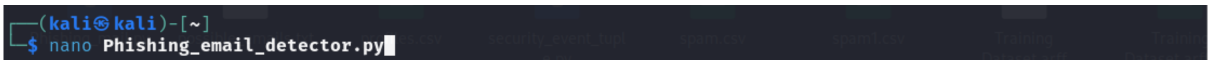
This file contains various parameters for checking if an email is a phishing email or not.

Some of these parameters can be seen by opening the .csv file: -



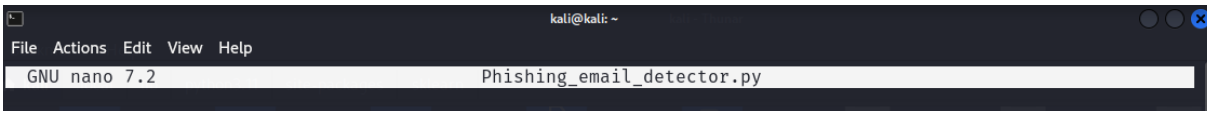
	A	B	C	D	E	F	G	H	I
1	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Si	having_Sub_Domain	SSLfinal_St	Domain_re
	-1	1	1	1	-1	-1	-1	-1	-1

3. Now open any editor of your choice with the name of the Python file you want to create. Here we have used the Nano editor.



```
┌──(kali㉿kali)-[~]
└─$ nano Phishing_email_detector.py
```

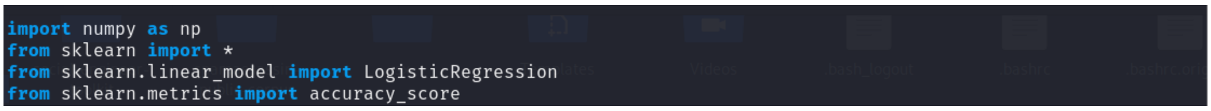
4. We will start writing our code into the editor



```
kali@kali: ~
File  Actions  Edit  View  Help
  GNU nano 7.2                        Phishing_email_detector.py
```

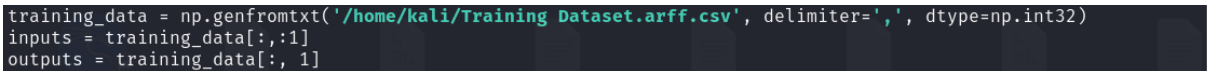
The entire code can be obtained from https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%204/Phishing_email_detector.py

5. First, **import** all the required libraries.



```python
import numpy as np
from sklearn import *
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

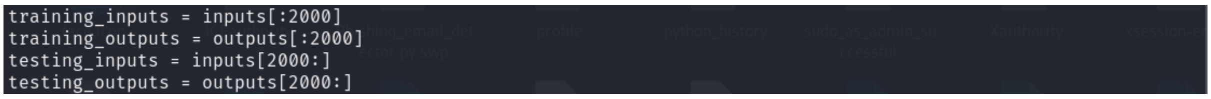
6. This script given below is used to **load the dataset**: -



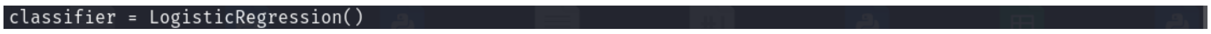
```python
training_data = np.genfromtxt('/home/kali/Training Dataset.arff.csv', delimiter=',', dtype=np.int32)
inputs = training_data[:,:1]
outputs = training_data[:, 1]
```

The name of the dataset we are using is “Dataset.arff.csv” and this is the converted csv file we saw in the feature extraction part (step 2)

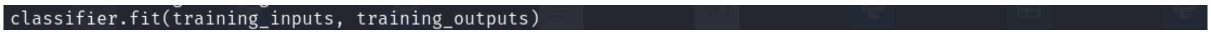
7. Now we will **split the dataset** into training and testing data.


```
training_inputs = inputs[:2000]
training_outputs = outputs[:2000]
testing_inputs = inputs[2000:]
testing_outputs = outputs[2000:]
```

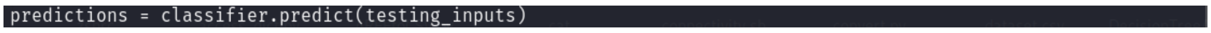
8. Now we will use the LogisticRegression() function which runs the Logistic Regression Machine Learning algorithm.


