Експериментално изследване върху набор от данни:

Откриване на фишинг атаки

Въведение

В динамичния дигитален свят, предотвратяването на зловредни атаки е от важно значение за защитата на лични данни и безопастността на информацията. Фишинг атаките са разпространен метод за компрометиране на сигурността и кражба на лични данни. Те се състоят в това потребителя да бъде подмамен да влезе в подправен фалшив сайт, които изглежда неразличимо от легитимната му версия.

Целта на тази разработка е да бъдат създадени, анализирани и сравнени алгоритми от машинното обучение за откриване на фишинг атака по URL адрес.

Експериментална рамка – методи и техника на анализ

### Набори от данни

Експерименталните изследвания се основават на набора от данни от сайта Kaggle: <https://www.kaggle.com/datasets/sajid576/sql-injection-dataset>,

[<https://www.kaggle.com/datasets/shashwatwork/web-page-phishing-detection-dataset>](https://www.kaggle.com/datasets/sajid576/sql-injection-dataset). Датасетът съдържа 11430 записа с по 87 колони – url, status (legitimate/phishing) и 85 характеристики, описващи url-a.

### Среда за разработка и библиотеки

Използваната среда за разработка е Jupyter Notebook в комбинация със следните инсталирани библиотеки:

- pandas, seaborn, matplotlib (зареждане/визуализация на данни),

- numpy (математическите изчисления),

- scikit-learn (имплементация на моделите)

### Алгоритми

#### 1. ****Наивен Бейс (Naïve Bayes)****

* **Как работи:** Основава се на теоремата на Бейс, като приема, че всички характеристики са независими. Изчислява вероятностите за различни класове на базата на характеристиките.
* **Силни страни:** Лесен за разбиране и бърз за обучение, подходящ за големи набори от данни.
* **Слаби страни:** Чувствителен към независимостта на характеристиките, не се справя добре при сложни зависимости между тях.

#### 2. ****K най-близки съседи (k-Nearest Neighbors, kNN)****

* **Как работи:** Класифицира данни, като намира най-близките k съседи в пространството на характеристиките и избира класа на мнозинството.
* **Силни страни:** Прост и ефективен за малки набори от данни, не изисква изрично обучение.
* **Слаби страни:** Става бавен при големи набори от данни, чувствителен към шума в данните и голямо количесто характеристики (нефилтрирани характеристики).

#### 3. ****Случайна гора (Random Forest)****

* **Как работи:** Използва ансамбъл от дървета на решения, като всяко дърво е обучено върху произволен поднабор от данните и характеристиките. Крайното решение е усреднено (за регресия) или взема мнозинството (за класификация).
* **Силни страни:** Много устойчив на пренасищане, справя се добре с големи и сложни данни.
* **Слаби страни:** Изисква повече ресурси за обучение и предсказване, неинтуитивен за интерпретация.

#### 4. ****Логистична регресия (Logistic Regression)****

* **Как работи:** Използва логистична функция за моделиране на вероятността за принадлежност към даден клас. Често се използва за бинарна класификация.
* **Силни страни:** Лесен за интерпретация, подходящ за линейно разделими данни.
* **Слаби страни:** Ограничен при нелинейни зависимости, чувствителен към мултиколинеарност.

#### 5. ****Невронни мрежи (Neural Networks)****

* **Как работи:** Моделира сложни зависимости чрез мрежа от взаимосвързани слоеве от изкуствени неврони. Използва методи като обратно разпространение за обучение.
* **Силни страни:** Много мощен при сложни задачи и големи набори от данни.
* **Слаби страни:** Изисква значителни изчислителни ресурси, труден за настройка и податлив на пренасищане без подходящи техники.

### Подход при изпълнението

Подходът включва използване на различни аналитични техники и инструменти, които ще ни позволят да идентифицираме уязвимости и да предложим ефективни мерки за защита.

* Предварителна обработка и почистване на данни;
* Анализ на набора от данни;
* Извличане на съществените особености с цел обучение на точен модел върху тях;
* Изграждане на модели;
* Предсказване и оценка – тестване на моделите на базата на тестови набор от данни и оценяването им по различни параметри (precision, recall, f1 score, accuracy);
* Сравнение на производителността на имплементираните модели.

Обработка и анализ на данните

### Импортиране на необходими библиотеки

В началото на кода са импортирани няколко основни библиотеки, необходими за анализ и обработка на данните:

* numpy и pandas за работа с масиви и структури от данни;
* seaborn и matplotlib.pyplot за визуализация на данните;
* модули от sklearn за предварителна обработка на данни, разделяне на данни и оценка на модели.

