BÁO CÁO THỰC HÀNH LAB 1

**Môn học: Phương pháp học máy trong an toàn thông tin**

**Tên chủ đề: Setting Up Your Machine Learning for Cybersecurity Arsenal**

*GVHD: Nguyễn Hữu Quyền*

1. **THÔNG TIN CHUNG:**

*(Liệt kê tất cả các thành viên trong nhóm)*

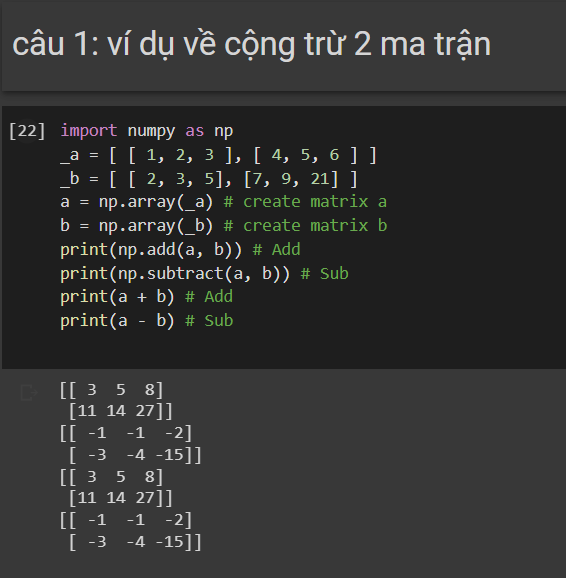
|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Bạch Văn Xuân Thông | 20521978 | 20521978@gm.uit.edu.vn |
| 2 | Nguyễn Minh Trí | 20522049 | 20522049@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-0)**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | Yêu cầu 1 | 100% |
| 2 | Yêu cầu 2 | 100% |
| 3 | Yêu cầu 3 | 100% |
| 4 | Yêu cầu 4 | 80% |
| 5 | Yêu cầu 5 | 100% |
| 6 | Yêu cầu 6 | 100% |
| 7 | Yêu cầu 7 | 100% |
| 8 | Yêu cầu 8 | 90% |

**Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.**

1. **Sinh viên cho ví vụ về phép cộng, trừ hai ma trận numpy**



1. **Sinh viên sử dụng pandas xử lý các yêu cầu sau:**

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản, đồ điện tử, ảnh chụp màn hình