```
classifier = LogisticRegression()
```

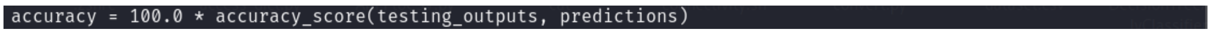
9. **Training**: As a decision tree is a supervised machine learning algorithm, the final output will be based on labelled data whose output is known based on training data. To train the model use the following function: -


```
classifier.fit(training_inputs, training_outputs)
```

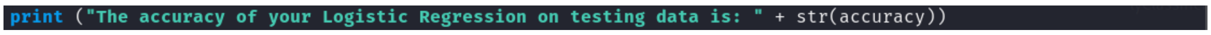
10. **Testing:** For computing the predictions, we will use the following function: -


```
predictions = classifier.predict(testing_inputs)
```

11. **Evaluation:** While checking the accuracy of this model, the result will lie between 0 & 1. To convert this value to percentage we have multiplied the result with 100.


```
accuracy = 100.0 * accuracy_score(testing_outputs, predictions)
```

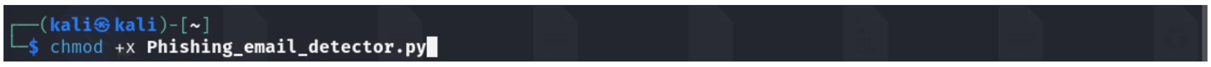
12. Finally, we print the result to display the accuracy of the Logistic Regression Model.


```
print ("The accuracy of your Logistic Regression on testing data is: " + str(accuracy))
```

13. Save the file using ctrl+x → Type in "y" and press enter.

14. Before running this Python program, we have to convert the file to an executable file. For this we will use the chmod command with +x option followed by the Python file name: -

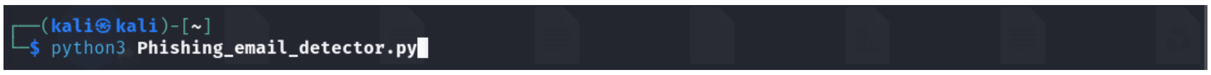
$ **chmod +x Phishing_email_detector.py**



```
┌──(kali㉿kali)-[~]
└─$ chmod +x Phishing_email_detector.py
```

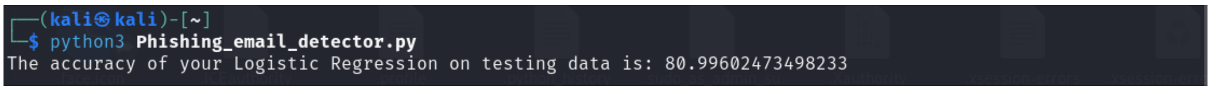
15. Run the Python program to check the accuracy of the Logistic Regression algorithm using the python3 command followed by the executable file name: -

$ **python3 Phishing_email_detector.py**



```
┌──(kali㉿kali)-[~]
└─$ python3 Phishing_email_detector.py
```

We get the result as displayed below: -

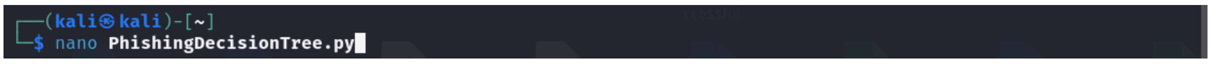


