עבודת גמר – Phishing Websites Module

מאת:

אלעד ישעיהו ומתן וינמן

עבודה זו מוגשת כעבודה בהיקף של 2 יח' כמילוי חלקי של הדרישות לקראת קבלת ציון במדע חישובי

עבודה זו בוצעה בהדרכת: **אריק בן חיים**

תוכן עניינים

[פרק 1 – מבוא לרשתות 3](#_Toc135245449)

[פונקציית מחיר 3](#_Toc135245450)

[Gradient Descent 3](#_Toc135245451)

[רגרסיה לינארית עם Gradient Descent 4](#_Toc135245452)

[רגרסיה שאינה לינארית עם Gradient Descent 5](#_Toc135245453)

[פרק 2 – רשתות 7](#_Toc135245454)

[דוגמה לרשת 9](#_Toc135245455)

[דוגמה נוספת – קלסיפיקציה 10](#_Toc135245456)

[פרק 3 - המודל 13](#_Toc135245457)

[פרק 4 – סיכום 23](#_Toc135245458)

[נספחים 26](#_Toc135245459)

[המחלקה CollectData 26](#_Toc135245460)

[גרפי הנתונים והיסטוגרמות ה Attributes 32](#_Toc135245461)

[התכנית של המודל 36](#_Toc135245462)

[אתרים 39](#_Toc135245463)

# פרק 1 – מבוא לרשתות

## פונקציית מחיר

פונקציית מחיר היא פונקציה המאפשרת לבדוק את התאמת הפלט של המחשב לפלט הרצוי. היא מקבלת את הפלט הרצוי ואת הפלט המתקבל, ובדרך כלל מחזירה מספר המבטא את התאמת הפלטים. ברוב המקרים, כאשר המספר יהיה מינימלי, סימן שההתאמה היא מיטבית. כדי למצוא את נקודת המינימום, עלינו לגזור את הפונקציה, ולמצוא את הנקודה עבורה הנגזרת מתאפסת. אחת מפונקציות המחיר הנפוצות היא MSE – קיצור של Mean Squared Error.

Error – מחשבים את השגיאה / ההפרש בין הערכים, Squared – מעלים אותו בריבוע, Mean – עושים ממוצע לכל השגיאות בריבוע. הנוסחה ל-MSE עבור שני ערכים x1, x2 היא:

## Gradient Descent

ישנם מספר אלגוריתמים למציאת נקודות קיצון. אחד האלגוריתמים הנפוצים הוא אלגוריתם "Gradient Descent". האלגוריתם מוצא את השיפוע של הפונקציה בנקודה, ולפיו מחליט אם "ללכת" ימינה או שמאלה על ציר ה-X, במידה התואמת את השיפוע. כאשר האלגוריתם מקדם את הנקודה אותה הוא בודק בשיעור קטן מספיק, סימן שהשיפוע קטן מספיק והגענו לנקודת הקיצון.

הנוסחה לאלגוריתם זה היא כדלקמן:

כאשר y הוא הפונקציה בה מחפשים את נקודת הקיצון, x הוא הפרמטר שלה, ו-lr הוא אורך צעד – קיצור של Learning Rate. גודל אורך הצעד קובע את ההשפעה של השיפוע על שיעור ה-x בו ינוע האלגוריתם. Lr קטן משפר את הדיוק, אולם מאריך את זמן הריצה וצורך יותר משאבים, בעוד lr גדול מקצר את זמן הריצה וצורך פחות משאבים אבל מפחית בדיוק. יש למצוא את האיזון בין זמן הריצה הקצר ובזבוז משאבים מועט לדיוק של שיעור נקודת הקיצון.

המינוס נועד בשביל ללכת לכיוון המינימום – כלומר, אם הפונקציה עולה נלך לכיוון הירידה, ולהיפך. אם נהפוך את המינוס לפלוס – הפונקציה תמצא נקודות מקסימום (תלך "למעלה" בעלייה ו"למטה" בירידה, במקום הפוך)

## רגרסיה לינארית עם Gradient Descent

כעת, נראה את השימוש באלגוריתם לצורך רגרסיה לינארית – כלומר, מציאת שיפועו של קו ישר (לינארי), וחיתוכו עם ציר ה-y.

import numpy as np  
  
mse = lambda x: np.mean(x \*\* 2)  
x = np.linspace(0, 4, 100)  
y = 2 \* x + 3  
  
h = 10 \*\* -10; lr = 0.01  
  
cost = lambda a, b: mse(y - a \* x - b)  
dcosta = lambda a, b: (cost(a + h, b) - cost(a, b)) / h  
dcostb = lambda a, b: (cost(a, b + h) - cost(a, b)) / h  
  
a1 = -1; a2 = 10  
b1 = -1; b2 = 10  
  
while np.sqrt((a2 - a1) \*\* 2 + (b2 - b1) \*\* 2) > h:  
 a1, a2 = a2, a2 - lr \* dcosta(a2, b2)  
 b1, b2 = b2, b2 - lr \* dcostb(a2, b2)  
  
print(f'a={a2} b={b2}')

בהתחלה, ייבאנו את הספריה numpy, בה אנו נעזרים בחישובים. לאחר מכן, הגדרנו את mse להיות פונקציה המקבלת x מסוים, המהווה מערך של נתונים, ומחזירה את ממוצע סכום ריבועיהם.

X מוגדר להיות מערך של נתונים, המהווים 100 מספרים בין 0 ל-4. המשתנה y מוגדר להיות מערך נתונים אחר, המכיל את ערכיה של הפונקציה y = 2x + 3 עבור ה-x המוגדרים.

אחרי מכן, הגדרנו פרמטר שנקרא h, המהווה גודל קטן (נשתמש בו בכמה מקרים), ואת הפרמטר lr – קצב הלמידה, ה-Learning Rate של אלגוריתם ה-Gradient Descent.

הגדרנו את פונקציית המחיר, פונקציית ה-cost שלנו, להיות ה-MSE, כאשר ההפרש הוא בין ערכי הפונקציה הנתונים, הצפויים, לבין ax + b, כאשר a הוא השיפוע אותו אנו חוזים ו-b הוא החיתוך עם הציר האנכי אותו אנו חוזים. מיד אחריה, הגדרנו את נגזרותיה של פונקציית המחיר לפי a ולפי b, לפי הגדרת הנגזרת:

כאשר dx במקרה שלנו הוא h, הגודל הקטן שהגדרנו קודם לכן.

הגדרנו את הערכים a1, a2, b1, b2, המהווים את ערכי ה-a וה-b אותם אנו חוזים, וערכי ה-a וה-b הקודמים שחזינו (לצורך ההפרש).

כעת, הגענו לחלק הלוגי האמיתי בקוד. בכל איטרציה של הלולאה, אנו משנים את a ו-b לפי Gradient Descent, ומציבים את הערך הקודם שלהם בפרמטרים a1 ו-b1. הלולאה תחזור על עצמה עד ששורש סך ריבועי ההפרשים יהיה קטן או שווה ל-h – הגודל הקטן שהגדרנו. כלומר הלולאה תחזור על עצמה עד שבקושי תהיה התקדמות, הן מבחינת השיפוע והן מבחינת החיתוך עם הציר.

בסוף הקוד, אנו מדפיסים את הפלט שהתקבל (השיפוע והחיתוך). הערכים שהתקבלו הם:

> a=1.999999991853208 b=3.0000000207993653

מה שמראה שהקוד עובד כמו שצריך

## רגרסיה שאינה לינארית עם Gradient Descent

כעת, נרצה להיות מסוגלים למצוא פרמטרים של פונקציות אחרות, גם כאלו שאינן לינאריות. במקרים אלו, נשתמש באותו קוד, אולם בשינוי קטן- נוסיף לפונקציית ה-cost את הפרמטרים הנוספים (בין אם חזקות של פרמטרים קיימים או פרמטרים חדשים לחלוטים), וניתן למחשב לעשות את שלו:

import numpy as np  
  
mse = lambda x: np.mean(x \*\* 2)  
x = np.linspace(0, 4, 100)  
y = 5 \* x \*\* 2 + 2 \* x + 3  
  
h = 10 \*\* -10; lr = 0.01  
  
cost = lambda a, b, c: mse(y - a \* x - b - c \* x \*\* 2)  
dcosta = lambda a, b, c: (cost(a + h, b, c) - cost(a, b, c)) / h  
dcostb = lambda a, b, c: (cost(a, b + h, c) - cost(a, b, c)) / h  
dcostc = lambda a, b, c: (cost(a, b, c + h) - cost(a, b, c)) / h  
  
a1 = -1; a2 = 10  
b1 = -1; b2 = 10  
c1 = -1; c2 = 10  
  
while np.sqrt((a2 - a1) \*\* 2 + (b2 - b1) \*\* 2 + (c2 - c1) \*\* 2) > h:  
 a1, a2 = a2, a2 - lr \* dcosta(a2, b2, c2)  
 b1, b2 = b2, b2 - lr \* dcostb(a2, b2, c2)  
 c1, c2 = c2, c2 - lr \* dcostc(a2, b2, c2)  
  
print(f'a={a2} b={b2} c={c2}')

במקרה זה, הוספנו לפונקציה ארגומנט נוסף – 5x2, ולכן הוספנו לקוד תמיכה בנגזרת לפי c (המקדם של x2), כלילה של c בפונקציית המחיר וכלילה של c בתנאי לעצירת הלולאה.

והתוצאות:

> a=1.9999999350430602 b=3.0000000556549984 c=5.000000014407462

כלומר הקוד עובד.