### Зареждане и първоначално изследване на данните

Данните се зареждат от CSV файл с помощта на pandas. За целите на анализа се извеждат 5 произволни записа от набора данни, за да се предостави преглед на съдържанието.

raw\_data = pd.read\_csv('dataset\_phishing.csv')

pd.set\_option('display.max\_columns', None)

raw\_data.sample(5)

Също така се проверява наличието на липсващи стойности в набора данни:

raw\_data.isna().sum()

### Премахване на колони с константни стойности

Чрез функцията remove\_constant\_columns се идентифицират и премахват колони, които имат една и съща стойност за всички редове:

def remove\_constant\_columns(df):

constant\_columns = [col for col in df.columns if df[col].nunique()==1]

print("Columns with constant values:", constant\_columns)

df\_cleaned = df.drop(columns=constant\_columns)

print("Number of remaining columns:", df\_cleaned.shape[1])

return df\_cleaned

df\_cleaned = remove\_constant\_columns(raw\_data)

### Преобразуване на целевата променлива и проверка на баланса

Целевата променлива status, която указва дали даден URL адрес е фишинг или не, се преобразува в числов формат (1 за фишинг, 0 за легитимен):

df\_cleaned['status'] = df\_cleaned['status'].map({'phishing': 1, 'legitimate': 0})

Балансът между класовете се визуализира чрез лентова диаграма:

df\_cleaned['status'].value\_counts().plot(kind='bar')

### Описание и визуализация на данните

Информация за типовете данни и липсващите стойности се извежда чрез:

df\_cleaned.info()

### Избор на числови данни

За по-нататъшни анализи се избират само числовите колони, изключвайки текстовата колона url:

numerical\_data = df\_cleaned.select\_dtypes(include=['float64', 'int64'])

### Анализ на корелациите

Изчислява се корелационната матрица за числовите колони, като специално внимание се обръща на корелацията със стойностите в колоната status:

corr\_matrix = numerical\_data.corr()

status\_corr = corr\_matrix['status']

### Избор на значими признаци чрез корелация

Функцията feature\_selector\_correlation избира признаци с корелация над определен праг спрямо status:

def feature\_selector\_correlation(cmatrix, threshold):

selected\_features = []

feature\_score = []

i = 0

for score in cmatrix:

if abs(score) > threshold:

selected\_features.append(cmatrix.index[i])

feature\_score.append(['{:3f}'.format(score)])

i += 1

result = list(zip(selected\_features, feature\_score))

return result

selected\_features = feature\_selector\_correlation(status\_corr, 0.23)

selected\_features = [feature for feature in selected\_features if feature[0] != 'status']

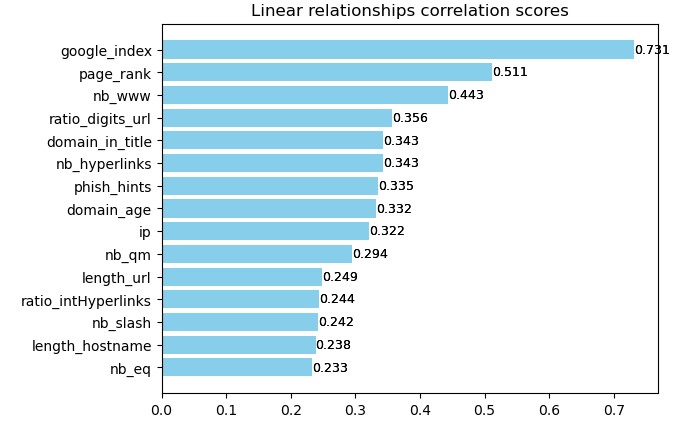
### Визуализация на резултатите от избора на признаци

Създава се диаграма, която показва линейните корелационни коефициенти на избраните признаци:

scores = pd.Series({item[0]: float(item[1][0]) for item in selected\_features})

absolute\_linear\_scores = scores.abs()

plot\_scores(absolute\_linear\_scores)



Фигура 1.Характеристики, подредени по линейната им корелация със статуса

### Допълнителен анализ на корелациите със Spearman

Чрез топлинна карта се визуализират корелациите между избрани признаци и целевата променлива:

def corr\_heatmap(data, idx\_s, idx\_e):

y = data['status']

temp = data.iloc[:, idx\_s:idx\_e]

if 'url' in temp.columns:

del temp['url']

temp['status'] = y

sns.heatmap(temp.corr(), annot=True, fmt='.2f')

plt.show()

corr\_heatmap(df\_cleaned, 72, 82)