```
┌──(kali㉿kali)-[~]
└─$ python3 Phishing_email_detector.py
The accuracy of your Logistic Regression on testing data is: 80.99602473498233
```

Practical 2: Phishing Email Detection Using Decision Tree ML Algorithm

1. Open the nano editor with the desired file name: -

$ **nano PhishingDecisionTree.py**

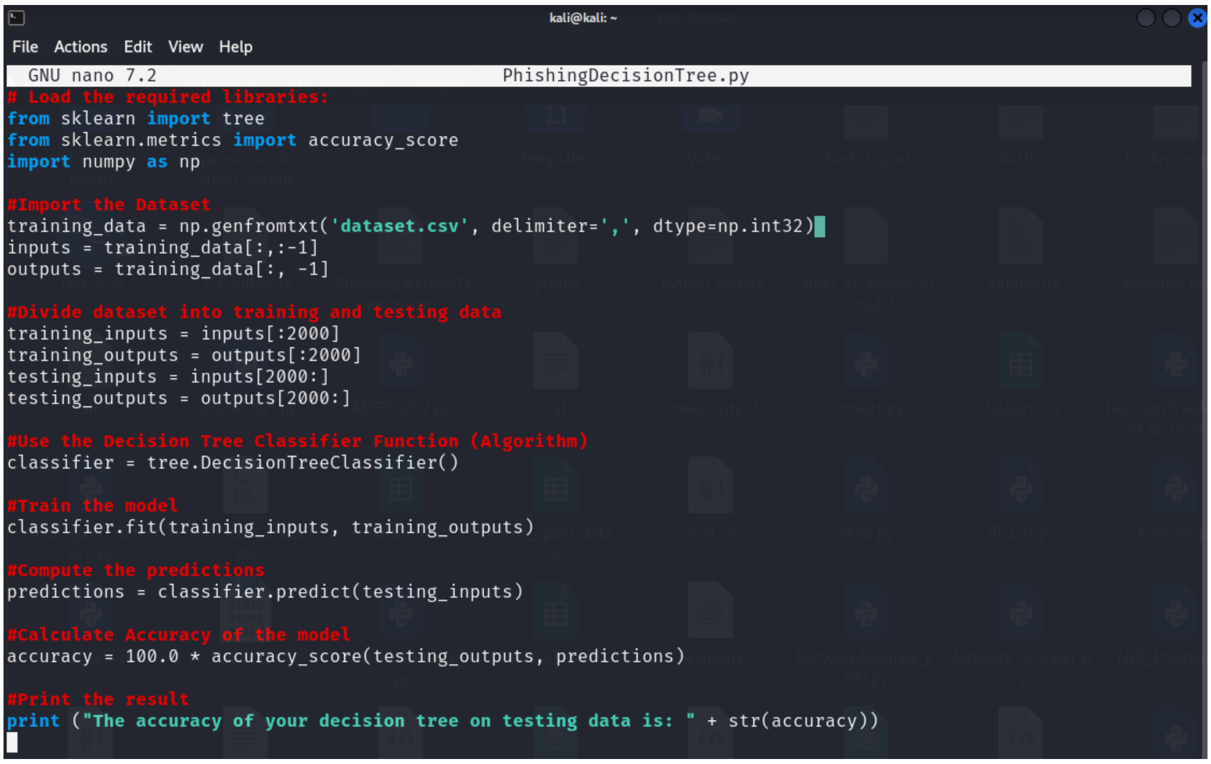


```
┌──(kali㉿kali)-[~]
└─$ nano PhishingDecisionTree.py
```

2. The dataset used here is the same as the one used above. Follow Steps 4 to 13 from the Logistic Regression walkthrough above. Just

the imported library used for Logistic Regression previously, will be different here –> **from sklearn import tree** is being used here for importing the Decision Tree function. Step number 8 will also be different because we will apply the Decision Tree function here instead of Logistic Regression. Rest of the machine learning lifecycle steps will be the same here.

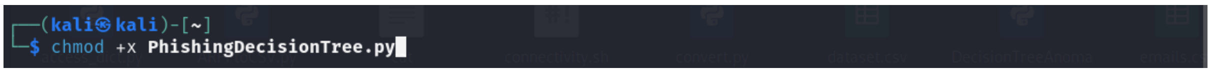
The code used can be obtained from https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%204/PhishingDecisionTree.py


kali@kali: ~
File Actions Edit View Help
GNU nano 7.2 PhishingDecisionTree.py

```
# Load the required libraries:
from sklearn import tree
from sklearn.metrics import accuracy_score
import numpy as np

#Import the Dataset
training_data = np.genfromtxt('dataset.csv', delimiter=',', dtype=np.int32)
inputs = training_data[:,:-1]
outputs = training_data[:, -1]

#Divide dataset into training and testing data
training_inputs = inputs[:2000]
training_outputs = outputs[:2000]
testing_inputs = inputs[2000:]
testing_outputs = outputs[2000:]

#Use the Decision Tree Classifier Function (Algorithm)
classifier = tree.DecisionTreeClassifier()