Mô tả được tạo tự động

Ảnh có chứa văn bản, đồ điện tử, máy tính tay

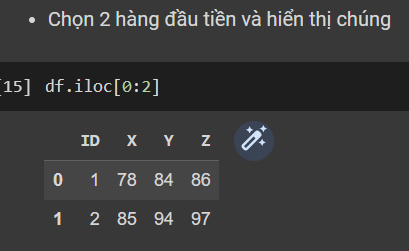
Mô tả được tạo tự động

Ảnh có chứa văn bản, đồ điện tử, máy tính tay

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động



Ảnh có chứa văn bản, màu đen, đồ điện tử, bạc

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

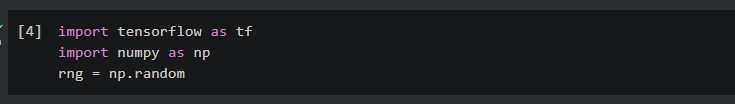
Ảnh có chứa văn bản

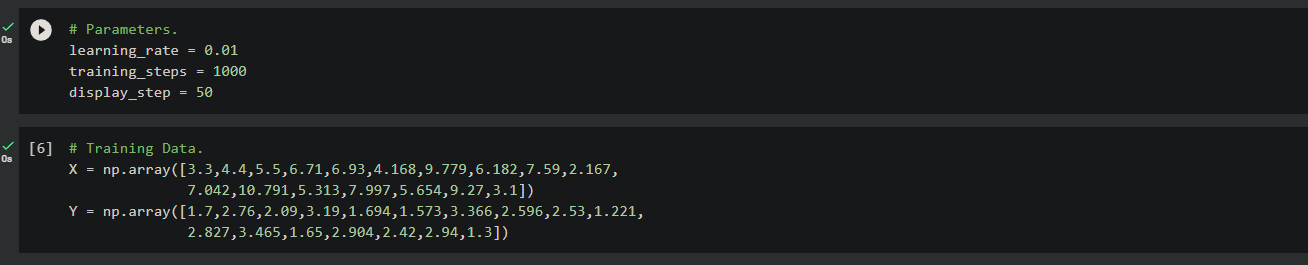
Mô tả được tạo tự động

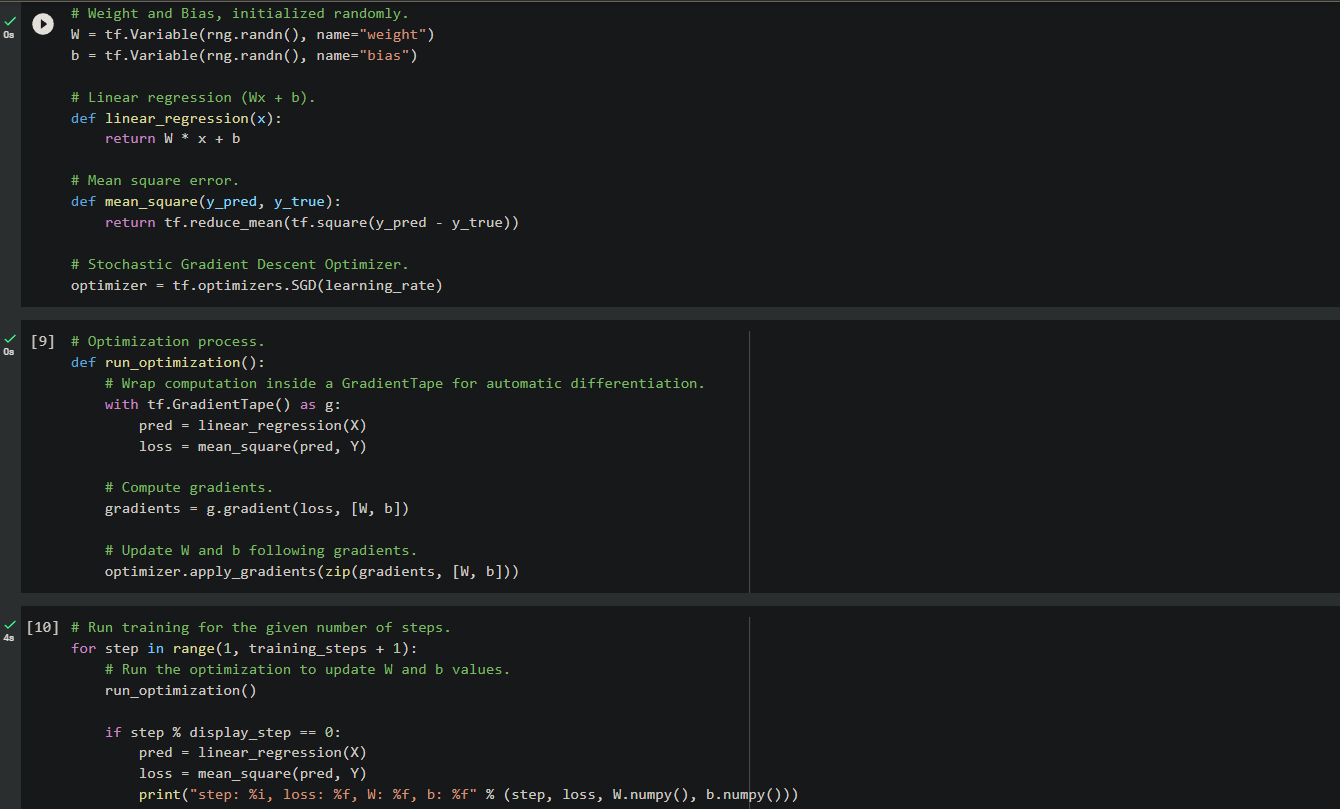
**3. Sinh viên tự tìm hiểu thực hiện lại ví dụ dùng mô hình Linear Regression trong**

