

Predykcja ataku phishingowego w wiadomości e-mail za pomocą *nadzorowanego nauczania maszynowego*

Dataset:

- **Phishing Email Curated Datasets**
 - <https://zenodo.org/records/8339691>

Pobieranie niezbędnych modułów

- pandas - praca z *Data Framami*
- numpy - obliczenia
- matplotlib.pyplot - wizualizacja
- sklearn - wszelakie narzędzia do *Machine Learningu*

```
In [1]: import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import KFold, train_test_split, cross_val_score, GridS
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
import sklearn.metrics as skm

import import_ipynb
```

Wczytanie uprzednio przygotowanego Data Framu

```
In [2]: learning_set = pd.read_csv('ML_DataFrame.csv')
print(learning_set.head())
```

	Unnamed: 0	label	urls_count	protocol	contains_ip	url_length	\
0	0	1.0	1.0	0.0	0.0	21.0	
1	1	1.0	1.0	0.0	0.0	25.0	
2	2	1.0	3.0	0.0	0.0	110.0	
3	3	0.0	3.0	0.0	0.0	22.0	
4	4	1.0	1.0	0.0	0.0	136.0	

	TLD_alpha	subdomain_level	slash_count	dots_count	hyphens_count	\
0	1.0	0.0	3.0	1.0	0.0	
1	1.0	1.0	2.0	2.0	0.0	
2	1.0	1.0	6.0	5.0	0.0	
3	0.0	1.0	2.0	2.0	0.0	
4	1.0	2.0	4.0	4.0	2.0	

	has_non_latin
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

```
In [3]: print(learning_set.isna().sum())
learning_set.dropna(inplace=True)
```

```
Unnamed: 0      0
label           0
urls_count      0
protocol        0
contains_ip     0
url_length      0
TLD_alpha       0
subdomain_level 84
slash_count     0
dots_count      0
hyphens_count   0
has_non_latin   0
dtype: int64
```

Wybieranie X i y

X:

- urls_count
- protocol
- contains_ip
- url_length
- TLD_alpha
- subdomain_level
- slash_count
- dots_count
- hyphens_count
- has_non_latin

y:

- label

```
In [4]: X = learning_set.loc[:, 'urls_count':'has_non_latin'].values
y = learning_set.loc[:, 'label'].values
print(X.shape, y.shape)
```

(41632, 10) (41632,)

Rozdzielanie X, y na treningowe i testowe zestawy

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Normalizujemy wartość X-ów

```
In [6]: scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
In [7]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

(33305, 10) (8327, 10) (33305,) (8327,)

Szukanie najlepszych parametrów dla *KNeighborsClassifier*

```
In [8]: kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'n_neighbors': np.arange(1, 15, 1),
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid=params, cv=kf)
knn_cv.fit(X_train, y_train)

print('Najlepsze parametry dla KNeighborsClassifier:')
for p, val in knn_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', knn_cv.best_score_)

best_knn = knn_cv.best_estimator_
test_accuracy = best_knn.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

Najlepsze parametry dla KNeighborsClassifier:
n_neighbors: 13
p: 1
weights: distance
Uzyskana precyzja: 0.8729022068366331
Precyzja zestawu testowego: 0.8734238020895881

Szukanie najlepszych parametrów dla *LogisticRegression*

```
In [ ]: from sklearn.linear_model import LogisticRegression
kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10]
}
logreg_forest = LogisticRegression()
logreg_cv = GridSearchCV(logreg_forest, param_grid=params, cv=kf)
logreg_cv.fit(X_train, y_train)

print('Najlepsze parametry dla LogisticRegression:')
for p, val in logreg_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', logreg_cv.best_score_)

best_logreg = logreg_cv.best_estimator_
test_accuracy = best_logreg.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

Szukanie najlepszych parametrów dla *DecisionTreeClassifier*

```
In [10]: kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
tree = DecisionTreeClassifier()
tree_cv = GridSearchCV(tree, param_grid=params, cv=kf)
tree_cv.fit(X_train, y_train)

print('Najlepsze parametry dla DecisionTreeClassifier:')
for p, val in tree_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', tree_cv.best_score_)

best_tree = tree_cv.best_estimator_
test_accuracy = best_tree.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