#Train the model
classifier.fit(training_inputs, training_outputs)

#Compute the predictions
predictions = classifier.predict(testing_inputs)

#Calculate Accuracy of the model
accuracy = 100.0 * accuracy_score(testing_outputs, predictions)

#Print the result
print ("The accuracy of your decision tree on testing data is: " + str(accuracy))
```

3. Convert the file to an executable one using the chmod +x command followed by the executable file name: -

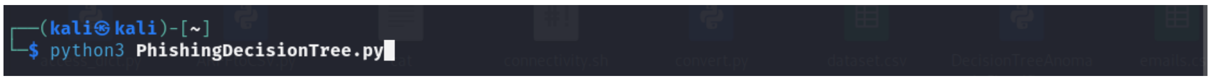
$ **chmod +x PhishingDecisionTree.py**



```
┌──(kali㉿kali)-[~]
└─$ chmod +x PhishingDecisionTree.py
```

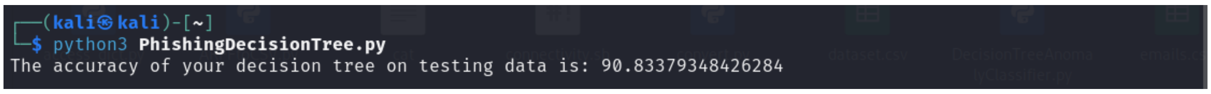
4. Run the Python program to check the accuracy of the Decision Tree algorithm using the python3 command followed by the executable file name: -

$ **python3 PhishingDecisionTree.py**



```
┌──(kali㉿kali)-[~]
└─$ python3 PhishingDecisionTree.py
```

The result obtained is displayed below: -



```
┌──(kali㉿kali)-[~]
└─$ python3 PhishingDecisionTree.py
The accuracy of your decision tree on testing data is: 90.83379348426284
```

We can observe that using the decision tree algorithm has improved our accuracy significantly. This way we can analyze which machine learning algorithm is more efficient for classifying an email into phishing and legitimate.

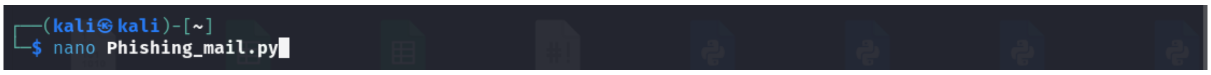
Practical 3: Phishing Email Detection Using Support Vector Machine (SVM)

1. This time we will use a different dataset for our model. This dataset consists of several messages, each marked as Spam or Ham (not spam).

This dataset is taken from Kaggle. You can download it from https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%204/spam.csv

2. Open the nano editor using the **nano** command followed by the desired file name.

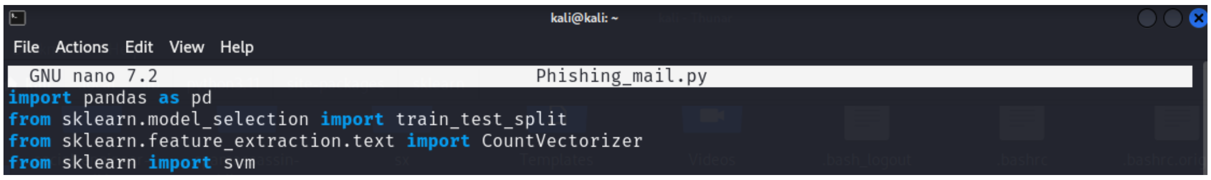
$ **nano Phishing_mail.py**



```
┌──(kali㉿kali)-[~]
└─$ nano Phishing_mail.py
```

3. We can begin entering our code. This is available at https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%204/Phishing_mail.py

4. First part is to **import** the required libraries.