**thư viện scikit-learning bằng các thư viện sau:**

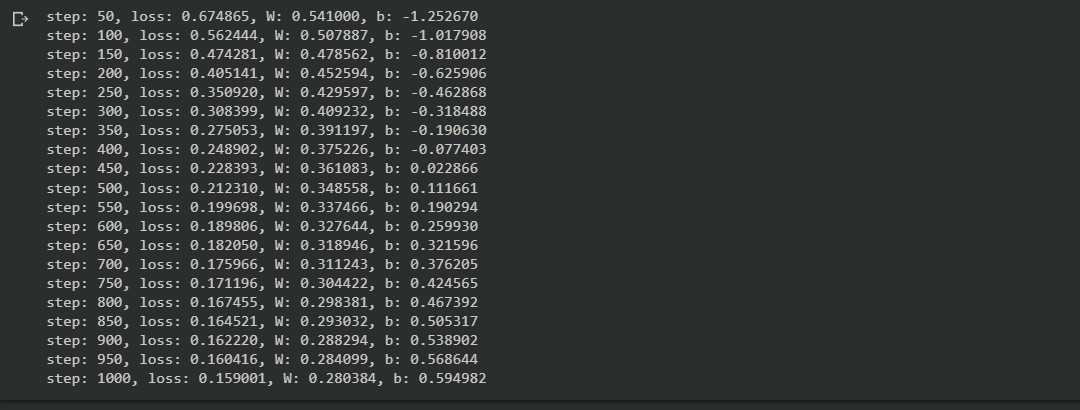
**Tensorflow:**



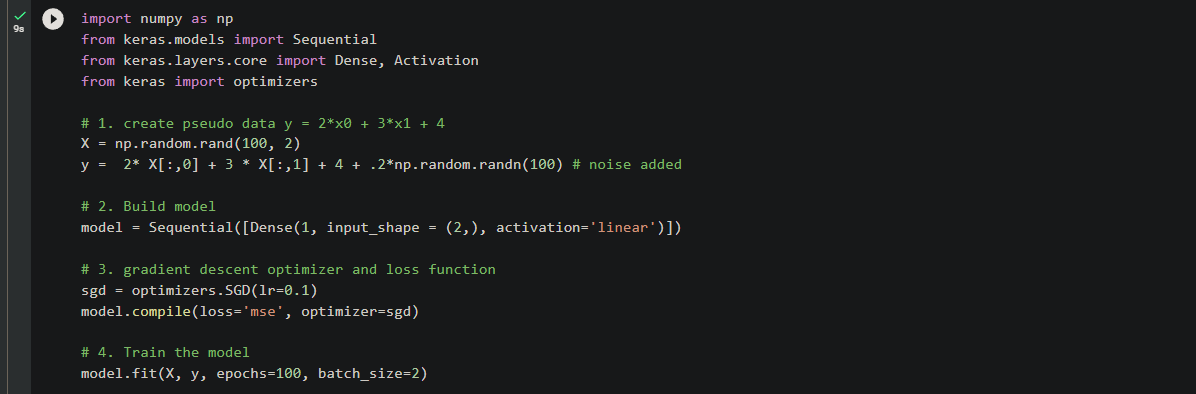




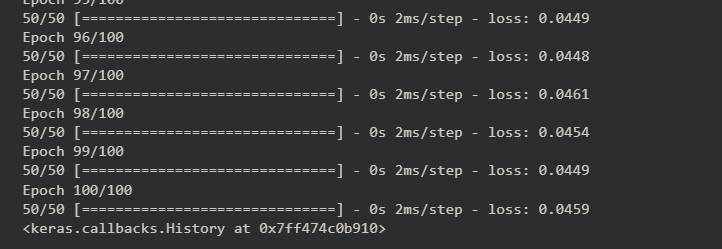
Kết quả



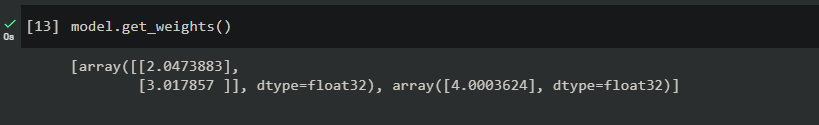
**Keras**



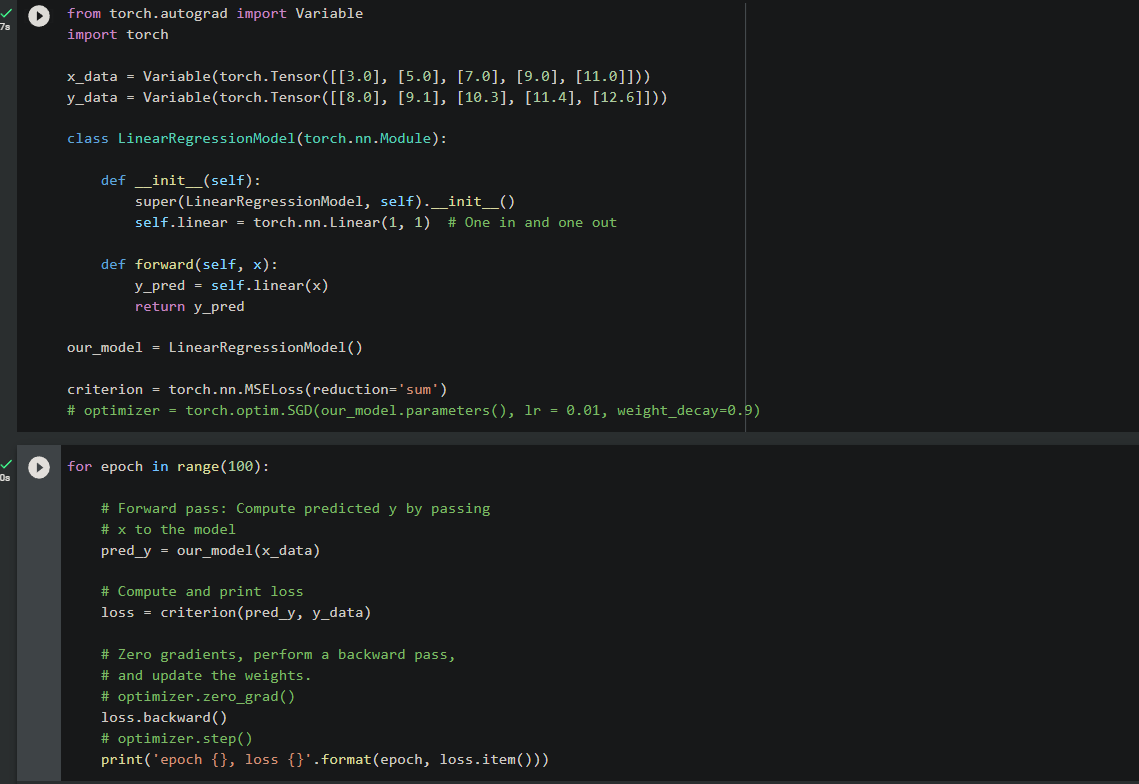
Kết quả



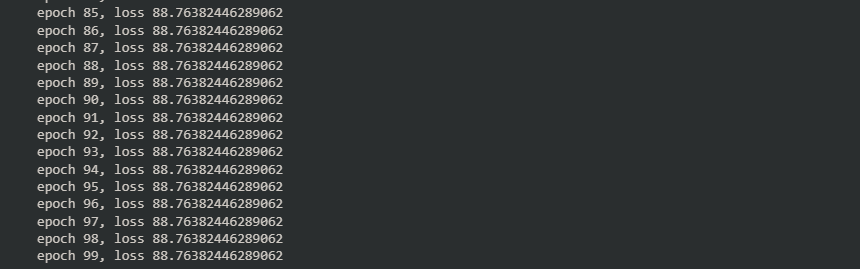
Để xem hệ số tìm được của linear regression, ta sử dụng:



**Pytorch**



Kết quả

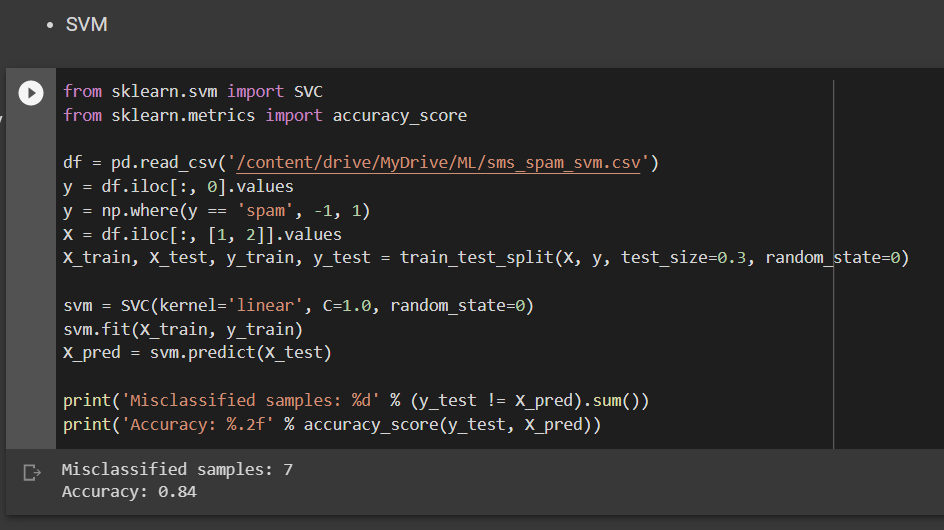


Cảm nghĩ cá nhân khi sử dụng các thư viện

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Scikit-learning | Tensorflow | Keras | Pytorch |
| Tổng quan | Một thư viện học máy nói chung, cung cấp các thuật toán cơ bản | là các đối tượng Python và các ứng dụng TensorFlow là các ứng dụng Python. Nó có nghĩa là Tensorflow là một khái niệm về trăn. | Keras là một khung học sâu cấp cao hơn, nó tóm tắt nhiều chi tiết, làm cho mã trở nên đơn giản và ngắn gọn hơn so với trong PyTorch hoặc TensorFlow, | Một thư viện hỗ trợ nhiều phương tiện liên quan cho Deep Learning |
| Tính năng đáng chú ý | Cung cấp các thuật toán chuyên về học máy Decision Tree, Logistic Regression , … etc. |  | Cung cấp các API nhất quán và đơn giản, nó giảm thiểu số lượng hành động của users cần thiết cho các trường hợp sử dụng phổ biến | Autograd - một thuật toán có thể tự động tính toán độ dốc của các hàm của bạn, được xác định theo các hoạt động cơ bản  Các quy trình tối ưu hóa dựa trên Gradient để tối ưu hóa quy mô lớn, dành riêng cho tối ưu hóa mạng thần kinh |
| Tốc độ |  | Kết thúc việc đào tạo 4.000 bước trong khoảng 15 đến 20 phút nghe có vẻ thuận tiện. | Keras chậm hơn so với Pytorch | Pytorch có tốc độ thực thi cao hơn, phù hợp cho hiệu suất cao |
| Dataset size |  | TensorFlow được sử dụng cho các bộ dữ liệu lớn và phức tạp cũng như các mô hình hiệu suất cao, đòi hỏi thực thi nhanh chóng. | Bởi lý do trên nên Keras phù hợp cho dataset nhỏ | Pytorch có thể vận hành trên dataset lớn |
| Case sử dụng | Chủ yếu khi cần sử dụng các thuật toán ***Machine Learning*** truyền thống | Phù hợp khi sử dụng Deep Learning, cả 2 thư viện đều update các mô hình Neuron Network hiện đại nhất khá tốt và nhanh.  Tùy vào từng ngữ cảnh và một số tiêu chí đánh giá để chọn lọc thư viện ([xem tại đây](https://itguru.vn/blog/pytorch-va-tensorflow-nen-chon-framework-nao/)), không có thư viện nào hoàn toàn tốt hơn | | |