### C:\Users\1\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot 2024-12-25 024619.jpg

Фигура 2.Топлинна карта за сверяване на получените корелации

### Използване на Mutual Information за избор на признаци

За по-прецизно определяне на значимите признаци се използва методът mutual\_info\_classif, който отчита както линейни, така и нелинейни зависимости:

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

X = df\_cleaned.drop(['url', 'status'], axis=1)

Y = df\_cleaned['status']

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)

absolute\_linear\_scores\_names = absolute\_linear\_scores.index.tolist()

plot\_scores(mi\_scores.head(15), highlight\_features=absolute\_linear\_scores\_names)

### C:\Users\1\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot 2024-12-25 024652.jpg

Фигура 3. Линейни (лилави) и нелинейни (сини) характеристики, корелиращи със статуса

### Подготовка на данните за обучение на моделите

Избраните признаци се използват за създаване на набори от данни за обучение и тест, като се използва train\_test\_split:

X = df\_cleaned[top\_mi\_scored\_features\_names]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=41)

### Стандартизация на данните

Преди обучение на моделите, данните се стандартизират с StandardScaler, за да се гарантира равнопоставеност на признаците:

scaler = StandardScaler()

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

X\_test\_scaled = scaler.transform(X\_test)

Този процес на анализ и предварителна обработка на данните осигурява добра основа за изграждането на ефективни машинно обучителни модели.

Изграждане на модели

### Инициализиране на моделите

За обучението са използвани пет различни модела за машинно обучение:

* **Naive Bayes** — GaussianNB;
* **KNN** — KNeighborsClassifier;
* **Логистична регресия** — LogisticRegression;
* **Случайни гори** — RandomForestClassifier;
* **MLP Класификатор** — MLPClassifier.

classifiers = {

'Naive Bayes': GaussianNB(),

'KNN': KNeighborsClassifier(),

'Logistic Regression': LogisticRegression(verbose=True),

'Random Forest': RandomForestClassifier(verbose=True),

'MLP Classifier': MLPClassifier(verbose=True)

}

### Определяне на хиперпараметри за GridSearchCV

За всеки модел са зададени различни хиперпараметри, които да бъдат оптимизирани чрез GridSearchCV:

param\_grids = {

'Naive Bayes': {

'var\_smoothing': [1e-9, 1e-8, 1e-7]

},

'KNN': {

'n\_neighbors': [3, 5, 7, 9, 11],

'p': [1, 2, 3]

},

'Logistic Regression': {

'C': [0.1, 1, 10]

},

'Random Forest': {

'n\_estimators': [100, 200, 300],

'max\_depth': [None, 10, 20]

},

'MLP Classifier': {

'hidden\_layer\_sizes': [(50,), (100,), (50, 50)],

'activation': ['relu', 'tanh'],

'solver': ['adam'],

'max\_iter': [500, 1000],

'learning\_rate\_init': [0.001, 0.0005],

'early\_stopping': [True]

}

}

### Обучение на моделите

Обучението на моделите се извършва чрез GridSearchCV, използвайки 5-кратна кръстосана проверка (5-fold cross-validation). Най-добрите параметри и резултати за всеки модел се съхраняват.

trained\_models = {}

training\_times = {} # Dictionary to store training times

start\_time = datetime.now()

print(f"Training started at: {start\_time.strftime('%H:%M:%S')}")

for name, clf in classifiers.items():

model\_start\_time = datetime.now()

print(f"{name} training started at: {model\_start\_time.strftime('%H:%M:%S')}")

# Train the model

grid\_search = GridSearchCV(estimator=clf, param\_grid=param\_grids[name], cv=5, n\_jobs=-1, scoring='accuracy')

grid\_search.fit(X\_train\_scaled, Y\_train)

trained\_models[name] = grid\_search

model\_end\_time = datetime.now()

print(f"{name} training finished at: {model\_end\_time.strftime('%H:%M:%S')}")