# פרק 2 – רשתות

תמונה שמכילה תרשים

התיאור נוצר באופן אוטומטירשת נוירונים היא סוג של רשת מלאכותית, שתפקידה להפיק פלט רצוי כתגובה לקלט מסוים, על סמך דוגמאות שניתנו בעבר. הרשת בנויה משלוש שכבות או יותר – שכבת הקלט (Input Layer), שכבת הפלט (Output Layer) וכל מה שקורה ביניהן – השכבות הנסתרות (Hidden Layers). הרשת עצמה מורכבת מנוירונים, שהם יחידות קטנות, שתפקידן לקבל נתונים ולהפעיל עליהם פעולה מסוימת. לכל שכבה מגיע הפלט של השכבה שלפניה, והיא מעבירה את הפלט שלה לשכבה שאחריה. כך, עובר מידע ברשת, תוך שמופעלות עליו פעולות שונות, ובסוף התהליך מתקבל פלט מהשכבה האחרונה. להלן שרטוט סכמטי של רשת נוירונים:

כפי שניתן לראות, הרשת מקבלת מידע לאוסף מסוים של נוירונים, וכל נוירון ברשת מקושר לכל אחד מהנוירונים בשכבה שלפניו ובשכבה שאחריו. המידע היוצא מנוירון עשוי לעבור שינוי נוסף – לעבור דרך פונקציה מסוימת הנקראת "פונקציית אקטיבציה". פונקציית אקטיבציה היא פונקציה ממאגר מוגדר של פונקציות, למשל tanh, adam או softmax, שנועדה לערוך את תוצאת החישוב כך שיהיה יותר נוח לעבוד עימו – למשל, להגביל את התחומים האפשריים של הערכים המספריים כך שיהיו בין 0 ל-1, או בין 1- ל-1. למשל, כך נראית פונקציית tanh (טנגנס היפרבולי):

תמונה שמכילה צבעוני, חשיכה

התיאור נוצר באופן אוטומטי

תמונה שמכילה תרשים, צילום מסך, עיגול, קו

התיאור נוצר באופן אוטומטיבתמונה הבאה, ניתן לראות דוגמה למבנה של נוירון, שתפקידו לחשב ממוצע משוקלל, ולאחר מכן מפעיל על התוצאה פונקציית אקטיבציה:

ניתן לראות כי הנוירון מקבל כקלט n ערכים ואת המשקלים של כל אחד, מחשב את הממוצע המשוקלל, ולאחר מכן מפעיל על התוצאה פונקציה מסוימת. כך פועל כל נוירון – כאשר החישוב אינו בהכרח ממוצע משוקלל, וכמו כן פונקציית האקטיבציה תלויה בהרכב הרשת, ויכולה אף שלא להופיע כלל.

למידת המכונה עצמה מתבצעת עם מאגר נתונים לאימון המודל - **training**. כל נוירון מתחיל עם פעולות שרירותיות, והפלט המתקבל מהרשת מושווה עם התוצאה הרצויה. אם הם לא מתאימים, הפעולות שמפעיל כל נוירון ישתנו מעט, והרשת תרוץ שוב עם אותם הנתונים, ותשווה את הפלט החדש עם הפלט הרצוי. התהליך חוזר על עצמו עד שהפלט המתקבל תואם את מה שאמור לצאת, וכך הרשת מסיימת את שלב האימון שלה - ומוכנה לנבא על סמך האימון הזה פלט לכל קלט אותו תקבל.

## דוגמה לרשת

על מנת להדגים את השימוש ברשתות נוירונים, כתבנו רשת בעצמנו – המבצעת רגרסיה לינארית, באמצעות ספריית Sklearn ב-python. בספריה, יש מחלקה שנקראת LinearRegression – המתעלת רשת נוירונים לצורך מציאת מקדמיו של קו ישר. להלן הקוד:

import numpy as np  
from sklearn import linear\_model as lm  
from sklearn.model\_selection import train\_test\_split as split  
  
x = np.linspace(-10, 10, 500)y = 5 \* x\*\*3 - 2 \* x\*\*2 -1 \* x + 5  
x\_train, x\_test, y\_train, y\_test = split(np.asarray([x, x\*\*2, x\*\*3, x\*\*4]).T, y, test\_size=0.2)  
model = lm.LinearRegression()  
model.fit(x\_train, y\_train)  
print(f'y = {np.round(model.coef\_[3])} \* x^4 + {np.round(model.coef\_[2])} \* x^3 + {np.round(model.coef\_[1])} \* x^2 + {np.round(model.coef\_[0])} \* x + {np.round(model.intercept\_)}')  
  
mse = np.mean((model.predict(x\_test) - y\_test)\*\*2)  
print("Mean squared error:", mse)

כפי שניתן לראות, בהתחלה ייבאנו את כל מה שצריך מהספרייה sklearn, וגם את הספריה numpy – המסייעת בעבודה עם מספרים בשפת python. בשורות הראשונות של הקוד, הגדרנו את x ו-y – שהם קלטים הניתנים למודל, כאשר מטרתו של המודל לזהות את המקדמים של החזקות השונות של x. כפי שניתן לראות, הפונקציה שהגדרנו היא . אמנם השתמשנו במודל המשמש למציאת מקדמי קו לינארי, אולם נתנו לו כפרמטרים את החזקות השונות של x – כלומר, הוא מצא את המקדמים של הקו הלינארי שתלוי ב-x, x2 ו-x3. בנוסף, נתנו לו כפרמטר נוסף את x4 – כדי לראות האם יצליח לזהות שהפונקציה איננה ממעלה רביעית.

הפעולה train\_test\_split, אותה ייבאנו בשם split ויישמנו לאחר מכן, מחלקת מאגר של נתונים לנתונים המיועדים לאימון ונתונים המיועדים לבדיקה. הפרמטר test\_size שנתנו לפעולה מציין את החלק מהנתונים שישמש לבדיקת הרשת – כלומר, מכיוון שכתבנו 0.2, הקצנו 20% מהנתונים שברשותנו לצורך בדיקה, ואת השאר לאימון של הרשת. לאחר מכן, השתמשנו בפקודה model.fit – שמטרתה להתאים את המודל הלינארי לפרמטרים שנתנו למודל, על פי הקלטים והפלטים שנתנו לפעולה כפרמטרים.

כאשר סיימנו לאמן את המודל, רצינו לראות האם הוא אכן הצליח לחזות את המקדמים של הפונקציה שנתנו לו. ואכן, הפלט שהתקבל היה:

> y = -0.0 \* x^4 + 5.0 \* x^3 + -2.0 \* x^2 + -1.0 \* x + 5.0

כלומר, המודל אכן הצליח לחזות את המקדמים הנכונים. ניתן לראות שהוא עבד גם על ידי בדיקת ערכה של פונקציית ה-MSE, שהוא:

Mean squared error: 1.175602348199648e-22

כלומר מספר מאוד קטן. לכן, ניתן לראות שהמודל אכן הצליח לענות על הבעיה.

## דוגמה נוספת – קלסיפיקציה

נוסף על הרשת הכתובה לעיל, כתבנו גם רשת נוירונים המשמשת לקלסיפיקציה – כלומר חלוקה של נתונים לקבוצות, על סמך מידע קודם. לצורך הדוגמה, התבססנו על מאגר נתונים המשמש לחקירת סרטן השד – Breast Cancer:

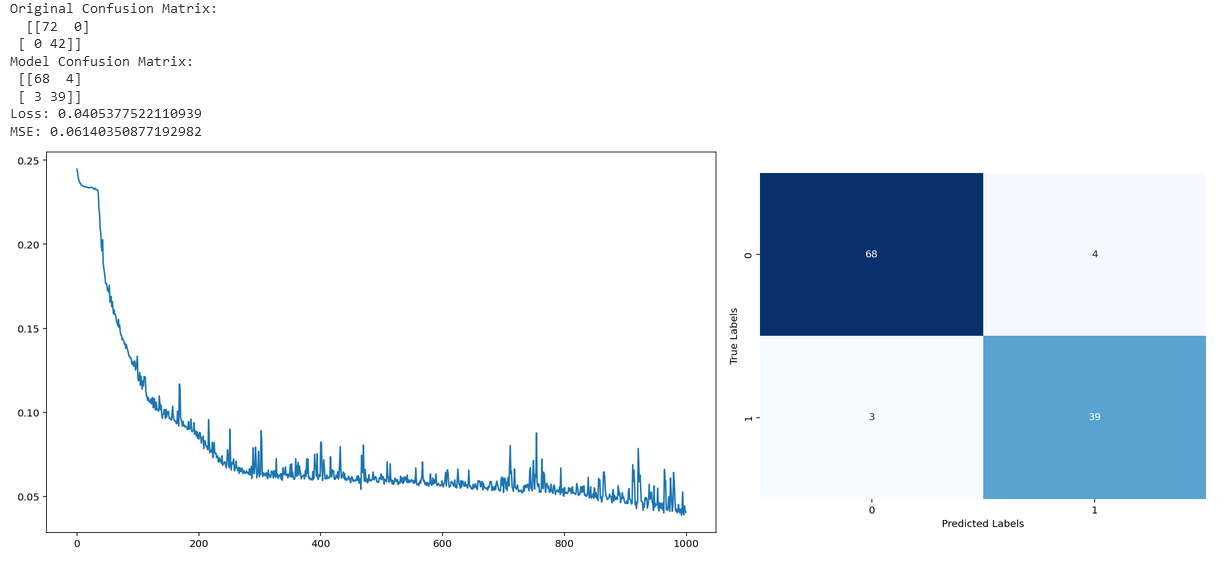
import numpy as np  
import pandas as pd  
import seaborn as sns  
import tensorflow as tf  
from sklearn.datasets import load\_breast\_cancer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import mean\_squared\_error as mse  
import matplotlib.pyplot as plt  
  
def plot\_confusion\_matrix(cm):  
 plt.figure(figsize=(8, 6))  
 sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)  
 plt.xlabel('Predicted Labels')  
 plt.ylabel('True Labels')  
 plt.show()  
  
class\_names = ['Class 0', 'Class 1']  
  
data = load\_breast\_cancer()  
X = np.array(pd.DataFrame(data.data, columns=data.feature\_names))  
Y = np.array(pd.Series(data.target))

y1 = Y \* 0  
y1[np.argwhere(Y == 1)] = 1  
y2 = Y \* 0  
y2[np.argwhere(Y == 0)] = 1  
y\_new = np.array([y1, y2]).T  
  
state = np.array([0, 1])  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y\_new, test\_size=0.2, random\_state=1)  
  
model = tf.keras.Sequential([  
 tf.keras.layers.Dense(units=2, activation='sigmoid'),  
 tf.keras.layers.Dense(units=2, activation='softmax')  
])  
  
model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['accuracy'])  
history = model.fit(x\_train, y\_train, epochs=1000, verbose=0)  
y\_pred = np.round(model.predict(x\_test)).astype(int)  
  
cm = confusion\_matrix(np.dot(y\_test, state), np.dot(y\_pred, state))  
print('Original Confusion Matrix:\n ', confusion\_matrix(np.dot(y\_test, state), np.dot(y\_test, state)))  
print('Model Confusion Matrix:\n', cm)  
  
print(f'Loss: {history.history["loss"][-1]}')  
loss = history.history['loss']  
print('MSE:', mse(y\_test, y\_pred))  
  
plt.plot(loss)  
plot\_confusion\_matrix(cm)