**4. Sinh viên hoàn thành code phát hiện spam với SVMs và Linear regression**

**A. SVMs**



1. **Linear Regression**

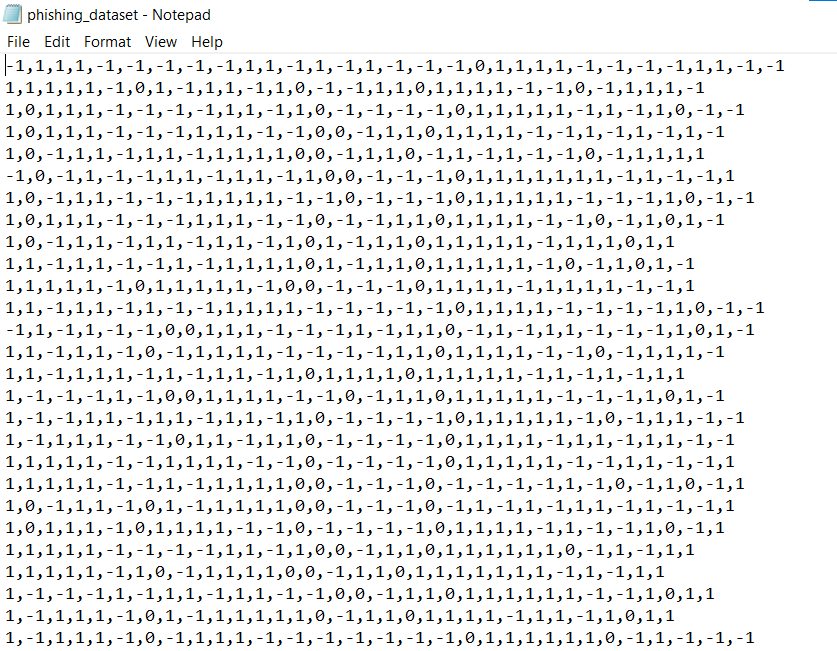
Ảnh có chứa văn bản

Mô tả được tạo tự động

1. **Sinh viên cho biết chức năng của phương thức genfromtxt() trong thư viện numpy**

Phương thức genfromtxt() được sử dụng trong câu lệnh:

phishing\_dataset = np.genfromtxt('phishing\_dataset.csv', delimiter=',', dtype=np.int32) dùng để chuyển file dataset .csv sang định dạng txt, đồng thời split dấu “**,**” để xử lý dữ liệu



1. **Sinh viên hoàn thiện code Decision trees trên và đánh giá kết quả nhận được so với phương pháp Logistic regression.**

## Logistic Regression

Ảnh có chứa văn bản

Mô tả được tạo tự động

## Decision tree

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Qua 2 hình trên, ta thấy độ chính xác của Decision Tree mang lại kết quả tốt hơn Logistic Regression.

Lý do: Theo như trên mạng, Decision Tree thích hợp với dữ liệu được gán nhãn và phân biệt chỉ với 2 giá trị “có” hoặc “không có” (trong trường hợp datasets đang sử dụng là 1 và -1), khi chạy thì cây đơn giản chỉ chọn lựa theo nhị phân 🡪 Độ chính xác cao hơn so với Logistic Regression.

Còn với Logistic Regression thì đánh giá dựa trên đồ thị nên có thể đôi lúc 1 vài datasets khác có thể cho kết quả tốt hơn, nhưng với datasets này thì kết quả thấp hơn so với Decision Tree 1 chút.

1. **Sinh viên thực hiện code phát hiện phising website bằng mô hình học máy Logistic regression và Decision trees với train và test trên tập dữ liệu**

Có thể tham khảo file notebook đính kèm

**A. Logistic regression**

import numpy as np

import pandas as pd

data = pd.read\_csv("/content/drive/MyDrive/Machine Learning/Phishing\_Legitimate\_full.csv")

#Convert data

float\_cols = data.select\_dtypes('float64').columns

for c in float\_cols:

    data[c] = data[c].astype('float32')

int\_cols = data.select\_dtypes('int64').columns

for c in int\_cols:

    data[c] = data[c].astype('int32')

data.rename(columns={'CLASS\_LABEL': 'labels'}, inplace=True)

# Phân loại thông tin để tìm mối tương qua giữa tuyến tính và phi tuyến tính của các labels và feature

from sklearn.feature\_selection import mutual\_info\_classif

X = data.drop(['id', 'labels'], axis=1)

y = data['labels']

discrete\_features = X.dtypes == int

mi\_scores = mutual\_info\_classif(X, y, discrete\_features=discrete\_features)

mi\_scores = pd.Series(mi\_scores, name='MI Scores', index=X.columns)

mi\_scores = mi\_scores.sort\_values(ascending=False)

# Dùng LogisticRegression

# Chia data thành các tập và kiểm tra

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=50)

# Tiêu chuẩn hóa data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Xác dịnh mô hình của Logistic Regression

def sigmoid(z):

    return 1 / (1 + np.exp(-z))

def loss(y\_true, y\_pred):

    return (-y\_true \* np.log(y\_pred) - (1 - y\_true) \* np.log(1 - y\_pred)).mean()

def gradient(X, y, weights):

    z = np.dot(X, weights)

    y\_pred = sigmoid(z)

    gradient = np.dot(X.T, (y\_pred - y)) / y.size

    return gradient

def train(X\_train, y\_train, X\_test, y\_test, learning\_rate, epochs):

