

НТУУ "КПІ ім Ігоря Сікорського"

Фізико-технічний інститут

Методи машинного навчання

Лабораторна робота 6

Робота з відкритим програмним кодом

ФБ-14 Разумний Ілля

Хід виконання роботи:

1. Знайти наукову статтю англійською мовою не старішу 2016 року, яка має відповідний їй код на GitHub у відкритому доступі. Темати статті може бути Object Detection/Tracking/Classification, Segmentation та інші області машинного чи глибокого навчання

Phishing Detection Using Machine Learning Techniques

Article	GitHub
https://arxiv.org/pdf/2009.11116v1	https://github.com/fafal-abnir/phishing_detection

Стаття датується **20 вересня 2020**

Авторами є 3 науковця з **Computer Engineering** в університетах **Тегерана, Іран**

У статті порівнюються методи машинного навчання для **прогнозування фішингових сайтів** (усього їх 12(9 унікальних)). Ціль статті полягає у дослідженні та оцінці ефективності цих методів

У статті згадуються методи вчинення фішингу, такі як маніпуляція посиланнями, уникнення фільтрів, підробка сайтів, приховані перенаправлення та соціальна інженерія

Розглядаються методи запобігання фішинговим атакам, зокрема навчання користувачів та **програмне виявлення**, яке включає чорні списки, візуальні методи порівняння та методи виявлення фішингових сайтів на **основі машинного навчання** (про них буде)

Використовували набір даних, який був добре досліджений та оцінений іншими вченими, і супроводжується документом з описом даних і стратегіями їх генерації

Для оновлення набору даних реалізували код, який витягує характеристики нових фішингових сайтів із сайту PhishTank. Набір даних містить близько 11,000 зразків сайтів, з яких 10% використовувались для тестування. Кожен сайт позначений як легітимний або фішинговий

Опис колонок датасета:

Having ip address	Айпішник замість домена
URL Length	Довжина адреси. Довгими URL приховують підозрілі частини
Shortening Service	Скорочене посилання на оригінальну адресу сайту
Having at symbol	Наявність @ змушує браузер ігнорувати все до символу
Double slash redirection	Наявність // вказує на редирект на інший сайт
Prefix Suffix	Різдження – змушує вірити у легітимність сайту, як от http://www.Confirmed-paypal.com
Having Sub Domain	Чи є субдомен в посиланні
SSL State	Чи є шифрування
Domain Registration Length	Фактично, фішингові сайти живуть мало
Favicon	Іконка сайту. Якщо вона підгружається з домену, відмінного від сайту, то великі шанси на фішинговий сайт
Using Non-Standard Port	Краще відкривати порти, коли потрібно. І використовувати мережі засоби
HTTPS token	Містять обманливі токени, як от http://https-www-mellat-phish.ir
Request URL	Перевіряє чи картинка, відео, звуки завантажені з іншого домену
URL of Anchor	Чи є <a> HTML тег. Працює як і request url
Links in tags	Легітимні сайти використовують теги <Meta>, <Script>, <Link> для управління контентом і ресурсами
Server form handler	Чи доменне ім'я в SFH різне від імені сторінки
Submitting Information to e-mail	Фішер редиректить інформацію про користувачів до свого емейла
Abnormal URL	Витягнуто з бази WHOIS. Identity є частиною URL для норм сайтів
Website Redirect Count	Чи редирект більше 4 раз
Status Bar Customization	Використання JavaScript для фейкового посилання в статус барі
Disabling Right Click	Викл ПКМ. Працює як onMouseOver щоб приховати посилання
Using Pop-up window	Чи є вилізаючі вікна
IFrame	Тег, який дозволяє показувати частину додаткової сторінки
Age of domain	Чи вік домену менше місяця

DNS Record	Чи є запис DNS
Web Traffic	Вимірює популярність сайта залежно від цифри відвідувачів
Page Rank	Вимірює важливість сторінки в Інтернеті від 0 до 1
Google Index	Чи є сайт в індексі Google
Links Pointing To Page	Кількість посилань на сторінку
Statistical Report	Чи належить айпі адреса до топ фішингових