```
kali@kali: ~
File  Actions  Edit  View  Help
  GNU nano 7.2                                    Phishing_mail.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
```

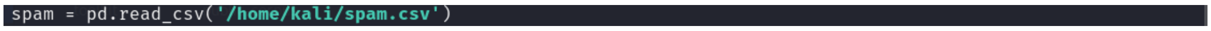
Pandas will be used for reading our dataset.

sklearn.model_selection means that model_selection is a package in sklearn library which will have functions for splitting our data and will also perform training and testing on the respective sets of data.

sklearn.feature_extraction.text will be used to perform feature extraction on the email or message text.

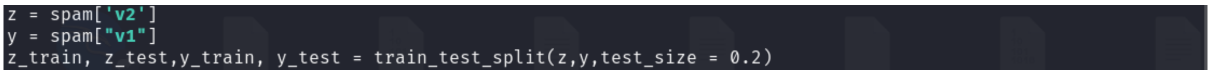
Finally, we import svm from sklearn which is the algorithm we will use for detecting spam based on the extracted features.

5. **Load** the dataset.



```
spam = pd.read_csv('/home/kali/spam.csv')
```

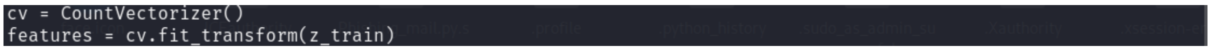
6. **Split** the dataset for training and testing.



```
z = spam['v2']
y = spam["v1"]
z_train, z_test,y_train, y_test = train_test_split(z,y,test_size = 0.2)
```

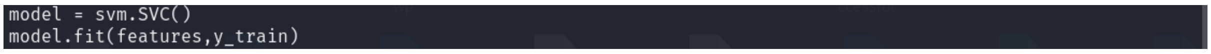
test_size = 0.2 means that 80% of the data is used for training and 20% will be used for testing.

7. Perform **feature extraction**.



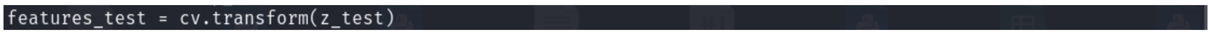
```
cv = CountVectorizer()
features = cv.fit_transform(z_train)
```

8. **Train** the model using SVM machine learning algorithm.



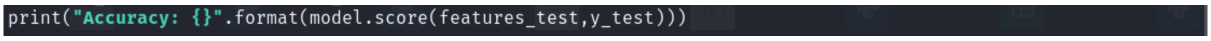
```
model = svm.SVC()
model.fit(features,y_train)
```

9. **Test** the model.



```
features_test = cv.transform(z_test)
```

10. **Check the accuracy** of the model and display it.

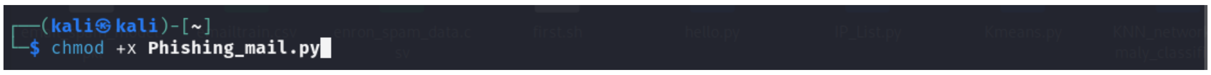


```
print("Accuracy: {}".format(model.score(features_test,y_test)))
```

11. Save the file. (“CTRL+x” followed by “y” and “Enter”)

12. Convert the file to an executable one using: -

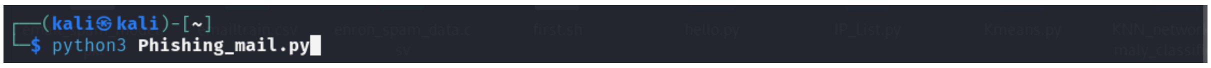
$ **chmod +x Phishing_mail.py**



```
┌──(kali㉿kali)-[~]
└─$ chmod +x Phishing_mail.py
```

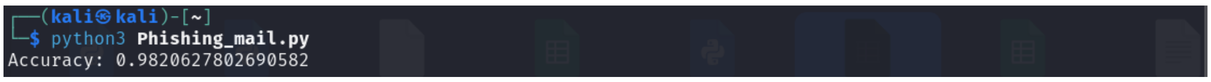
13. Execute the Python script to check the accuracy of the SVM model.

$ **python3 Phishing_mail.py**



```
┌──(kali㉿kali)-[~]
└─$ python3 Phishing_mail.py
```

14. Finally, we get the output.



```
┌──(kali㉿kali)-[~]
└─$ python3 Phishing_mail.py
Accuracy: 0.9820627802690582
```

We can see that the accuracy of our model is 98.2%

We can also modify the above code to run with Logistic Regression and Decision Tree Machine Learning algorithms and hence can check which algorithm is more accurate.

With this we have seen the crucial steps in the Machine Learning Lifecycle and how this can be used to detect spam or phishing mails/text messages.