    # Khởi tạo weights

    weights = np.zeros(X\_train.shape[1])

    # Mô hình huấn luyện sẽ như sau

    for i in range(epochs):

        # Tính grad

        grad = gradient(X\_train, y\_train, weights)

        # Cập nhật weights

        weights -= learning\_rate \* grad

        # Đánh giá mô hình trên tập huấn luyện và kiểm tra

        train\_loss = loss(y\_train, sigmoid(np.dot(X\_train, weights)))

        test\_loss = loss(y\_test, sigmoid(np.dot(X\_test, weights)))

        print(f"Epoch {i + 1}/{epochs}, train loss: {train\_loss:.4f}, test loss: {test\_loss:.4f}")

    return weights

# Train mô hình logistic regression

learning\_rate = 0.5

epochs = 50

weights = train(X\_train, y\_train, X\_test, y\_test, learning\_rate, epochs)

# Dự đoán trên tập kiểm tra

y\_pred = sigmoid(np.dot(X\_test, weights))

y\_pred\_binary = np.where(y\_pred >= 0.5, 1, 0)

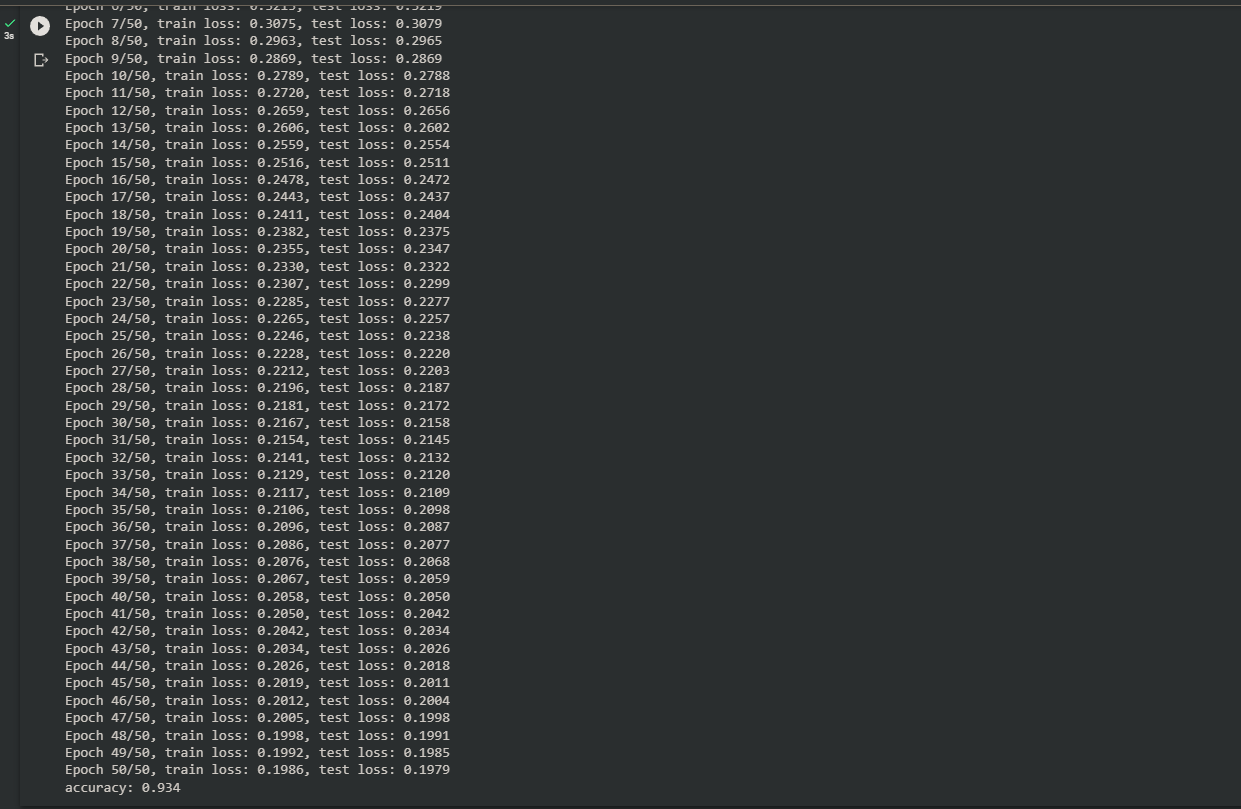
# Đánh giá sự chính xác của mô hình

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(y\_test, y\_pred\_binary)

print("accuracy:", accuracy)

Kết quả



**B. Decison trees**

import numpy as np

import pandas as pd

data = pd.read\_csv("/content/drive/MyDrive/Machine Learning/Phishing\_Legitimate\_full.csv")

# Convert data

float\_cols = data.select\_dtypes('float64').columns

for c in float\_cols:

    data[c] = data[c].astype('float32')

int\_cols = data.select\_dtypes('int64').columns

for c in int\_cols:

    data[c] = data[c].astype('int32')

data.rename(columns={'CLASS\_LABEL': 'labels'}, inplace=True)