Najlepsze parametry dla DecisionTreeClassifier:
criterion: gini
max_depth: None
min_samples_leaf: 1
min_samples_split: 2
Uzyskana precyzja: 0.8711907602071537
Precyzja zestawu testowego: 0.8698210640086466

Szukanie najlepszych parametrów dla *RandomForestClassifier*

```
In [11]: from sklearn.ensemble import RandomForestClassifier

kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
rand_forest = RandomForestClassifier()
rand_forest_cv = GridSearchCV(rand_forest, param_grid=params, cv=kf)
rand_forest_cv.fit(X_train, y_train)

print('Najlepsze parametry dla RandomForestClassifier:')
for p, val in rand_forest_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', rand_forest_cv.best_score_)

best_rand_forest = rand_forest_cv.best_estimator_
test_accuracy = best_rand_forest.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

Najlepsze parametry dla RandomForestClassifier:
max_depth: None
min_samples_leaf: 1
min_samples_split: 10
n_estimators: 200
Uzyskana precyzja: 0.8757846731617223
Precyzja zestawu testowego: 0.8741443497057764

Szukanie najlepszych parametrów dla *SVM Classifier*

```
In [12]: from sklearn.svm import SVC

kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto', 0.1, 1]
}
svm = SVC()
svm_cv = GridSearchCV(svm, param_grid=params, cv=kf)
svm_cv.fit(X_train, y_train)

print('Najlepsze parametry dla SVM Classifier:')
for p, val in svm_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', svm_cv.best_score_)

best_svm = svm_cv.best_estimator_
test_accuracy = best_svm.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

Najlepsze parametry dla SVM Classifier:
C: 10
gamma: 1
kernel: rbf
Uzyskana precyzja: 0.8508033776886236
Precyzja zestawu testowego: 0.8576918458028101

Ewaluacja modeli

```
In [13]: models = {
    'KNeighborsClassifier': KNeighborsClassifier(n_neighbors=13, p=1, weights='dist'),
    'LogisticRegression': LogisticRegression(C=10, penalty='l2'),
    'DecisionTreeClassifier': DecisionTreeClassifier(criterion='gini', max_depth=None),
    'RandomForestClassifier': RandomForestClassifier(max_depth=None, min_samples_le
    'SVM Classifier': SVC(C=10, gamma=1, kernel='rbf')
}

results = []
for model in models.values():
    kf = KFold(n_splits=6, shuffle=True, random_state=10)
    cv_results = cross_val_score(model, X_train, y_train, cv=kf)
    results.append(cv_results)
```