# Calculate and store training time

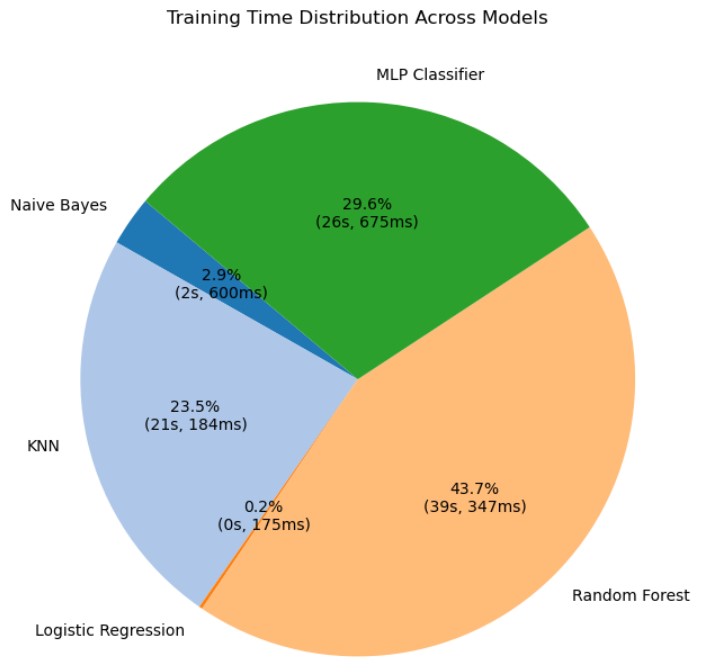
duration = (model\_end\_time - model\_start\_time).total\_seconds()

training\_times[name] = duration

end\_time = datetime.now()

print(f"Training finished at: {end\_time.strftime('%H:%M:%S')}")

В кода е включено и допълнитено проследяване на времето за трениране на моделите, което след това се визуализира на кръгова диаграма:



Фигура 4. Кръгова диаграма, отразяваша времето на обучение за моделите

### Резултати от обучението

Резултатите за всеки модел се извеждат, включително най-добрите параметри, най-добрия резултат от кръстосаната проверка и точността върху тестовите данни.

for name, grid\_search in trained\_models.items():

print(f"{name}:")

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Score:", grid\_search.best\_score\_)

best\_model = grid\_search.best\_estimator\_

Y\_pred = best\_model.predict(X\_test\_scaled)

test\_accuracy = accuracy\_score(Y\_test, Y\_pred)

print("Test Accuracy:", test\_accuracy)

print("Confusion Matrix:\n", confusion\_matrix(Y\_test, Y\_pred))

print("Classification Report:\n", classification\_report(Y\_test, Y\_pred))

print()

### Обобщение на най-добрите модели

Създава се обобщение на най-добрите модели и техните хиперпараметри:

print("Summary of Best Models:")

for name, grid\_search in trained\_models.items():

print(f"{name}:")

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Score (CV):", grid\_search.best\_score\_)

print()

### Визуализация на резултатите

Резултатите от обучението се визуализират чрез диаграми за всяка матрица на объркване, както и чрез извеждане на точността, прецизността, пълнотата и F1-оценката за всеки модел.

for name, grid\_search in trained\_models.items():

Y\_pred = grid\_search.predict(X\_test\_scaled)

print("Recap for the " + name +" model:\n", classification\_report(Y\_test, Y\_pred) )

conf\_matrix = confusion\_matrix(Y\_test, Y\_pred);

report = classification\_report(Y\_test, Y\_pred, output\_dict=True);

accuracy = accuracy\_score(Y\_test, Y\_pred)

precision = report['weighted avg']['precision']

recall = report['weighted avg']['recall']

f1\_score = report['weighted avg']['f1-score']

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1 Score: {f1\_score:.2f}")

print(f"Accuracy: {accuracy:.2f}")

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap='Blues')

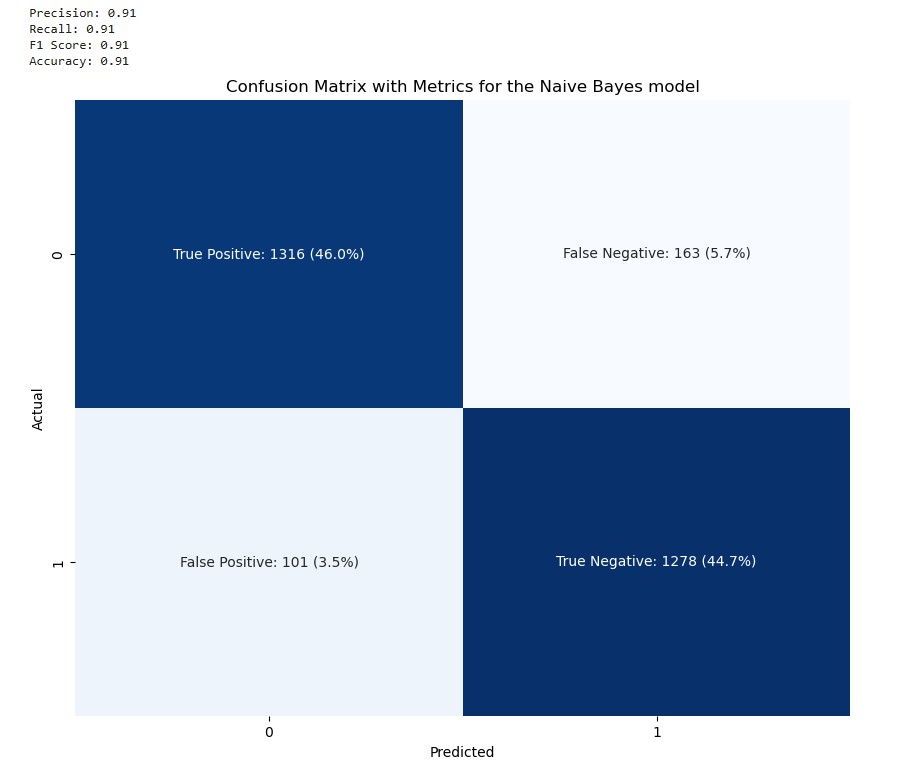
plt.xlabel('Predicted')