#  Phân loại thông tin để tìm mối tương qua giữa tuyến tính và phi tuyến tính của các labels và feature

from sklearn.feature\_selection import mutual\_info\_classif

X = data.drop(['id', 'labels'], axis=1)

y = data['labels']

discrete\_features = X.dtypes == int

mi\_scores = mutual\_info\_classif(X, y, discrete\_features=discrete\_features)

mi\_scores = pd.Series(mi\_scores, name='MI Scores', index=X.columns)

mi\_scores = mi\_scores.sort\_values(ascending=False)

class Node:

    def \_\_init\_\_(self, feature\_idx=None, threshold=None, left=None, right=None, value=None):

        self.feature\_idx = feature\_idx # index của feature

        self.threshold = threshold     # Ngưỡng của việc phân chua

        self.left = left               # cây con bên phải

        self.right = right             # cây con bên trái

        self.value = value             # giá trị của node lá

class DecisionTree:

    def \_\_init\_\_(self, max\_depth=10):

        self.max\_depth = max\_depth

        self.root = None

    def fit(self, X, y):

        self.root = self.build\_tree(X, y, self.max\_depth) # xây dựng decision tree

    def predict(self, X):

      # duyệt qua cây

        return np.array([self.traverse(x, self.root) for x in X])

    def traverse(self, x, node):

      # kiểm tra nếu node hiện tại là node lá

        if node.value is not None:

      # nếu đúng thì return về class dự đoán của label

            return node.value

      # hệ số của feature

        elif x[node.feature\_idx] < node.threshold:

      # đi theo nhánh bên trái của cây

            return self.traverse(x, node.left)

        else:

      # đi theo nhánh bên phải của cây

            return self.traverse(x, node.right)

    def build\_tree(self, X, y, max\_depth, min\_samples\_leaf=1):

      # dừng nếu đạt đến độ sâu tối đa hoặc tất cả các labels đều giống nhau

        if max\_depth == 0 or len(np.unique(y)) == 1 or len(X) < min\_samples\_leaf:

            leaf\_value = np.bincount(y).argmax()

            return Node(value=leaf\_value)

      # tìm feature và ngưỡng tốt nhất để split

        feature\_idx, threshold = self.find\_best\_split(X, y)

      # dừng lại nếu không có sự phân chia nào cải thiện được thông tin

        if feature\_idx is None or threshold is None:

            leaf\_value = np.bincount(y).argmax()

            return Node(value=leaf\_value)

      # chia data dựa trên feature và ngưỡng tìm được

        left\_indices = X[:, feature\_idx] < threshold

        right\_indices = X[:, feature\_idx] >= threshold

        left\_X, left\_y = X[left\_indices], y[left\_indices]

        right\_X, right\_y = X[right\_indices], y[right\_indices]

      # Xây dựng tính đệ quy cho cây con trái và phải

        left\_subtree = self.build\_tree(left\_X, left\_y, max\_depth-1, min\_samples\_leaf)

        right\_subtree = self.build\_tree(right\_X, right\_y, max\_depth-1, min\_samples\_leaf)

      # trả về nút gốc của cây con

        return Node(feature\_idx=feature\_idx, threshold=threshold, left=left\_subtree, right=right\_subtree)

    def find\_best\_split(self, X, y):

      # tìm số lượng feature trong ma trận của feature

        n\_features = X.shape[1]

        best\_feature\_idx, best\_threshold, best\_info\_gain = None, None, -1

        parent\_entropy = self.entropy(y)

      # Lặp lại từng feature và từng ngưỡng để tìm ra sự phân chia tốt nhất

        for feature\_idx in range(n\_features):

            feature\_values = X[:, feature\_idx]

            thresholds = np.unique(feature\_values)

            for threshold in thresholds:

                left\_indices = feature\_values < threshold

                right\_indices = feature\_values >= threshold

                left\_y, right\_y = y[left\_indices], y[right\_indices]

                child\_entropy = (len(left\_y) \* self.entropy(left\_y) + len(right\_y) \* self.entropy(right\_y)) / len(y)

                info\_gain = parent\_entropy - child\_entropy

      # Cập nhật feature, ngưỡng và sự tăng lên của thông tin nếu sự phân chia hiện tại tốt hơn

                if info\_gain > best\_info\_gain:

                    best\_feature\_idx, best\_threshold, best\_info\_gain = feature\_idx, threshold, info\_gain

        return best\_feature\_idx, best\_threshold

    # Tính entropy của 1 bộ nhãn

    def entropy(self, y):

        \_, counts = np.unique(y, return\_counts=True)

        probabilities = counts / len(y)

        return -np.sum(probabilities \* np.log2(probabilities))

# Train test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X.values, y.values, test\_size=0.5, random\_state=50)

# Train decision tree

dt = DecisionTree(max\_depth=10)

dt.fit(X\_train, y\_train)

# Tạo prediction

y\_pred = dt.predict(X\_test)

# Tính accuracy

accuracy = np.sum(y\_pred == y\_test) / len(y\_test)

print("accuracy:", accuracy)

Kết quả



1. **Sinh viên thực hiện code phát hiện phising website bằng mô hình học máy Logistic regression hoặc Decision trees với train và test trên tập dữ liệu phishtank.**