2. Зробити короткий опис результатів, яким присвячена стаття, який алгоритм/ідею вона пропонує, як це досягається, в чому описаний алгоритм/ідея краще за інші алгоритми/ідеї (в чому полягає новизна)

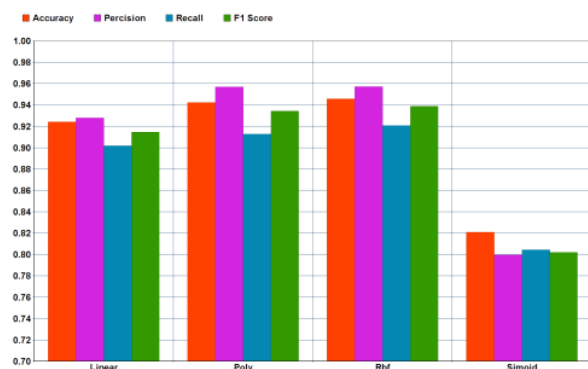
Основною таблицею є порівняння показників продуктивності моделей

TABLE III
CLASSIFICATION RESULTS FOR DIFFERENT METHODS

classifier	train time (s)	test time(s)	accuracy	recall	precision	F1 score
logistic regression	0.080971	0.006414	0.926550	0.943968	0.925700	0.934704
decision tree	0.021452	0.003737	0.965988	0.971414	0.967681	0.969531
random forest	0.436126	0.021941	0.972682	0.981484	0.969852	0.975622
ada booster	0.336519	0.016766	0.936953	0.954362	0.933943	0.944032
KNN	0.112972	0.353562	0.952780	0.962968	0.952783	0.957827
neural network	9.088517	0.006925	0.969879	0.978723	0.967605	0.973112
SVM_linear	1.647538	0.053979	0.927726	0.945592	0.926268	0.935779
SVM_poly	1.048257	0.074207	0.949254	0.968816	0.941779	0.955083
SVM_rbf	1.341540	0.103329	0.952149	0.968815	0.946580	0.957543
SVM_sigmoid	1.344607	0.109696	0.827498	0.846515	0.844311	0.845305
gradient boosting	0.891888	0.005298	0.948621	0.962481	0.946234	0.954260
XGBoost	0.506072	0.006237	0.983235	0.981047	0.987235	0.976802

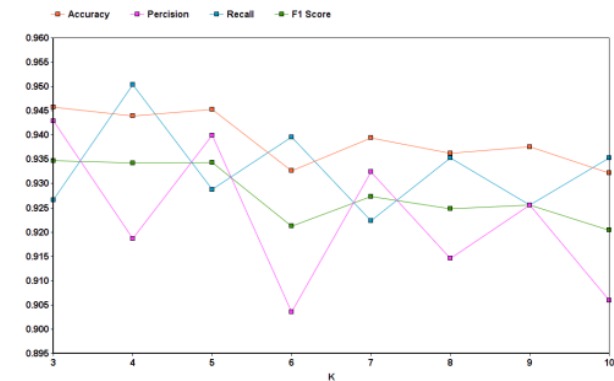
Власного розробленого алгоритма не було, натомість було використано доволі великий список популярних моделей МН та їх порівняння, що дозволяє знайти найефективніший метод виявлення фішингових сайтів

Для SVM використовували декілька ядер

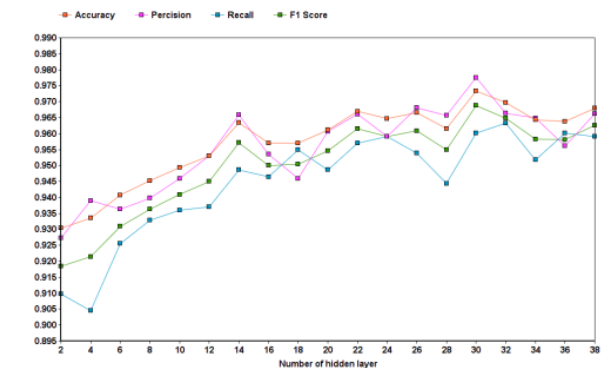


Random Forest показав високу точність, стійкість до шуму та викидів, швидкість і простоту реалізації, але має багато гіперпараметрів для налаштування. Основний недолік — випадковість у тренувальних і тестових даних, що не підходить для всіх наборів даних