plt.ylabel('Actual')

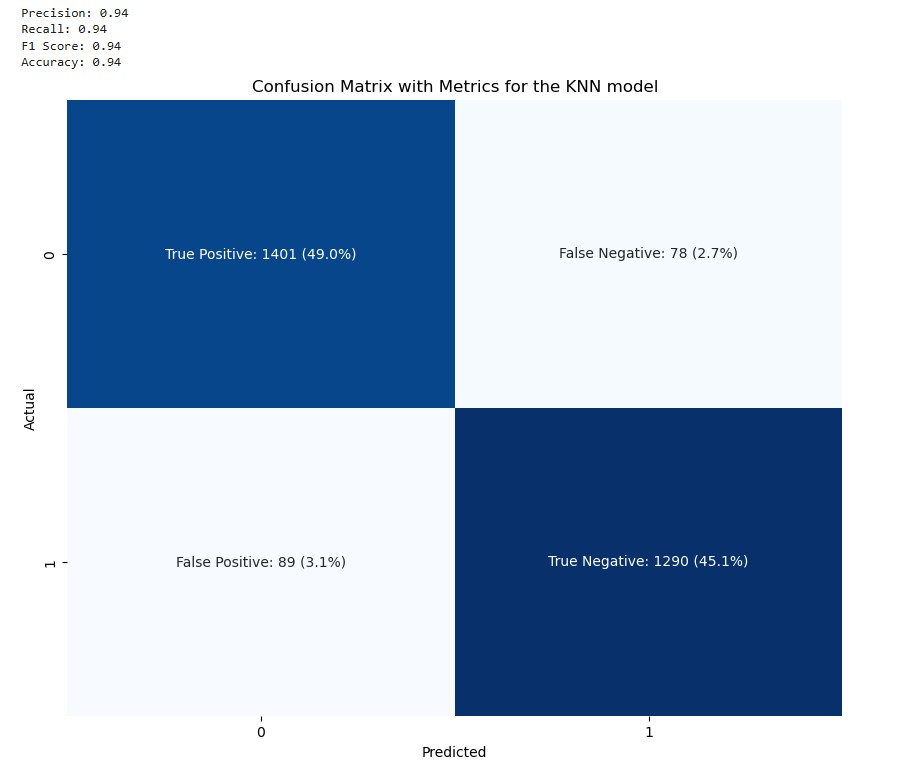
plt.title('Confusion matrix for the ' + name + " model")

plt.show()

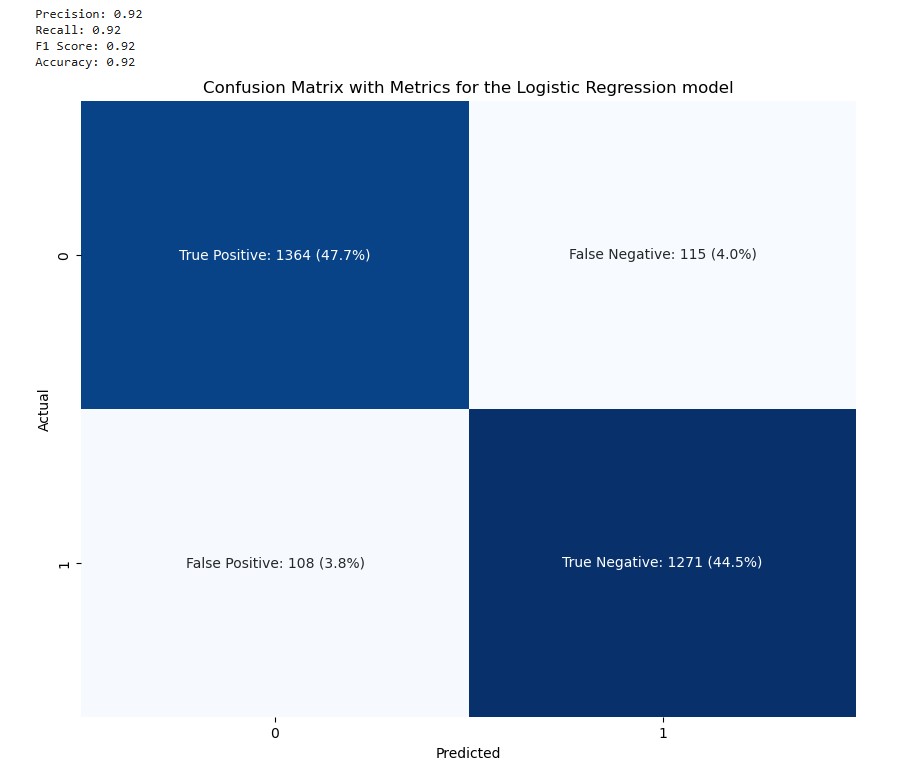
print()