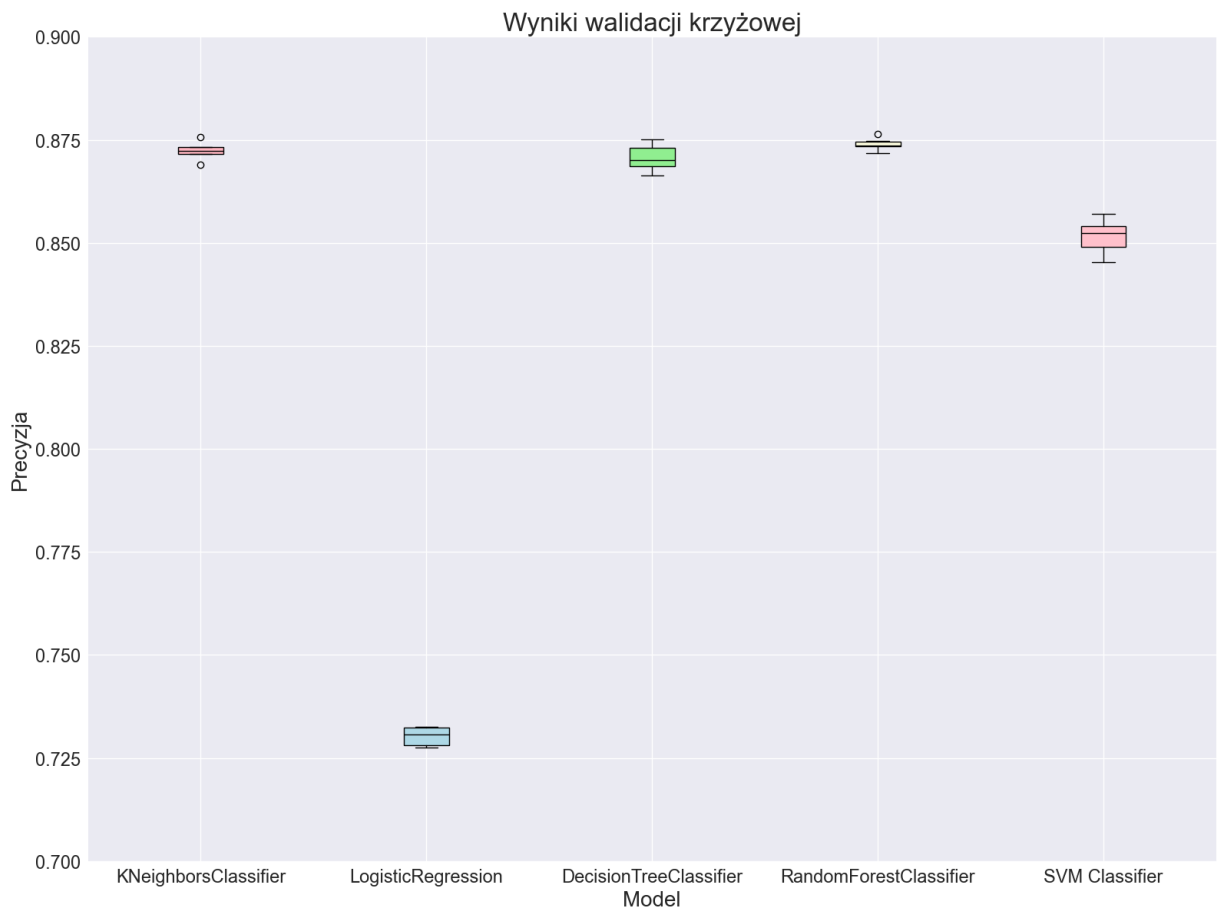
```
In [14]: plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots(figsize=(16, 12))

boxplot = ax.boxplot(
    results,
    labels=models.keys(),
    patch_artist=True,
    boxprops=dict(edgecolor='black'),
    whiskerprops=dict(linewidth=1),
    capprops=dict(linewidth=1),
    medianprops=dict(color='black', linewidth=1),
    widths=0.2,
)

box_colors = ['#FFB6C1', '#ADD8E6', '#90EE90', '#FFFFFFE0', '#FFC0CB']
for box, color in zip(boxplot['boxes'], box_colors):
    box.set(facecolor=color)

ax.set_title('Wyniki walidacji krzyżowej', fontsize=24)
ax.tick_params(axis='both', labelsize=17)
ax.set_xlabel('Model', fontsize=20)
ax.set_ylabel('Precyzja', fontsize=20)
ax.set_ylim(0.70, 0.9)
ax.yaxis.grid(True)

plt.tight_layout()
plt.show()
```



Nauczanie najskuteczniejszego modelu

Najlepszy okazał się ***RandomForestClassifier***

```
In [15]: import sklearn.metrics as skm
random_forest = RandomForestClassifier(max_depth=None,
                                     min_samples_leaf=1,
                                     min_samples_split=10,
                                     n_estimators=50)

rand_forest.fit(X_train, y_train)
y_pred = rand_forest.predict(X_test)
confusion_matrix = skm.confusion_matrix(y_test, y_pred)

target_names = ['safe', 'phishing']
class_report = skm.classification_report(y_test, y_pred, target_names=target_names)
print(class_report)
```

	precision	recall	f1-score	support
safe	0.84	0.83	0.83	3175
phishing	0.89	0.90	0.90	5152
accuracy			0.87	8327
macro avg	0.87	0.86	0.87	8327
weighted avg	0.87	0.87	0.87	8327