בתחילת הקוד, לאחר שייבאנו את הספריות המתאימות, ייבאנו את המידע של המודל באמצעות הפונקציה load\_breast\_cancer(). לאחר מכן, הגדרנו X ו-Y, שהם המשתנים עליהם נתבסס באימון המודל, ואחריהם מטריצה שנקראת y\_new – המכילה שני טורים. אם מדובר בגידול שפיר / Class 1 , הטור הימני יכיל 1, ואם מדובר בגידול מזיק / Class 0, הטור השמאלי יכיל 1. השתמשנו במטריצה כזו על מנת שיהיה נוח יותר ליצור confusion matrix, המציגה את תוצאות האימון בצורה נוחה יותר. לאחר שיצרנו את y\_new ואת X, חילקנו את הנתונים למידע המיועד לאימון ולמידע המיועד לבדיקה, כאשר כמות הנתונים לבדיקה מהווה 20% מכמות הנתונים הכוללת.

אחרי כן, הגיע החלק המשמעותי בקוד – יצירת הרשת עצמה. כפי שניתן לראות, הרשת מיוצגת על ידי עצם מסוג Sequential. כפרמטר, נתנו לו רשימה עם שני עצמים מסוג Dense – כאשר כל עצם כזה מהווה שכבת נוירונים. בכל שכבה, שמנו 2 נוירונים, כאשר לאחד הגדרנו את פונקציית האקטיבציה sigmoid, ולשני את פונקציית האקטיבציה softmax. בתום ההגדרה של הרשת, בנינו אותה באמצעות compile, ואימנו אותה באמצעות הפקודה model.fit. הפרמטר epochs מייצג את מספר החזרות של הרשת על פעולת האימון, עד למתן התוצאה הסופית. לאחר האימון, מצאנו את ערכי ה-y החזויים עבור ערכי ה-x\_test, על מנת להשוות אותם עם ערכי ה-y האמיתיים ולבדוק האם הרשת אכן עבדה כמו שצריך.

לצורך כך, יצרנו confusion matrix – "מטריצת בלבול", שנועדה להראות בצורה נוחה יותר איפה המודל טעה ואיפה הוא צדק. ה-Original Confusion Matrix שהדפסנו הוא ה-Confusion Matrix עבור ערכי ה-y המקוריים (כלומר מה אמור להיות כשהכל נכון), וה-Matrix השני הוא ה-Confusion Matrix עבור ערכי ה-y אותם מנבא המודל. כמו כן, הדפסנו את ה-Loss (אחוז השגיאה של המודל), את ה-MSE של המודל, ושרטטנו את ה-Confusion Matrix ואת ה-Loss של המודל כתלות במשך זמן האימון.

הפלט שקיבלנו נראה כך:

ממבט ראשון, ניתן לראות כי המודל הגיע לארבעה אחוזי שגיאה ול-MSE של 0.06 – כלומר דיוק די גבוה. ניתן לראות כי לאורך הזמן, ה-LOSS ירד באופן מונוטוני, עד שהגיע לנקודה נמוכה בה סיים את האימון. מה-Confusion Matrix ניתן ללמוד כי המודל חזה 68 אפסים להיות אפס, 39 אחדות להיות אחד, וטעה רק ב-7 תחזיות – הוא חזה 4 אפסים להיות אחד, ו-3 אחדות להיות אפס. כלומר הרשת שלנו הגיעה לתוצאות טובות מאוד.

# פרק 3 - המודל

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטיתמונה שמכילה מחשב, לבוש, טקסט, הנעלה

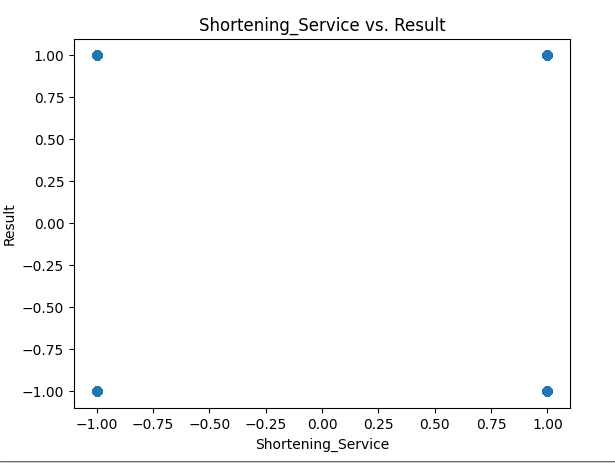
התיאור נוצר באופן אוטומטיהמודל שבחרנו לאמן לצורך המטלה הוא מודל המזהה אם אתר הוא אתר פישינג / זדוני, כלומר אחד המשמש לתקיפת משתמשים ברשת, או אם הוא אתר ידידותי (רגיל). המודל מקבל מספר מאפיינים של האתר – למשל, האם כתובתו היא כתובת IP או כתובת דומיין, האם הדומיין שלו מנוסח בצורה חשודה, האם האתר מכיל הפניות לקבצים שלא משויכים לכתובת בה הוא נמצא ועוד. למודל 30 Attributes, אולם בעבודה על הפרויקט גילינו כי אנחנו יכולים למצוא על אתר נתון רק 21 מהם, באמצעות הכלים המוכרים לנו.

ה-Attributes בהם השתמשנו הם:

1. האם כתובה בשדה הכתובת של האתר כתובת IP (4 מספרים המופרדים בנקודות) או כתובת דומיין, כלומר טקסט? אם מדובר בכתובת IP, כנראה שהאתר זדוני.
2. אורך כתובת הדומיין – ככל שהכתובת ארוכה יותר, כך סביר יותר שהאתר זדוני.
3. האם כתובת הדומיין מקוצרת באמצעות TinyURL – אם כן סביר שהאתר זדוני, מפני שקיצור הכתובת מסווה את המקור שלה.
4. האם התו '@' מופיע בכתובת האתר – אם כן, הדפדפן יתעלם מכל מה שכתוב אחריו, וסביר שאתר שמשתמש בזה יהיה זדוני.
5. האם התו '//' מופיע בכתובת – מפני שהוא מציין הפנייה מחדש לאתר אחר, ואתר שמשתמש בזה עשוי להיות זדוני.
6. האם התו '-' מופיע בכתובת האתר – מפני שהוא מבלבל עם התו '.', ואתר שמשתמש בזה עשוי להיות זדוני.
7. כמות הנקודות המופיעות בכתובת האתר – ככל שיש יותר מהן, סביר יותר שהאתר זדוני.
8. האם תעודת האמינות של האתר (ה-Certificate שלו) מוקצה על ידי מקור אמין, והאם היא מוקצה לזמן ארוך (עברה יותר משנה מאז הקצאתה). אתר שלא עונה על הקריטריונים האלה עשוי להיות זדוני.
9. האם הזמן שייקח עד שתוקף התעודה הנ"ל של האתר תפוג ארוך. אם לא, ייתכן והאתר זדוני.
10. האם סמל האתר (המופיע ליד כותרתו בכרטיסייה בדפדפן) לקוח מכתובת אחרת שאינה משויכת לאתר זה. אם כן, סביר שהאתר זדוני.
11. האם האתר פועל באמצעות פרוטוקול https (המאובטח), או באמצעות http. אם הוא פועל ב-http, סביר שהוא זדוני.
12. האם יש באתר תמונות וסרטונים הלקוחים מאתר אחר. אם כן, האתר עשוי להיות זדוני.
13. האם יש באתר לינקים המפנים לכתובת אחרת. אם כן, האתר עשוי להיות זדוני.
14. האם המידע השמור על האתר, למשל בתגי meta (המכילים metadata, ומשמשים למשל לפרסום האתר בגוגל) נלקח מכתובת אחרת. אם כן, האתר עשוי להיות זדוני.
15. האם כמות ההפניות מחדש שבוצעו עד שהאתר הסופי נטען גדולה. כלומר, כמות הפעמים שנשלחה בקשת תוכן לאתר מסוים וחזרה תשובה כגון "פנה לאתר אחר בכתובת '...'". אם כמות זו גדולה, סביר והאתר הוא אתר זדוני.
16. האם מתבצעת פעולה כאשר המשתמש באתר מקרב את העכבר שלו לשדה הכתובת של האתר. אם כן, ייתכן והאתר זדוני.
17. האם האתר מבטל או משנה את הפעולה של לחיצה ימנית על העכבר – אם כן, הוא עשוי להיות זדוני.
18. האם באתר מופיעות הודעות קופצות ("Pop-Up") המכילות שדות טקסט. שדות טקסט אלו אינם מאובטחים, ולכן אם הם קיימים האתר עשוי להיות זדוני.
19. האם האתר מכיל עצמים מסוג iframe, המציגים תוכן של אתר אחר או קובץ נפרד. אם כן, ייתכן והוא זדוני.
20. זמן היצירה של כתובת הדומיין. אם הוא קצר, כנראה שהאתר זדוני.
21. אם אין שרתי DNS (המתרגמים כתובות דומיין לכתובות IP) המכילים מידע על כתובת האתר, כנראה שהוא זדוני.