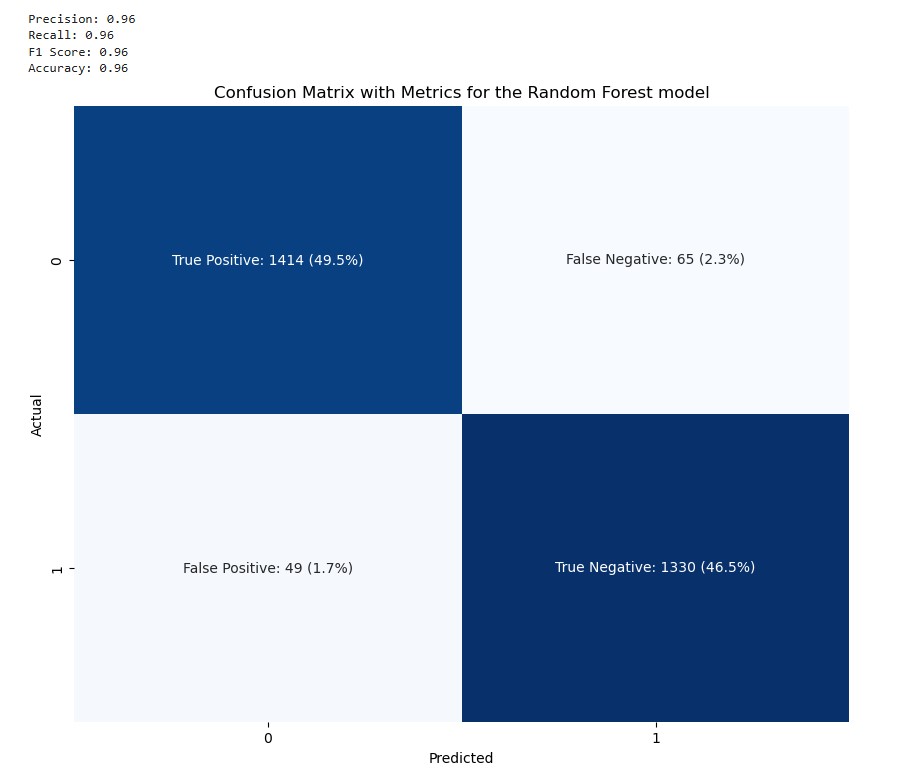
Фигура 5. Confusion matrix за Naïve Bayes модела



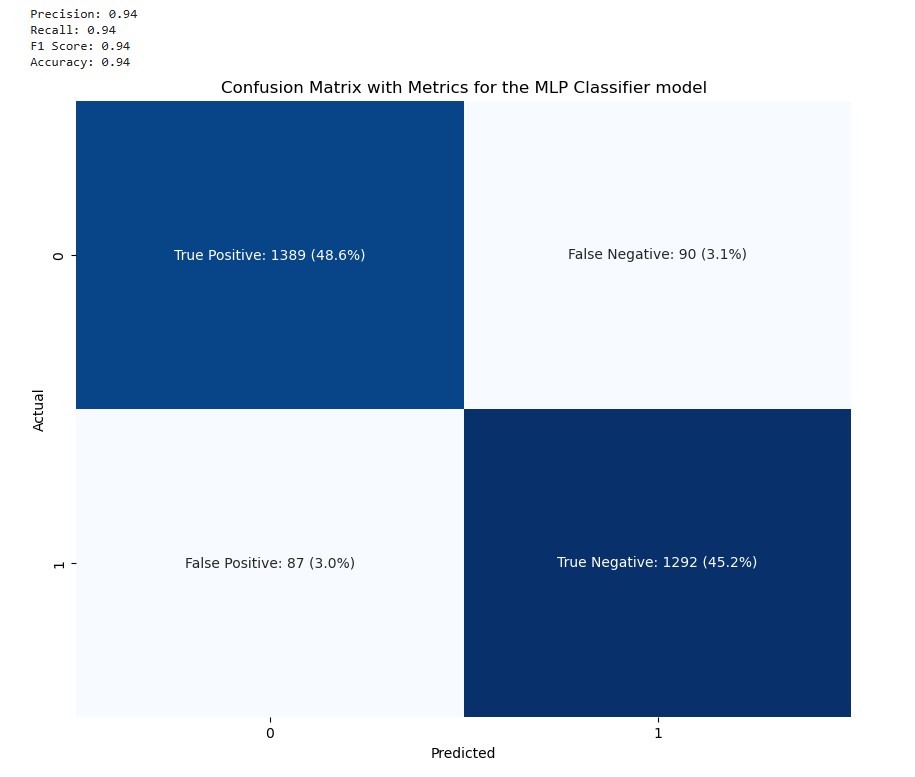
Фигура 6. Confusion matrix за KNN модела