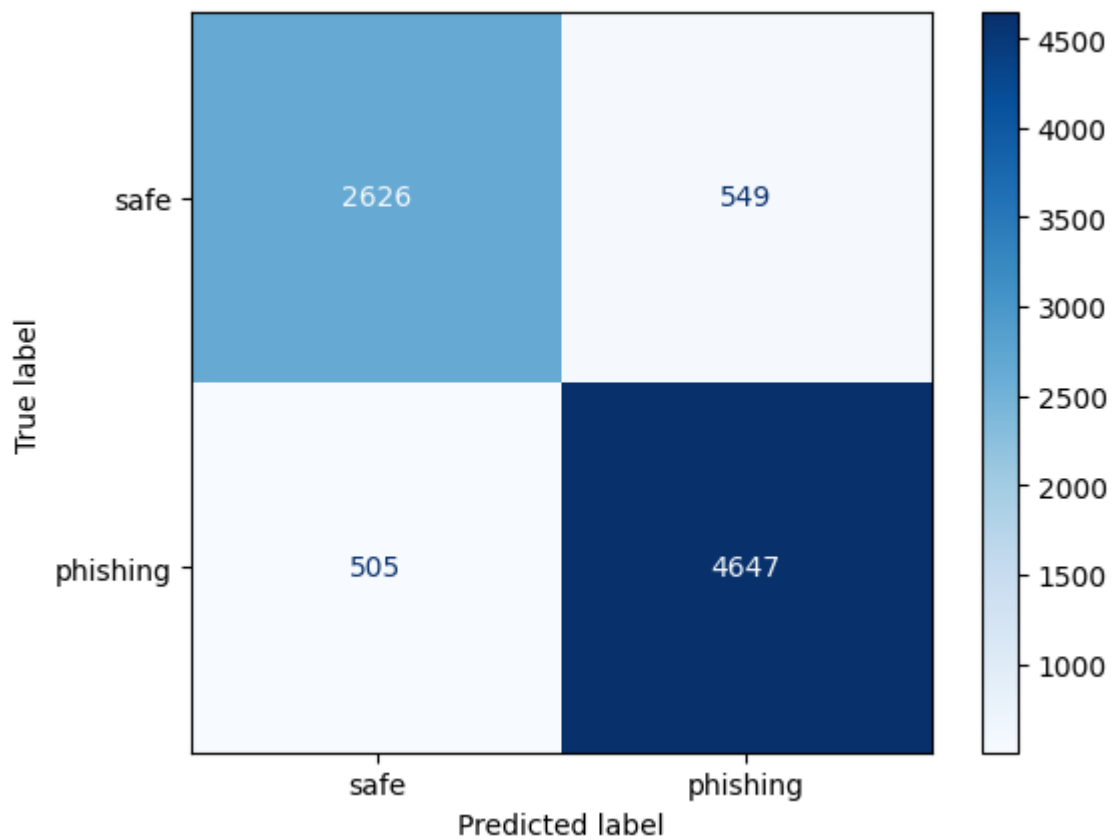
Skuteczność modelu

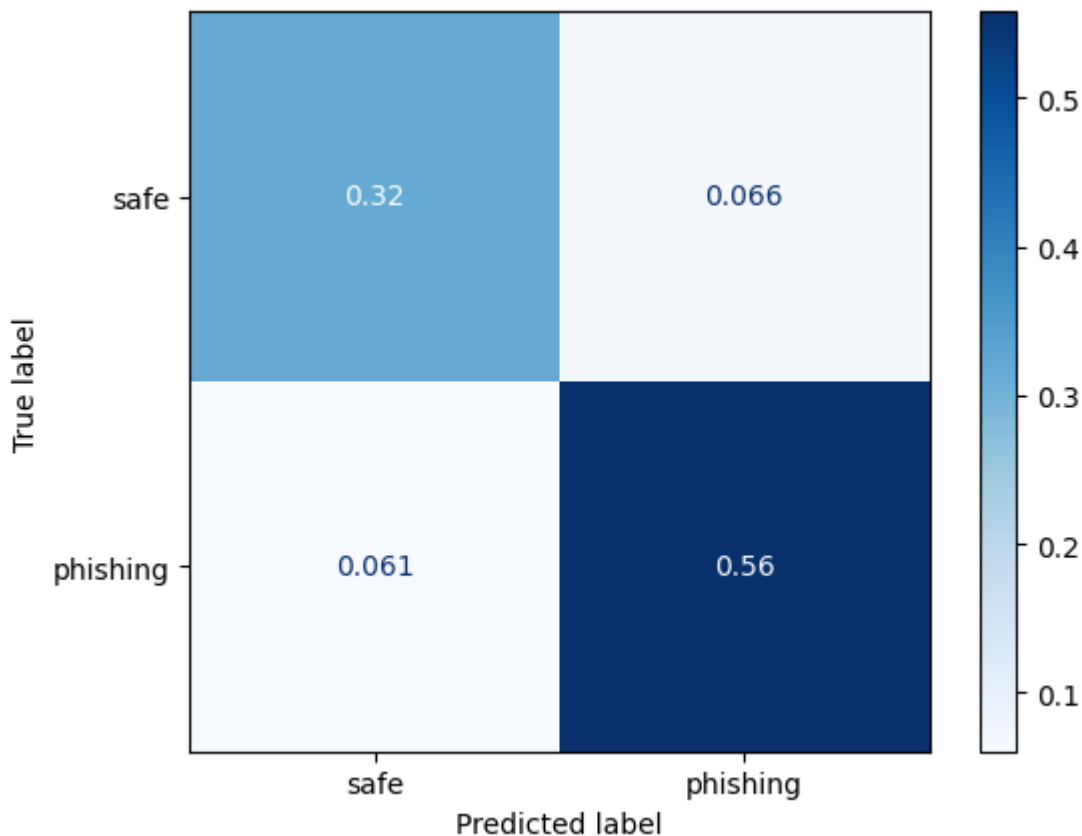
```
In [16]: plt.style.use('default')
cm_display = skm.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix,
    display_labels=['safe', 'phishing'])

cm_display.plot(cmap='Blues')
plt.show()

# Wartości względne
cm_display = skm.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix/np.sum(confusion_matrix),
    display_labels=['safe', 'phishing'])

cm_display.plot(cmap='Blues')
plt.show()
```





Test na przykładzie z życia

```
In [17]: mail_example = {
'sender': 'Marcin Sawiński <Marcin.Sawinski@ue.poznan.pl>',
'subject': 'Projekty zaliczeniowe z PSI',
'body': '''
Drodzy Studenci,

Chciałbym Was prosić o zapisanie w arkuszu tematów projektów zaliczeniowych
https://uniekonpoznan.sharepoint.com/:x:/s/AI\_2023\_2024/EQ5avb31cRpAliTAMGt

Pozdrawiam,

Marcin Sawinski
--
Wiadomość wysłana przez system USOS.
Łączna liczba adresatów tej wiadomości: 117

Nadawcą korespondencji i jednocześnie administratorem Państwa danych osobowych
al. Niepodległości 10, 61-875 Poznań, z którym można skontaktować się listo
Państwa dane osobowe przetwarzane będą w celu prowadzenia niniejszej koresp
Państwu prawo dostępu do danych, ich sprostowania, ograniczenia przetwarzan
przetwarzania oraz wniesienia skargi do Prezesa Urzędu Ochrony Danych Osobo
danych osobowych dostępne są w Polityce prywatności.

...
'''
}
#TODO: DOKOŃCZYĆ
```

Rezultat projektu

Przy pomocy **RandomForestClassifier** udało się wytrenować model z wynikami:

- (accuracy) ~
- (recall) ~
- (F1) ~

Problemy podczas projektu

- dostęp do API
 - brak sprawdzania domen w blacklistach
 - brak sprawdzania adresów e-mail w blacklistach
 - słabe/brak informacji o szyfrowaniu SSL
 - brak sprawdzania wieku domeny
- Dane zbierane w latach 2008-2022
- Człowiek minimalnie obeznany w internecie poradziłby sobie z klasyfikowaniem ataków phishingowych z datasetu.