על מנת לראות האם ישר קשר בין ה Attributes השונים לבין ה Result, שרטטנו גרפים של ה Result כתלות בכל Attribute. להלן שני גרפים מהתוצאות:

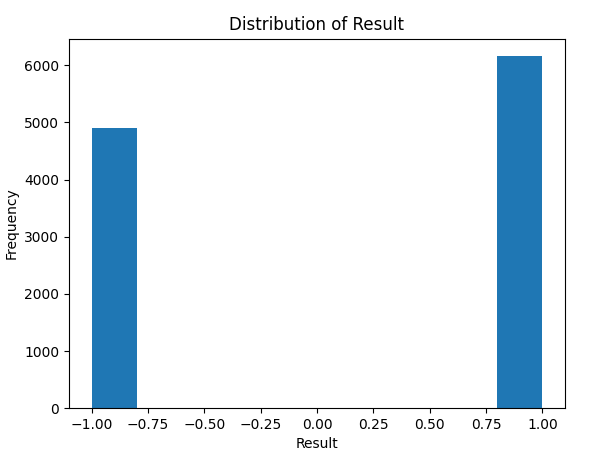
תמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי

ניתן לראות שבשני הגרפים (האחד מימין של Shortening\_Service-ה Attribute השלישי, והשני משמאל של having\_Sub\_Domain- ה Attribute השביעי) בציר הx ישנו הAttribute ובציר ה y ישנו הResult . אפשר להבחין כי למרות ששרטטנו את הגרף עם פעולת ה Scatter, כלומר פיזור של הנקודות, לא מצאנו איזו הבחנה או אינדיקציה מיוחדת לגבי הנתונים. לכן אנחנו מניחים שאי אפשר להסיק מידע מועיל מהגרפים.

שלב הבא, חילצנו מהנתונים היסטוגרמות שחשבנו כי הן עלולות להיות מעניינות. בחרנו לשרטט היסטוגרמה ל Result ול URL\_Length – הAttribute השני ברשימה:

תמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי

בציר ה x ישנו שם ה Attribute שאותו אנחנו בוחנים, ובציר ה y את הכמות של כל קטגוריה (תמים – שנמצא כמספר 1-, חשוד – שנמצא כמספר 0, וזדוני – שנמצא כמספר 1. בנוגע להיסטוגרמה הראשונה (URL\_Length), ראינו כי יש כמות מספקת משלושת הקטגוריות משלושת האתרים, גם יש יותר אתרים שהם לא חשודים בכלל. לכן אנו יכולים להסיק כי המודל ינבא תוצאות מוצלחות. בנוגע להיסטוגרמה השנייה (Result), אנו רואים כי הנתונים מתפלגים בצורה די שווה בין האתרים הזדוניים, לאתרים התמימים, מה שמוסיף להשערה כי התוצאות שהמודל יניב יהיו קרובות למציאות.

בחזרה לכתיבה המודל, את המידע לגבי כל Attribute אספנו באמצעות ספריות שונות בשפת python, כאשר מה שניתן לקוד הוא כתובת האתר, ומטרתו להשיג את כל הנתונים הללו על ידי בדיקות שונות ושליחת הודעות שונות לכתובת הנ"ל.

התחלנו מלכתוב את הקוד המחלץ את כל המידע מכתובת הדומיין הניתנת לו, אותו ריכזנו בספרייה CollectData (הקוד המלא נמצא בנספחים). לאחר מכן, ניגשנו לכתוב את המודל עצמו. בהתחלה, רצינו לכתוב מודל המניב תוצאה כלשהי, בין אם היא טובה או לא:

import random  
import numpy as np  
import pandas as pd  
import seaborn as sns  
import tensorflow.keras as tf  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix, mean\_squared\_error as mse  
from Data import CollectData  
  
URL = 'https://www.nike.com'

def plot\_confusion\_matrix(cm):plt.figure(figsize=(8, 6))  
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)plt.xlabel('Predicted Labels')  
 plt.ylabel('True Labels')plt.show()

df = pd.read\_csv('Training\_Dataset.csv')  
print('Got CSV')

df = pd.DataFrame((df['having\_IP\_Address'], df['URL\_Length'], df['Shortening\_Service'],  
 df['having\_At\_Symbol'], df['double\_slash\_redirecting'], df['Prefix\_Suffix'],  
 df['having\_Sub\_Domain'], df['SSLfinal\_State'], df['Domain\_registration\_length'],  
 df['Favicon'], df['HTTPS\_token'], df['Request\_URL'], df['URL\_of\_Anchor'], df['Links\_in\_tags'],  
 df['Redirect'], df['on\_mouseover'], df['RightClick'], df['popUpWindow'], df['Iframe'],  
 df['age\_of\_domain'], df['DNSRecord'], df['Result'])).T

x = np.array(df.iloc[:, :-1]) *# data*y = np.array(df.iloc[:, -1]) *# result*  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=1)

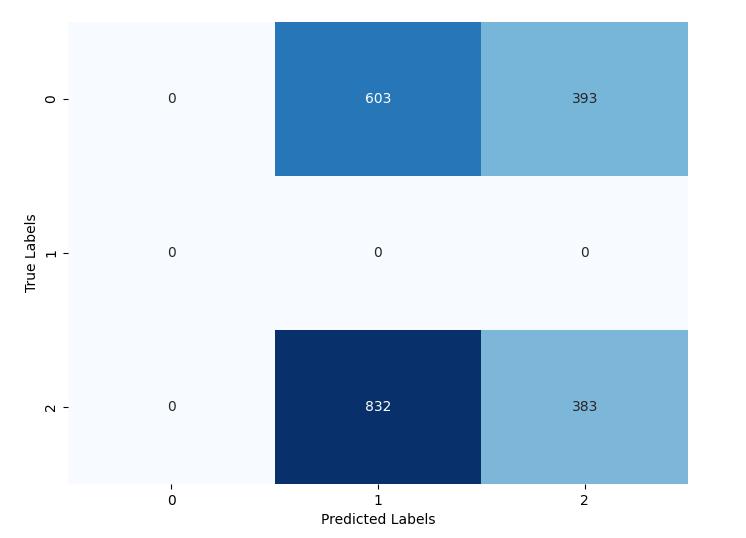
model = tf.models.Sequential()  
model.add(tf.layers.Dense(units=2))

model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['accuracy'])  
history = model.fit(x\_train, y\_train, epochs=500, validation\_split=0.2, verbose=0)  
  
example\_features = CollectData()[URL]  
if example\_features is None:  
 quit(print('DATA EXTRACTION FAILED'))  
  
y\_pred = np.argmax(model.predict(x\_test), axis=-1)  
  
*# Reshape example features into a 2D numpy array:*example\_features\_array = np.array(example\_features).reshape(1, -1)  
  
*# Evaluate the model*:  
cm = confusion\_matrix(y\_test, y\_pred)  
print('Confusion Matrix:\n', cm)  
  
*# Loss:*print(f'Loss: {history.history["loss"][-1]}')  
loss = history.history['loss']  
  
*# MSE:*print('MSE:', mse(y\_test, y\_pred))  
  
prediction = model.predict(example\_features\_array)  
print(f"Prediction: {prediction}")  
  
*# Plot*plt.plot(loss)  
plot\_confusion\_matrix(cm)

התחלנו את הקוד מייבוא הספריות הרלוונטיות לפרויקט שלנו. לאחר מכן, הגדרנו פונקציה בשם plot\_confusion\_matrix, המשמשת לשרטוט Confusion Matrix באמצעות הספריה Seaborn. אנו משתמשים בפונקציה בסוף הקוד, בעת ניתוח התוצאות.

בהתחלה, אנו קוראים את Training Dataset.csv – שהוא קובץ מסוג csv, Comma Separated Values, המכיל את המידע הנחוץ לאימון המודל – ערכי כל ה-Attributes, והתוצאה המתאימה להם – האם האתר עבורו נקבעו ה-Attributes שבכל שורה הוא אתר Phishing, או לא. לאחר מכן, אנו מאחסנים במשתנה df את הטבלה של ה-Attributes הרלוונטיים לנו – כלומר, אלו שאנו יכולים לחלץ מכתובת ה-URL הניתנת לנו.

לאחר מכן, אנו מפרידים בין התוצאות ל-Attributes, מפצלים את המידע לקבוצת אימון ולקבוצת בדיקה, ואז אנו יוצרים את הרשת, המכילה שכבה אחת ובה שני נוירונים (בשביל 2 תוצאות אפשריות – "malicious" או "non-malicious"). אחרי כן, אנו מייבאים את המידע מכתובת ה-URL באמצעות הספרייה CollectData אותה כתבנו, ומנבאים האם האתר הוא Phishing או לא עבור המידע אותו חילצנו. לאחר מכן, אנו בודקים את המודל – מדפיסים למסך את ה-Confusion Matrix, ה-Loss וה-MSE. לאחר כל הבדיקות, אנו מדפיסים את התוצאה של המודל למסך. במקרה זה, בדקנו האם האתר [www.nike.com](http://www.nike.com) זדוני או לא.