Фигура 7. Confusion matrix за Logistic Regression модела



Фигура 8. Confusion matrix за Random Forest модела

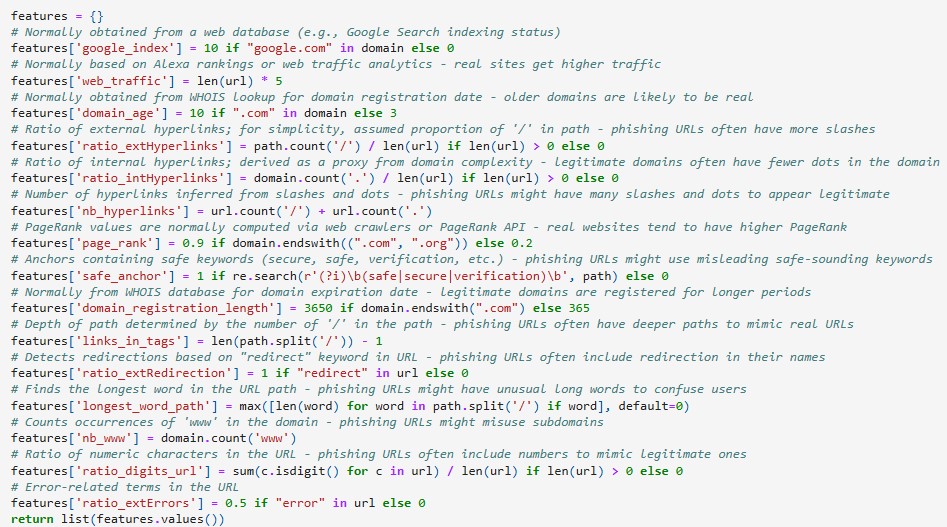


Фигура 9. Confusion matrix за Neural Network модела

### Тестване на модели с реални URL адреси

Моделите се тестват върху реални и фишинг URL адреси, за да се оцени тяхната производителност извън обхвата на учебния набор от данни.

За целта се използва примерна тестова функция за извличане на характеристики от данни в текстови формат, какъвто е URL-а. Тази функция е много неточна и приложението и е само тестово. Описани са значенията на всяка характеристика и откъде би се извличала тя в реални условия:



Фигура 10. Избраните при обучението характеристики и тяхното значение

Това е и кодът, който проверява оценката на моделите за представените тестови адреси:

real\_features\_scaled = scaler.transform(real\_features)

phishing\_features\_scaled = scaler.transform(phishing\_features)

for name, grid\_search in trained\_models.items():

real\_addresses\_pred = grid\_search.predict(real\_features\_scaled)

phishing\_addresses\_pred = grid\_search.predict(phishing\_features\_scaled)

print(f"Predictions for {name} model:")

print("Real addresses:", real\_addresses\_pred)