Có thể tham khảo file notebook đính kèm

import os

import sys

import re

import matplotlib

import pandas as pd

import numpy as np

from os.path import splitext

import ipaddress as ip

import tldextract

import whois

import datetime

from urllib.parse import urlparse

df = pd.read\_csv("/content/drive/MyDrive/Machine Learning/verified\_online.csv")

#df=df.sample(frac=1)

df = df.sample(frac=1).reset\_index(drop=True)

df.head()

len(df)

#2016's top most suspicious TLD and words

Suspicious\_TLD=['zip','cricket','link','work','party','gq','kim','country','science','tk']

Suspicious\_Domain=['luckytime.co.kr','mattfoll.eu.interia.pl','trafficholder.com','dl.baixaki.com.br','bembed.redtube.comr','tags.expo9.exponential.com','deepspacer.com','funad.co.kr','trafficconverter.biz']

#trend micro's top malicious domains

def countdots(url):

    return url.count('.')

def countdelim(url):

    count = 0

    delim=[';','\_','?','=','&']

    for each in url:

        if each in delim:

            count = count + 1

    return count

import ipaddress as ip #works only in python 3

def isip(uri):

    try:

        if ip.ip\_address(uri):

            return 1

    except:

        return 0

def isPresentHyphen(url):

    return url.count('-')

def isPresentAt(url):

    return url.count('@')

def isPresentDSlash(url):

    return url.count('//')

def countSubDir(url):

    return url.count('/')

def get\_ext(url):

    """Return the filename extension from url, or ''."""

    root, ext = splitext(url)

    return ext

def countSubDomain(subdomain):

    if not subdomain:

        return 0

    else:

        return len(subdomain.split('.'))

def countQueries(query):

    if not query:

        return 0

    else:

        return len(query.split('&'))

featureSet = pd.DataFrame(columns=('url','no of dots','presence of hyphen','len of url','presence of at',\

'presence of double slash','no of subdir','no of subdomain','len of domain','no of queries','is IP','presence of Suspicious\_TLD',\

'presence of suspicious domain','create\_age(months)','expiry\_age(months)','update\_age(days)','country','file extension','label'))

featureSet = pd.DataFrame(columns=('url','no of dots','presence of hyphen','len of url','presence of at',\

'presence of double slash','no of subdir','no of subdomain','len of domain','no of queries','is IP','presence of Suspicious\_TLD',\

'presence of suspicious domain','label'))

from urllib.parse import urlparse

import tldextract

def getFeatures(url, label):

    result = []

    url = str(url)

    #add the url to feature set

    result.append(url)

    #parse the URL and extract the domain information

    path = urlparse(url)

    ext = tldextract.extract(url)

    #counting number of dots in subdomain

    result.append(countdots(ext.subdomain))

    #checking hyphen in domain

    result.append(isPresentHyphen(path.netloc))

    #length of URL

    result.append(len(url))

    #checking @ in the url

    result.append(isPresentAt(path.netloc))

    #checking presence of double slash

    result.append(isPresentDSlash(path.path))

    #Count number of subdir

    result.append(countSubDir(path.path))

    #number of sub domain

    result.append(countSubDomain(ext.subdomain))

    #length of domain name

    result.append(len(path.netloc))

    #count number of queries

    result.append(len(path.query))

    #Adding domain information

    #if IP address is being used as a URL

    result.append(isip(ext.domain))

    #presence of Suspicious\_TLD

    result.append(1 if ext.suffix in Suspicious\_TLD else 0)

    #presence of suspicious domain

    result.append(1 if '.'.join(ext[1:]) in Suspicious\_Domain else 0 )

    #result.append(get\_ext(path.path))

    result.append(str(label))

    return result

    #Yay! finally done!

for i in range(len(df)):

   features = getFeatures(df["url"].loc[i], df["target"].loc[i])

   featureSet.loc[i] = features

   featureSet.head()

   featureSet.groupby(featureSet['label']).size()

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

import sklearn.ensemble as ek

from sklearn.model\_selection import train\_test\_split

from sklearn import tree, linear\_model

from sklearn.feature\_selection import SelectFromModel

import sklearn.externals

import joblib

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix

from sklearn.pipeline import make\_pipeline

from sklearn import preprocessing

from sklearn import svm

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import cross\_val\_score

X = featureSet.drop(['url','label'],axis=1).values

y = featureSet['label'].values

model = { "DecisionTree":tree.DecisionTreeClassifier(max\_depth=10),

         "RandomForest":ek.RandomForestClassifier(n\_estimators=50),

         "Adaboost":ek.AdaBoostClassifier(n\_estimators=50),

         "GradientBoosting":ek.GradientBoostingClassifier(n\_estimators=50),

         "GNB":GaussianNB(),

         "LogisticRegression":LogisticRegression()

}

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X, y ,test\_size=0.2)

results = {}

for algo in model:

    clf = model[algo]

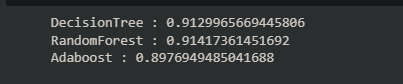
    clf.fit(X\_train,y\_train)

    score = clf.score(X\_test,y\_test)

    print ("%s : %s " %(algo, score))

    results[algo] = score

Kết quả



1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-0)