תוצאות הריצה של הקוד הן:

Loss: 0.31803420186042786

MSE: 1.3600180913613749

תמונה שמכילה צילום מסך, טקסט, מלבן, תרשים

התיאור נוצר באופן אוטומטיבציר ה-x של ה-Confusion Matrix נמצאים הערכים שחזה המודל, ובציר ה-y נמצאים הערכים האמיתיים. בשביל שהתוצאה של ה-Confusion Matrix תהיה מיטבית, עליה ליצור אלכסון כך שכל ערך ברשימת הנתונים המקורית יהיה זהה לערך המקביל לו ברשימת הנתונים שהמודל חזה, באופן הבא:

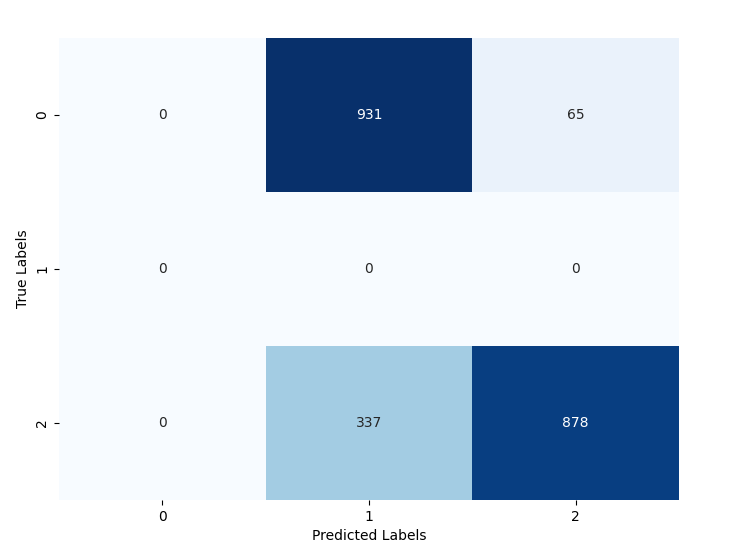
לפי ה-Confusion Matrix, ה-Loss וה-MSE שקיבלנו, הסקנו כי המודל שלנו לא עובד כמו שצריך. לכן, רצינו לשפר אותו. השיפור שלנו נעשה על ידי הוספת פונקציית אקטיבציה – כלומר, שינינו את שורת הקוד:

model.add(tf.layers.Dense(units=2))

model.add(tf.layers.Dense(units=2, activation="sigmoid"))

ל:

בחרנו בפונקציית ה-sigmoid על מנת שהמודל יניב תוצאות בין 0 ל-1, כאשר אתר תמים יניב תוצאה הגדולה מ-0.5, ואתר ייחשב לזדוני אם התוצאה קטנה מ-0.5.

תוצאות התכנית החדשה היו:

Loss: 0.5743133425712585

MSE: 0.6910900045228403

ניתן לראות כי על אף שחל שיפור משמעותי ב-MSE, ה-Loss לא השתפר, ואף נעשה גרוע יותר. כמו כן, מטריצת הבלבול כדי לשפר את המודל הפעם, בחרנו להוסיף שכבת נוירונים נוספת, אשר משתמשת בפונקציית האקטיבציה softmax – המסייעת בהקטנת הערכים הקטנים כבר והגדלת הערכים הגדולים כבר, על מנת לייצר הפרדה יותר ברורה בין שני הערכים. כלומר, לפני יצירת השכבה הקיימת, הוספנו את השורה:

**model.add(tf.layers.Dense(units=2, activation="softmax"))**  
model.add(tf.layers.Dense(units=2, activation="sigmoid"))

תמונה שמכילה טקסט, צילום מסך, מלבן, תרשים

התיאור נוצר באופן אוטומטיהתוצאות של המודל החדש:

Loss: 0.5698587894439697

MSE: 0.6015377657168702

ניתן לראות כי על אף שחל שיפור קל ב-Loss וב-MSE, המודל עדיין מניב הרבה תוצאות שגויות – בין אם בגלל שה-Loss מאוד גבוה, ובין אם בגלל שמטריצת הבלבול מאוד מגוונת, ומראה כי יש הרבה ערכים שהמודל חזה באופן שגוי.

החלטנו לשפר את המודל עוד יותר, באמצעות הפיכת המודל לבעיית קלסיפיקציה – כלומר, אם האתר זדוני המודל יקבל את הערך 0, ואם האתר תמים המודל יקבל את הערך 1. הוספנו את שורות הקוד הבאות לפני אימון המודל:

**y1 = y \* 0  
y1[np.argwhere(y == 1)] = 1  
y2 = y \* 0  
y2[np.argwhere(y == -1)] = 1  
y\_new = np.array([y1, y2]).T**  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, **y\_new**, test\_size=0.2, random\_state=1)

...

**state = np.array([0, 1])**

תמונה שמכילה צילום מסך, טקסט, מלבן, ריבוע

התיאור נוצר באופן אוטומטילמעשה, יצרנו שתי קבוצות של קלסיפיקציה, המעבדות את הנתונים שהמחלקה CollectData מחזירה: 1- ו-1. אימנו את המודל החדש שלנו, כאשר המשתנה state משמש למיפוי הערכים שהמודל חוזה והערכים האמיתיים לקטגוריות של "כתובות URL זדוניות" ו"כתובות URL לא זדוניות", וכך משמש לחישוב מטריצת הבלבול, המשווה את הקטגוריות החזויות לקטגוריות בפועל. נבדוק את התוצאות המתקבלות:

Loss: 0.05483977124094963

MSE: 0.07281772953414745

כלומר, נראה שהמודל עובד בצורה מצוינת – עם כ-94 אחוזי הצלחה. ככל הנראה, הבעיה לאורך היסטוריית השינויים של המודל שלנו היתה שלא השתמשנו בקלסיפיקציה.

נבדוק את התחזית של ה-URL:

Prediction: 0.791153609752655

The Features List: [-1, -1, -1, -1, -1, -1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1]

The example URL is predicted to be non-malicious.

ואכן, האתר שבדקנו ([www.nike.com](http://www.nike.com)) אינו זדוני, כלומר המודל צדק.

נבדוק גם אתר אחר שנלקח מרשימת אתרים זדוניים (<https://github.com/jarelllama/Scam-Blocklist/blob/main/domains.txt>). הקישור אל האתר הוא <https://3d-printing.store/>. תוצאות הריצה היו מדויקות:

תמונה שמכילה טקסט, צילום מסך, תרשים, מלבן

התיאור נוצר באופן אוטומטי

Loss: 0.053699199110269547

MSE: 0.07598371777476255

Prediction: 0.0657966285943985

The Features List: [-1, -1, -1, -1, -1, 1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1]

The example URL is predicted to be malicious.

להלן תמונה של האתר:

תמונה שמכילה טקסט, צילום מסך, שעון, תוכנה

התיאור נוצר באופן אוטומטי

ואכן, מצאנו שהאתר הנבדק הוא אתר זדוני. עוד אישוש לכך הוא אנטי-וירוס "VirusTotal" שמצא כי 2 מנועי סריקה מצאו את הקישור שלנו כחשוד:

תמונה שמכילה טקסט, צילום מסך, מספר, תוכנה

התיאור נוצר באופן אוטומטי

# פרק 4 – סיכום

נתחיל את הסיכום בהצגת התוצאות המסכמות של התכנית. להלן מטריצת הבלבול, ה LOSS, וה MSE של המודל:

Loss: 0.05387985333800316

MSE: 0.07553143374038897

תמונה שמכילה טקסט, צילום מסך, תצוגה, מלבן

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט, צילום מסך, מלבן, תרשים

התיאור נוצר באופן אוטומטי

כפי שאמרנו בעבר, ניתן לראות את התוצאות המדויקות של המודל, הן ב LOSS וה MSE המאוד קטנים (ניתן לראות את ה LOSS בגרף השמאלי), והן במטריצת הבלבול שמאוד קרובה למטריצה המיטבית עבור המודל. ניתן לראות כי ה MSE משתקף במטריצת הבלבול, שמראה על כקרוב ל-94 אחוזי הצלחה. הדבר אומר כי המודל עובד בצורה מספיק טובה לחזות אתרים זדוניים ותמימים.

בתחילת העבודה על המודל, היינו צריכים לקרוא המון על כל Attribute, ואחר כך לנסות להבין איך אנחנו מוצאים אותו מ Domain של אתר באמצעות Python, דבר שלא היה קל בכלל. לאחר מכן, היינו צריכים לכתוב את המחלקה CollectData (כזכור נמצאת בנספחים), שבה מצויה כל פונקציה המוצאת את המידע הרלוונטי בנוגע לכל Attribute. זמן מה לאחר תחילת העבודה הבנו שאנו לא מסוגלים לחלץ את המידע מכל Attribute במודל, מכיוון שהדבר אינו אפשרי בשפת Python, וככל הנראה לא אפשרי כלל. לכן, היינו צריכים לסנן בעת הקריאה מקובץ ה Excel את הAttributes שאנו לא יכולים לחלץ.

לאחר סיום כתיבת המחלקה, התחלנו לכתוב מודל פשוט, והאבן הדרך הראשונה הייתה לכתוב תכנית שלא תקרוס, ותציג נתונים התחלתיים, גם אם הם שגויים. לאחר שצלחנו זאת, התחלנו לחקור מה יכול לשפר את התוצאות של המודל. שינינו את כמות שכבות הנוירונים ואת כמות השכבות בכל שכבה, תוך שימוש בפונקציות אקטיבציה "Sigmoid" ו "SoftMax" בשביל תוצאות מיטביות. הדבר לא היה מספיק, והיינו צריכים לחפור עמוק יותר בשביל לממש את הפוטנציאל המלא שהמודל יכול להגיע אליו. אחרי חשיבה רבה והתבוננות על מודלים אחרים, גילינו בדיוק איך ליישם קלסיפיקציה נכונה וחכמה למודל. כך הגענו לכ-94 אחוזי הצלחה, ואפשר להגיד כי נשמנו לרווחה.

תוך כדי העבודה וכתיבת המודל, למדנו המון על החומר שנלמד במהלך השנה, והשכלנו המון בתחום הMachine Learning ו Neural Networks. למדנו שיטות חישוביות רבות ומגוונות, ומושגים רבים שעזרו לנו במהלך הכתיבה. ככל שהתעמקנו בעולם של למידת מכונה ורשתות נוירונים, ההבנה כיצד פועלים נוירונים בודדים הפכה חיונית עבור הפרויקט שלנו על חיזוי כתובות אתרים זדוניות במודל UCI Phishing Websites.

ראשית, כשבנינו את ארכיטקטורת רשת הנוירונים, הבנו את המשמעות של נוירונים בודדים. למדנו שנוירונים פועלים כיחידות עיבוד מידע, קולטים Input, מיישמים טרנספורמציות ומייצרים Output, והם עושים זאת באמצעות פונקציות Cost ו MSE. דבר זה עזר לנו להבין יותר את המושגים הנ"ל, וכך היה לנו יותר קל ליישם זאת במודל. החקר שלנו על מושגים כמו Gradient Descent, Weights Constants סיפקו לנו תובנות חשובות לגבי ההתנהגות והפעולות של נוירונים בודדים בתוך רשתות נוירונים. הבנה זו אפשרה לנו לקבל החלטות מושכלות בנוגע לארכיטקטורה, אופטימיזציה וכיוונון עדין של המודל שלנו, ובסופו של דבר שיפרה את יכולתו ללכוד דפוסים מורכבים ולסווג כתובות URL זדוניות בצורה מדויקת.