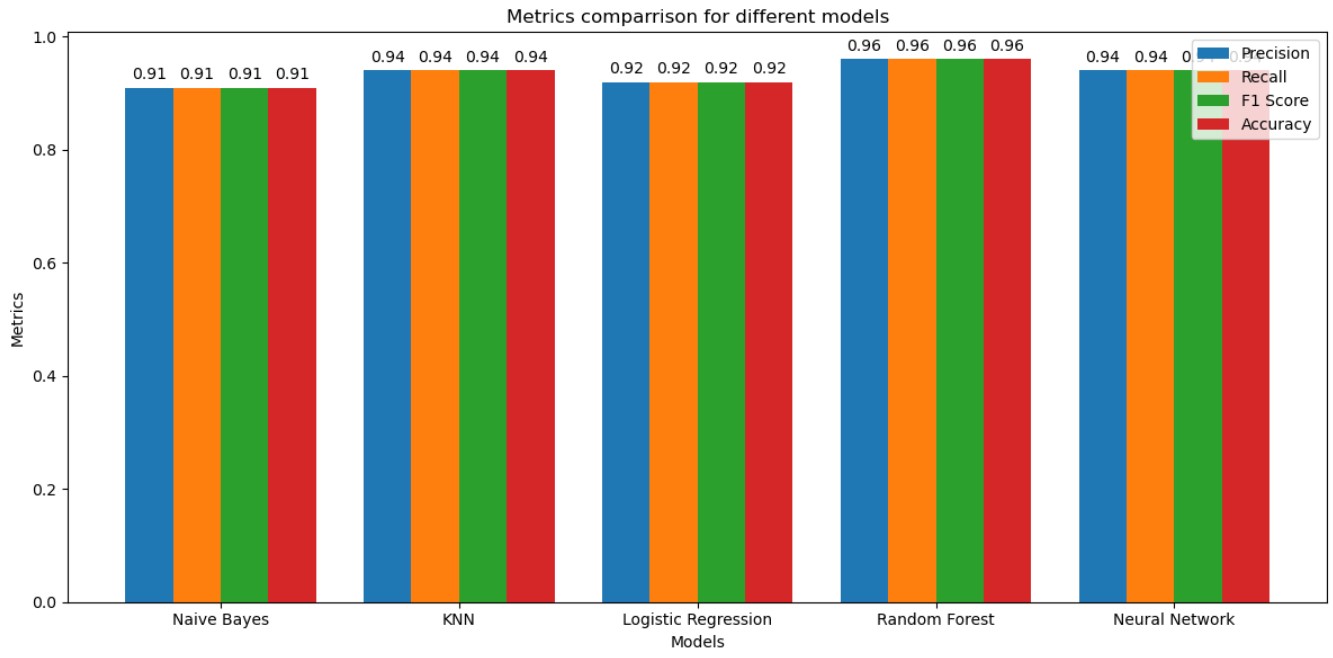
print("Phishing addresses:", phishing\_addresses\_pred)

print()

Адресите, които би трябвало да бъдат оценени като истински, са:  
["https://www.youtube.com/", "https://www.microsoft.com/en-us", "https://www.amazon.com", "https://www.wikipedia.org", "https://www.paypal.com"]

Адресите, които би трябвало да бъдат оценени като фишинг, са:  
 [ "http://secure-go0gle-login.verification.com, "https://microsoft-update.support-login.info", "http://amz-login-services.xyz", "https://paypal-secure-check.xyz", "http://account-verification-paypall.net"]

Заключение и изводи



*Фигура 11. Сравнение на алгоритмите в моделите за Phishing*

Резултатите от моделите, обучени да разпознават Phishing, са доста сходни. Всеки един алгоритъм може да се счете за надежден, тъй като всички надминават 90% точност в предсказанията. Това твърдение все пак трябва да се провери повторно и върху реални интернет адреси за да се приеме за напълно достоверно.

Тъй като датасетът е балансиран (еднакъв брой записи на легитимни и фишинг адреси), метриките като precission, recall, F1 score е нормално да клонят много близо или напълно да се препокриват със стойността accuracy. Това, че false positive/ false negative резултатите са много малко също допринася за смаляване на разликата между отделните характеристики и в крайна сметка за всеки модел тези метрики са отразени от едно и също число.

Съществена разлика в моделите може да бъде проследена във времето им за обучение, което би скалирало при по-големи набори трениращи данни. От Фиг.16 може да проследим значително по-дългото време за обучение на Random Forest, Neural network и KNN алгоритмите. Според графиката със сравнените крайни резултати, това са и 3-те малко по-точни от останалите алгоритми. При големи набори от трениращи данни или увеличение на броя следени характеристики е силно вероятно алгоритъма KNN да отпадне поради непредвидеността му за скалируемост. Naïve Bayes и Logistic Regression пък имат значително опростена сложност, която би се отразила негативно при евентуален по-сложен датасет с повече нелинейни или корелиращи помежду си характеристики. Това би изкарало хардуерно-изискващите алгоритми Random Forest и Neural Network като най-надеждни при нужда на скалиране на обема или сложността на трениращите данни за постигане на максимално точен модел.

Използвайки текущия датасет, разликата между резултатите от различните модели е почти незначителна при допускането че данните са предварително почистени, балансирани и характеристиките са подбрани до разумно малък брой.