שנית, בחירת פונקציות אקטיבציה מתאימות הייתה חיונית להצלחת המודל שלנו. בדקנו אפשרויות שונות, כגון פונקציות "Sigmoid", "SoftMax" ו "Tanh", והבנו כיצד הן מאפשרות לרשת ללכוד קשרים מורכבים בנתונים. במהלך כתיבת המודל, היו לנו הרבה קשיים בבחירת הפונקציה הנכונה, והמון ניסויים וטעיות. אך עם הטעיות הגיעו תובנות, ולמדנו המון שיטות חשיבה חדשות בכל הנוגע לשימוש נכון של פונקציות אקטיבציה כתלות במודל ובמקרה הספציפי הנידון. באמצעות ניסויים ותצפית על התנהגות המודל, השגנו תובנות לגבי ההשפעה של פונקציות הפעלה על ביצועי הרשת והיכולת שלה לסווג כתובות URL בצורה מדויקת.

יתר על כן, קלסיפיקציה, היה מושג בסיסי נוסף שנתקלנו בו ויישמנו. זיהינו את החשיבות של הכשרת המודל להבחין בין כתובות URL זדוניות לתמימות. בהתחלה היה קשה ליישם את הקלסיפיקציה הנכונה למודל שלנו, ולא הצלחנו להבין איך ליישם את הטכניקה. עם הזמן והתבוננות על מודלים ומערכות אחרות, למשל Iris Data ומודל ה Breast Cancer שעבדנו עליו , הבנו את השיטה בצורה יותר טובה, וצברנו שיטות חשיבה שונות הנוגעות בקלסיפיקציה, הנוגעות בעיקר קטלוג של הנתונים בצורה נכונה וחישוב נכון של דיוק המודל. על ידי הגדרת קטגוריות מתאימות ואופטימיזציה של המודל באמצעות טכניקות כמו MSE ומדדי דיוק שונים כגון LOSS ו Confusion Matrix, הצלחנו לבסוף להגיע לתוצאה הרצויה.

לסיכום, נגיד שמאוד נהנינו מהעבודה. חיזקנו את הידע שלנו במושגים שאנו כבר מבינים, השכלנו עם מושגים חדשים במהלך כתיבת המודל, ולמדנו שיטות חשיבה רבות ומגוונות, שלא היינו מגיעים אליהן לולא העבודה. למרות כל הקשיים שחווינו במהלך העבודה, הצלחנו להגיע לתוצאה שרצינו, ואנו גאים בתוצר שלנו. אנחנו בהחלט נמשיך את המחקר שלנו במציאת קישורים זדוניים.

# נספחים

## המחלקה CollectData

import re  
import ssl  
import socket  
import whois21  
import requests  
from datetime import datetime, timedelta  
  
IP\_REGEX = re.compile(r'^(https?://)?([1-2]?\d{1,2}.){3}[1-2]?\d{1,2}')  
HEX\_IP\_REGEX = re.compile(r'^(https?://)?(0x[\da-fA-F]{2}\.){3}(0x[\da-fA-F]{2})')  
DOMAIN\_REGEX = re.compile(r'^(https?://)?(www.)?(\w+\.)+\w+')  
TINY\_URL\_LIST = ['t.co', 'goo.gl', 'bit.ly', 'amzn.to', 'tinyurl.com', 'ow.ly', 'youtu.be']  
TINY\_URL\_REGEX = re.compile(fr'^(https?://)?(www.)?({"|".join(TINY\_URL\_LIST)})/')  
  
  
def get\_domain(url):  
 for i in ('http://', 'https://'):  
 if url.startswith(i):  
 url = url[len(i):]  
 if url.startswith('www.'):  
 url = url[4:]  
 return url.split('/')[0]  
  
  
def add\_prefix(url, https=True):  
 if https:  
 return 'https://www.' + url  
 return 'http://www.' + url  
  
class CollectData:  
  
 @staticmethod  
 def \_\_getitem\_\_(url: str):  
 if not any((DOMAIN\_REGEX.match(url), IP\_REGEX.match(url), HEX\_IP\_REGEX.match(url))):  
 return print('ERROR: url given is not formatted correctly.')  
 if url.startswith('http://'):  
 url = add\_prefix(get\_domain(url), False)  
 url = add\_prefix(get\_domain(url))  
 whois = CollectData.whois(get\_domain(url))  
 if whois is None: return print('ERROR: WhoIs failed.')  
 print(' - Got WhoIs!')  
 redirects, html, js = CollectData.get\_source\_code(url)  
 print(' - Got Source Code!')  
 data = []  
 data.append(CollectData.check\_ip\_address(url)); print(' 1 IP URL')  
 data.append(CollectData.check\_url\_length(url)); print(' 2 URL LEN')  
 data.append(CollectData.check\_tiny\_url(url)); print(' 3 TINY URL')  
 data.append(CollectData.check\_at(url)); print(' 4 @')  
 data.append(CollectData.check\_redirect(url)); print(' 5 //')

data.append(CollectData.check\_minus(url)); print(' 6 -')  
 data.append(CollectData.count\_subdomain(url)); print(' 7 SUBDOMAIN')  
 data.append(CollectData.certificate(get\_domain(url))); print(' 8 CERTIFICATE')  
 data.append(CollectData.domain\_expiry(whois)); print(' 9 DOMAIN EXPIRE')  
 data.append(CollectData.check\_favicon(url, html)); print('10 FAVICON')  
 data.append(CollectData.https\_token(url)); print('11 TOKEN')  
 data.append(CollectData.request\_url(url, html)); print('12 REQUEST')  
 data.append(CollectData.anchor\_url(url, html)); print('13 ANCHOR')  
 data.append(CollectData.meta\_script\_link\_url(url, html)); print('14 META SCRIPT LINK')  
 data.append(CollectData.check\_redirects(redirects)); print('15 REDIRECTS')  
 data.append(CollectData.check\_status\_bar\_change(html, js)); print('16 STATUS BAR')  
 data.append(CollectData.check\_right\_click\_disable(html, js)); print('17 RIGHT CLICK')  
 data.append(CollectData.check\_popup(html, js)); print('18 POPUP')  
 data.append(CollectData.check\_iframe(html)); print('19 IFRAME')  
 data.append(CollectData.domain\_age(whois)); print('20 DOMAIN AGE')  
 data.append(CollectData.dns\_records(whois)); print('21 DNS RECORDS')  
 print('Finished Extracting Data!')  
 return data  
  
@staticmethod  
def whois(url):  
 for i in range(3):  
 whois = whois21.WHOIS(url)  
 if whois.success and whois.expires\_date and whois.creation\_date:  
 return whois  
 return None

@staticmethod  
def get\_source\_code(url):  
 response = requests.get(url, allow\_redirects=True)  
 redirects = len(response.history)  
 html\_source = response.text  
  
 js\_source = ''  
 for js\_url in re.findall(r'<script (.+ )?src="(.\*?)"', html\_source):  
 if js\_url[-1].startswith('/'): js\_url = url + js\_url[-1]  
 else: js\_url = js\_url[-1]  
 js\_response = requests.get(js\_url)  
 js\_source += js\_response.text  
  
 return redirects, html\_source.lower(), js\_source.lower()  
  
@staticmethod  
def check\_ip\_address(url):  
 return 1 if IP\_REGEX.match(url) or HEX\_IP\_REGEX.match(url) else -1  
  
@staticmethod  
def check\_url\_length(url):  
 return -1 if len(url) < 54 else 0 if 54 <= len(url) <= 75 else 1  
  
@staticmethod  
def check\_tiny\_url(url):  
 return 1 if TINY\_URL\_REGEX.match(url) else -1

@staticmethod  
def check\_at(url):  
 return 1 if '@' in url else -1

@staticmethod  
def check\_redirect(url):  
 return 1 if url.find('//', 8) > 0 else -1  
  
@staticmethod  
def check\_minus(url):  
 return 1 if '-' in url else -1  
  
@staticmethod  
def count\_subdomain(url):  
 return -1 if url.count('.') - 2 <= 1 else 0 if url.count('.') - 2 == 2 else 1  
  
@staticmethod  
def certificate(url):  
 context = ssl.create\_default\_context()  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 with context.wrap\_socket(sock, server\_hostname=url) as conn:  
 conn.connect((url, 443))  
 cert = conn.getpeercert()  
 expiry\_date = datetime.strptime(cert['notAfter'], '%b %d %H:%M:%S %Y GMT')  
 return 1 if expiry\_date - datetime.now() <= timedelta(days=365) else -1

@staticmethod  
def domain\_expiry(whois):  
 return 1 if whois.expires\_date - datetime.now() <= timedelta(days=365) else -1  
  
@staticmethod  
def check\_favicon(url, html):  
 link = re.search(r'<link(.|\n)\*?>', html)  
 if link is None: return -1  
 link = link.group()  
 link = link[link.find('href'):]  
 if link.find('"') < 0 or 0 < link.find("'") < link.find('"'):  
 href = link.split("'")[1]  
 else:  
 href = link.split('"')[1]  
 return 1 if DOMAIN\_REGEX.match(href) and get\_domain(url) != get\_domain(href) else -1  
  
@staticmethod  
def https\_token(url):  
 return 1 if 'https' in url else -1

@staticmethod  
def request\_url(url, html):  
 images = re.finditer(r'<img(.|\n)\*?>', html)  
 outbound = 0  
 total = 0  
 for i in images:  
 i = i.group()  
 if 'href' in i:  
 src = i[i.find('href'):]  
 if src.find('"') < 0 or 0 < src.find("'") < src.find('"'):  
 src = src.split("'")[1]  
 else:  
 src = src.split('"')[1]

if DOMAIN\_REGEX.match(src) and get\_domain(url) != get\_domain(src):  
 outbound += 1  
 total += 1  
 elif 'src' in i: total += 1  
if total == 0: return -1  
return -1 if outbound / total < 22 else 0 if 22 <= outbound / total <= 61 else 1

@staticmethod  
def anchor\_url(url, html):  
 links = re.finditer(r'<a(.|\n)\*?>', html)  
 outbound = 0  
 total = 0  
 for i in links:  
 i = i.group()  
 for j in ('href', 'content'):  
 if j in i:  
 src = i[i.find(j):]  
 if src.find('"') < 0 or 0 < src.find("'") < src.find('"'):  
 src = src.split("'")[1]  
 else:  
 src = src.split('"')[1]  
 if (DOMAIN\_REGEX.match(src) and get\_domain(url) != get\_domain(src)) or not DOMAIN\_REGEX.match(src):  
 outbound += 1  
 total += 1  
 if 'src' in i: total += 1  
 if total == 0: return -1  
 return -1 if outbound / total < 31 else 0 if 31 <= outbound / total <= 67 else 1

@staticmethod  
def meta\_script\_link\_url(url, html):  
 outbound = 0  
 total = 0  
 for i in ('meta', 'script', 'link'):  
 links = re.finditer(fr'<{i}(.|\n)\*?>', html)  
 for i in links:  
 i = i.group()  
 if 'href' in i:  
 src = i[i.find('href'):]  
 if src.find('"') < 0 or 0 < src.find("'") < src.find('"'):  
 src = src.split("'")[1]  
 else:  
 src = src.split('"')[1]  
 if DOMAIN\_REGEX.match(src) and get\_domain(url) != get\_domain(src):  
 outbound += 1  
 total += 1  
 elif 'src' in i: total += 1  
 if total == 0: return -1  
 return -1 if outbound / total < 17 else 0 if 17 <= outbound / total <= 81 else 1

@staticmethod  
def check\_redirects(redirects):  
 return -1 if redirects <= 1 else 0 if 2 <= redirects <= 4 else 1

@staticmethod  
def check\_status\_bar\_change(html, js):  
 return 1 if (re.match(r'<script(.|\n)\*?>', html) is not None and  
 re.search(r'onmouseover\s\*=\s\*([\'"])(.\*?)\1',  
 re.search(r'<script(.|\n)\*?>(.|\n)\*?</script>', html).group()) and  
 'window.status' in re.search(r'onmouseover\s\*=\s\*([\'"])(.\*?)\1',  
 re.search(r'<script(.|\n)\*?>(.|\n)\*?</script>', html).group()).group(2)) \  
 or \  
 (re.search(r'onmouseover\s\*=\s\*([\'"])(.\*?)\1', js) and  
 'window.status' in re.search(r'onmouseover\s\*=\s\*([\'"])(.\*?)\1', js).group(2)) \  
 else -1  
  
@staticmethod  
def check\_right\_click\_disable(html, js):  
 return 1 if (re.match(r'<script(.|\n)\*?>', html) is not None and  
 'event.button==2' in re.search(r'<script(.|\n)\*?>(.|\n)\*?</script>', html).group()) or \  
 'event.button==2' in js else -1  
  
@staticmethod  
def check\_popup(html, js):  
 return 1 if (re.match(r'<script(.|\n)\*?>', html) is not None and  
 'window.prompt' in re.search(r'<script(.|\n)\*?>(.|\n)\*?</script>', html).group()) or \  
 'window.prompt' in js else -1  
  
@staticmethod  
def check\_iframe(html):  
 return -1 if re.match(r'<iframe(.|\n)\*?>', html) is None else 1  
  
@staticmethod  
def domain\_age(whois):  
 return 1 if whois.creation\_date - datetime.now() < timedelta(days=182) else -1  
  
@staticmethod  
def dns\_records(whois):  
 return -1 if whois.name\_servers else 1

כפי שניתן לראות, התחלנו את הקוד שלנו על ידי ייבוא הספריות המתאימות:

* **ssl**- קבוצה של פרוטוקולים ואלגוריתמים קריפטוגרפים המשמשים לאבטחת תקשורת רשת בין לקוחות ושרתים. בקוד שלנו, הספרייה משמשת לבדיקת תאריך תפוגה של אתר בכתובת URL מסוימת.
* **socket** - יצירה וטיפול בשקעי רשת, המאפשרים תקשורת בין תהליכים שונים במחשבים שונים או באותו מחשב, דרך רשת. בקוד שלנו, הספרייה משמשת ביחד עם ssl להשגת Certificate של אתר מסוים.
* **whois** - ספריית whois משמשת בדרך כלל לשאילתה ואחזור של פרטי רישום דומיין כגון הרושם, הרשם, תאריכי רישום ותפוגה ופרטים נוספים על שמות דומיין. בקוד זה, ספריית whois משמשת לאחזור פרטי רישום דומיין של כתובת URL נתונה, המשמשת בכמה משיטות חילוץ הנתונים.
* **requests -** ספריית 'requests' משמשת לביצוע בקשות HTTP ב-Python, והיא מפשטת את תהליך שליחת בקשות HTTP/1.1. זוהי ספרייה קלה לשימוש להגשת בקשות HTTP וטיפול בתגובות. ניתן להשתמש בו לביצוע בקשות HTTP שונות כגון GET, POST, PUT, DELETE וכו'. בקוד זה, ספריית 'requests' משמשת לאחזור תוכן HTML של אתר אינטרנט על ידי שליחת בקשת HTTP GET. שיטת get\_source\_code משתמשת בשיטת requests.get כדי לאחזר את תוכן ה-HTML של אתר אינטרנט, ולאחר מכן מחלצת ממנו קוד JavaScript באמצעות ביטוי רגולרי. שיטת request\_url משתמשת בשיטת requests.get כדי לבדוק אם כתובת URL נגישה או לא.

לאחר מכן, הגדרנו ארבעה ביטויי regex. מטרת הביטויים הרגולריים המוגדרים בקוד שלנו היא להתאים ולחלץ סוגים שונים של כתובות URL וכתובות IP מנתוני טקסט. לאחר מכן נעשה שימוש בביטויים רגולריים אלה בשילוב עם ספריית ה 'requests' כדי לחלץ מידע מדפי אינטרנט.

יצרנו בתחילת המחלקה את הפונקציה \_\_getitem\_\_ , המחזירה רשימה data שמכילה את המידע מכל הפונקציות הסטטיות של המחלקה. הרשימה מוסיפה לעצמה את המידע מכל פונקציה, וההוספה מסודרת לפי סדר ה [Attributes שרשום בתחילת הפרק](#attributes_for_module). הרשימה תיראה כדלהלן:

print(CollectData()["www.nike.com"]) 🡪

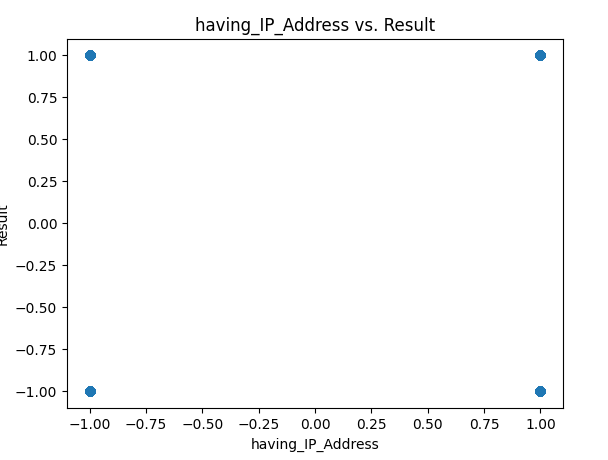
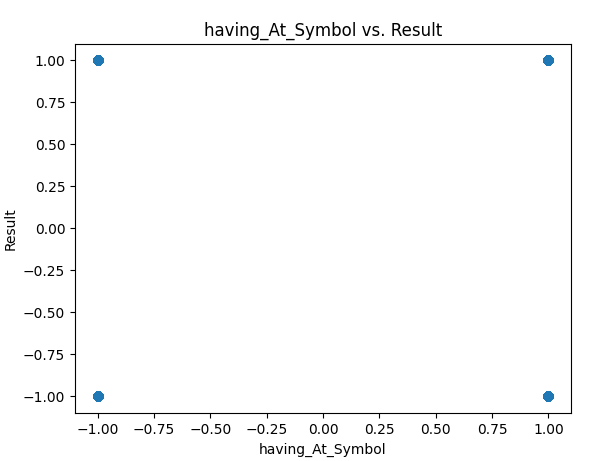
[-1, -1, -1, -1, -1, -1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1]

## גרפי הנתונים והיסטוגרמות ה Attributes

להלן התכנית עצמה שמשרטטת את הגרפים:

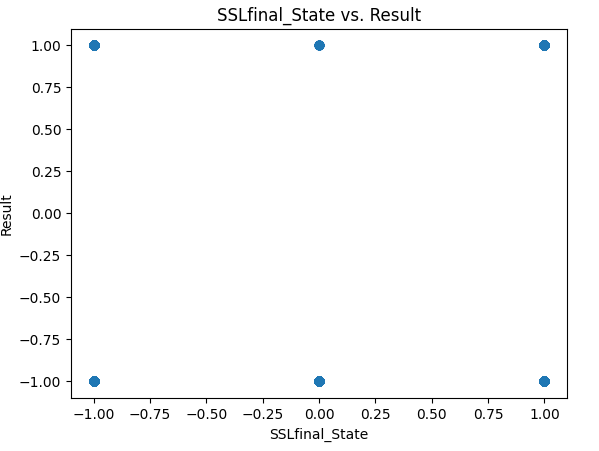
import pandas as pd  
import matplotlib.pyplot as plt  
  
*# Load the Excel chart into a Pandas DataFrame*df = pd.read\_csv("Training\_Dataset.csv")  
  
df = pd.DataFrame((df['having\_IP\_Address'], df['URL\_Length'], df['Shortening\_Service'],  
 df['having\_At\_Symbol'], df['double\_slash\_redirecting'], df['Prefix\_Suffix'],  
 df['having\_Sub\_Domain'], df['SSLfinal\_State'], df['Domain\_registration\_length'],  
 df['Favicon'], df['HTTPS\_token'], df['Request\_URL'], df['URL\_of\_Anchor'], df['Links\_in\_tags'],  
 df['Redirect'], df['on\_mouseover'], df['RightClick'], df['popUpWindow'], df['Iframe'],  
 df['age\_of\_domain'], df['DNSRecord'], df['Result'])).T  
  
*# Loop through each column (excluding "Result") and plot a graph*for col in df.columns[:-1]:  
 plt.scatter(df[col], df["Result"])  
 plt.xlabel(col)  
 plt.ylabel("Result")  
 plt.title(f"{col} vs. Result")  
 plt.show()  
  
*# Plot a histogram of the "Result" column*plt.hist(df["Result"])  
plt.xlabel("Result")  
plt.ylabel("Frequency")  
plt.title("Distribution of Result")  
plt.show()  
  
*# Plot a histogram of the "URL\_Length" column*plt.hist(df["URL\_Length"])  
plt.xlabel("URL\_Length")  
plt.ylabel("Frequency")  
plt.title("Distribution of URL Length")  
plt.show()

הגרפים נראים בצורה הבאה (חוץ מהגרפים של הAttributes having\_Sub\_Domain, Shortening\_Service שנמצאים [בתחילת פרק 3 של העבודה](#graphs_for_attributes)):

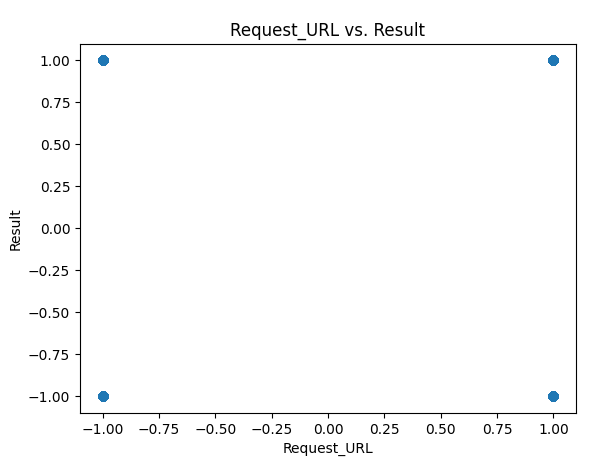
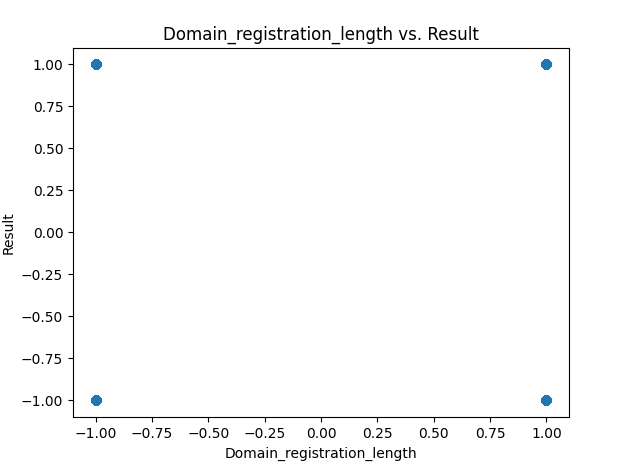
תמונה שמכילה טקסט, צילום מסך, מספר, תצוגה

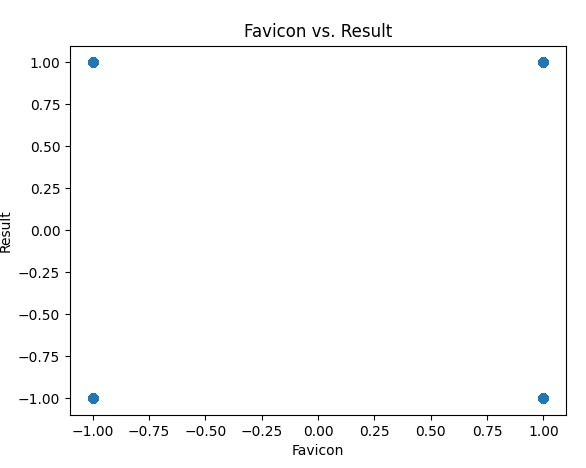
התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי

**תמונה שמכילה טקסט, צילום מסך, מספר

התיאור נוצר באופן אוטומטי**

****תמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

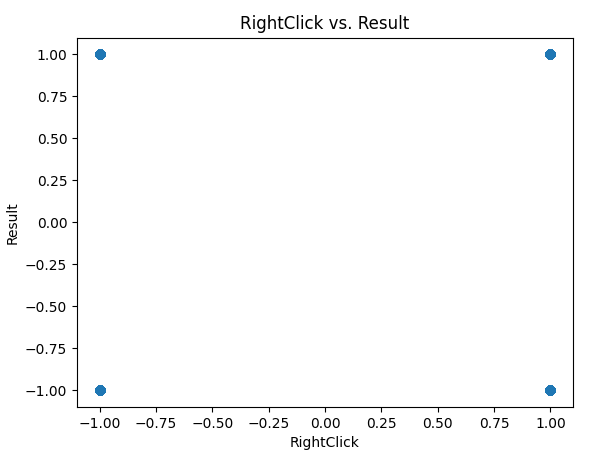
התיאור נוצר באופן אוטומטי****

**תמונה שמכילה טקסט, צילום מסך, מספר, תצוגה

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי*תמונה שמכילה טקסט, צילום מסך, מספר, קו

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי***

**תמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, תצוגה, מספר

התיאור נוצר באופן אוטומטי**

## התכנית של המודל

import random  
import argparse  
import numpy as np  
import pandas as pd  
import seaborn as sns  
import tensorflow.keras as tf  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix, mean\_squared\_error as mse  
from Data import CollectData  
  
*# Parse command-line arguments*parser = argparse.ArgumentParser(description='URL Classification')  
parser.add\_argument('url', type=str, help='URL to classify')  
args = parser.parse\_args()  
  
URL = args.url *# URL = 'www.3d-printing.store'*if not URL:  
 print('URL not provided. Exiting...')  
 exit()

def plot\_confusion\_matrix(cm):  
 *"""  
 This function takes in true labels and predicted labels as NumPy arrays and plots the corresponding confusion matrix using Seaborn.  
 Parameters:  
 cm: the confusion matrix  
 Returns:  
 None  
 """  
  
 # Set the figure size* plt.figure(figsize=(8, 6))  
  
 *# Create a heatmap using Seaborn* sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)  
  
 *# Add labels to the x and y axes* plt.xlabel('Predicted Labels')  
 plt.ylabel('True Labels')  
  
 *# Show the plot* plt.show()  
  
  
df = pd.read\_csv('Training\_Dataset.csv')  
print('Got CSV')  
  
df = pd.DataFrame((df['having\_IP\_Address'], df['URL\_Length'], df['Shortening\_Service'],  
 df['having\_At\_Symbol'], df['double\_slash\_redirecting'], df['Prefix\_Suffix'],  
 df['having\_Sub\_Domain'], df['SSLfinal\_State'], df['Domain\_registration\_length'],  
 df['Favicon'], df['HTTPS\_token'], df['Request\_URL'], df['URL\_of\_Anchor'], df['Links\_in\_tags'],  
 df['Redirect'], df['on\_mouseover'], df['RightClick'], df['popUpWindow'], df['Iframe'],  
 df['age\_of\_domain'], df['DNSRecord'], df['Result'])).T  
  
*# Split the dataset into features and target*x = np.array(df.iloc[:, :-1]) *# data*y = np.array(df.iloc[:, -1]) *# result*

*# Classify*y1 = y \* 0  
y1[np.argwhere(y == 1)] = 1  
y2 = y \* 0  
y2[np.argwhere(y == -1)] = 1  
y\_new = np.array([y1, y2]).T  
  
*# Split the data into training and testing sets*x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y\_new, test\_size=0.2, random\_state=1)  
  
model = tf.models.Sequential()  
model.add(tf.layers.Dense(units=2, activation='sigmoid')) *# output - 2 attribute*model.add(tf.layers.Dense(units=2, activation='softmax')) *# output - 2 attribute  
  
# Compile the model*model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['accuracy'])

*# Train the model*history = model.fit(x\_train, y\_train, epochs=1000, validation\_split=0.2, verbose=0)  
  
example\_features = CollectData()[URL]  
if example\_features is None:  
 quit(print('DATA EXTRACTION FAILED'))  
  
*# Predict on the test set*y\_pred = np.round(model.predict(x\_test))  
  
*# Reshape example features into a 2D numpy array*example\_features\_array = np.array(example\_features).reshape(1, -1)  
state = np.array([0, 1])  
  
*# Evaluate the model*cm = confusion\_matrix(np.dot(y\_test, state), np.dot(y\_pred, state))  
print('Confusion Matrix:\n', cm)  
  
*# Loss*print(f'Loss: {history.history["loss"][-1]}')  
loss = history.history['loss']  
  
*# MSE*print('MSE:', mse(y\_test, y\_pred))  
  
*# Predict the class (malicious or not) of the example URL*prediction = model.predict(example\_features\_array)  
prediction = np.dot(prediction, state)[0]  
print(f"Prediction: {prediction}")  
if prediction > 0.5:  
 print("The Features List: ", example\_features)  
 print("The example URL is predicted to be non-malicious.")  
else:  
 print("The Features List: ", example\_features)  
 print("The example URL is predicted to be malicious.")  
  
*# Plot*plt.plot(loss)  
plot\_confusion\_matrix(cm)

התכנית תקבל כפרמטר את ה URL, ואופן ההרצה שלה יהיה באופן הבא:

python original\_code\_network.py www.nike.com

כאשר במקום [www.nike.com](http://www.nike.com) יכול להיות כל קישור אחר, שכן הוא יתקבל כפרמטר לתכנית.

# אתרים

האתרים בהם השתמשנו לעבודה הם כדלהלן:

* <https://archive.ics.uci.edu/ml/index.php>
* <https://pypi.org/project/python-whois/>
* <https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285>
* <https://stackoverflow.com/questions/1087227/validate-ssl-certificates-with-python>
* https://www.virustotal.com/gui/home/url