У KNN найкраща ефективність досягається при $k = 5$. Мала кількість сусідів дає високу гнучкість, але високу дисперсію, а велика — гладку межу рішень з меншим варіантом, але більшим упередженням



Час навчання **Персептрона** був більший, а F1-оцінка XGBoost — кращою через малий розмір даних. Нейронна мережа не пояснює своїх прогнозів. Найкраща продуктивність була при 30 прихованих шарах і 500 епохах з раннім зупиненням



XGBoost швидкий і зменшує дисперсію через регуляризацію. Однак його складніше налаштувати та візуалізувати, і для досягнення високих результатів потрібно більше часу

Згідно з результатами, **вищу продуктивність** за часом обчислень і точністю показали ансамблеві методи, зокрема **Random Forest і XGBoost**

Основна ідея **ансамблевих** алгоритмів полягає в **комбінуванні кількох слабких учнів в один сильний**, що є головною перевагою таких методів у задачах класифікації

Не можна гарантувати, що комбінування кількох класифікаторів завжди буде ефективніше за найкращий окремих. Це дослідження стимулює подальші роботи щодо додавання нових ознак до набору даних для покращення точності моделей і комбінування машинного навчання з іншими методами

3. Розібратися у коді алгоритму, запустити його і продемонструвати запущене демо з *GitHub*

Краще всього пролистати 2 ноутбуки:

- **dataset_description.ipynb** — опис даних, візуалізація
- **All_Classifair.ipynb** — безпосередньо запуск моделей та показників

У прилеглих файлах є функція оцінки URL, але я не знайшов ті URL, над якими здійснювався збір. У статті зазначався PhishTank, і прилеглий код **feature_extraction.py** можна застосувати для цих URL (але треба скрапером їх спочатку отримати):

Verify A Phish

Showing unverified and online submissions

[See all submissions in the phish archive](#)

ID	Phish URL	Submitted	Valid?	Online?
8857409	https://ftmy9hw2.s3.us-west-1.amazonaws.com/R71XCZZM.html... added on Nov 18th 2024 12:30 PM	by n0x6fb0x6fdy	Unknown	ONLINE
8857408	https://gkmja8.s3.us-west-1.amazonaws.com/4VB1AO7V.html... added on Nov 18th 2024 12:30 PM	by n0x6fb0x6fdy	Unknown	ONLINE
8857404	https://asyconsultoria.com/.wb/index.html#cert@cert.u... added on Nov 18th 2024 12:30 PM	by socteam	Unknown	ONLINE
8857403	https://www.don-resurs.ru added on Nov 18th 2024 12:29 PM	by Felix0101	Unknown	ONLINE
8857402	https://claimgetfitmining.pages.dev added on Nov 18th 2024 12:29 PM	by Felix0101	Unknown	ONLINE
8857400	https://ethlayer.net added on Nov 18th 2024 12:28 PM	by tuanphuong	Unknown	ONLINE

Ось приклад для сайта з оцінками з курсу реверса:

```
print(generate_data_set("bit.ly/kpi_re2023"))  
[1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, -1, 1, -1, 0, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, 1, 1, 1]
```

Ось ці результати були занесені в **dataset.csv**

Короткий огляд ноутбуків:

- dataset_description.ipynb

[7] df

	having_IPhaving_IP_Address	URLURL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_State	Dom
index									
1	-1	1	1	1	-1	-1	-1	-1	
2	1	1	1	1	1	-1	0	1	
3	1	0	1	1	1	-1	-1	-1	
4	1	0	1	1	1	-1	-1	-1	
5	1	0	-1	1	1	-1	1	1	
...	
11051	1	-1	1	-1	1	1	1	1	
11052	-1	1	1	-1	-1	-1	1	-1	
11053	1	-1	1	1	1	-1	1	-1	
11054	-1	-1	1	1	1	-1	-1	-1	
11055	-1	-1	1	1	1	-1	-1	-1	

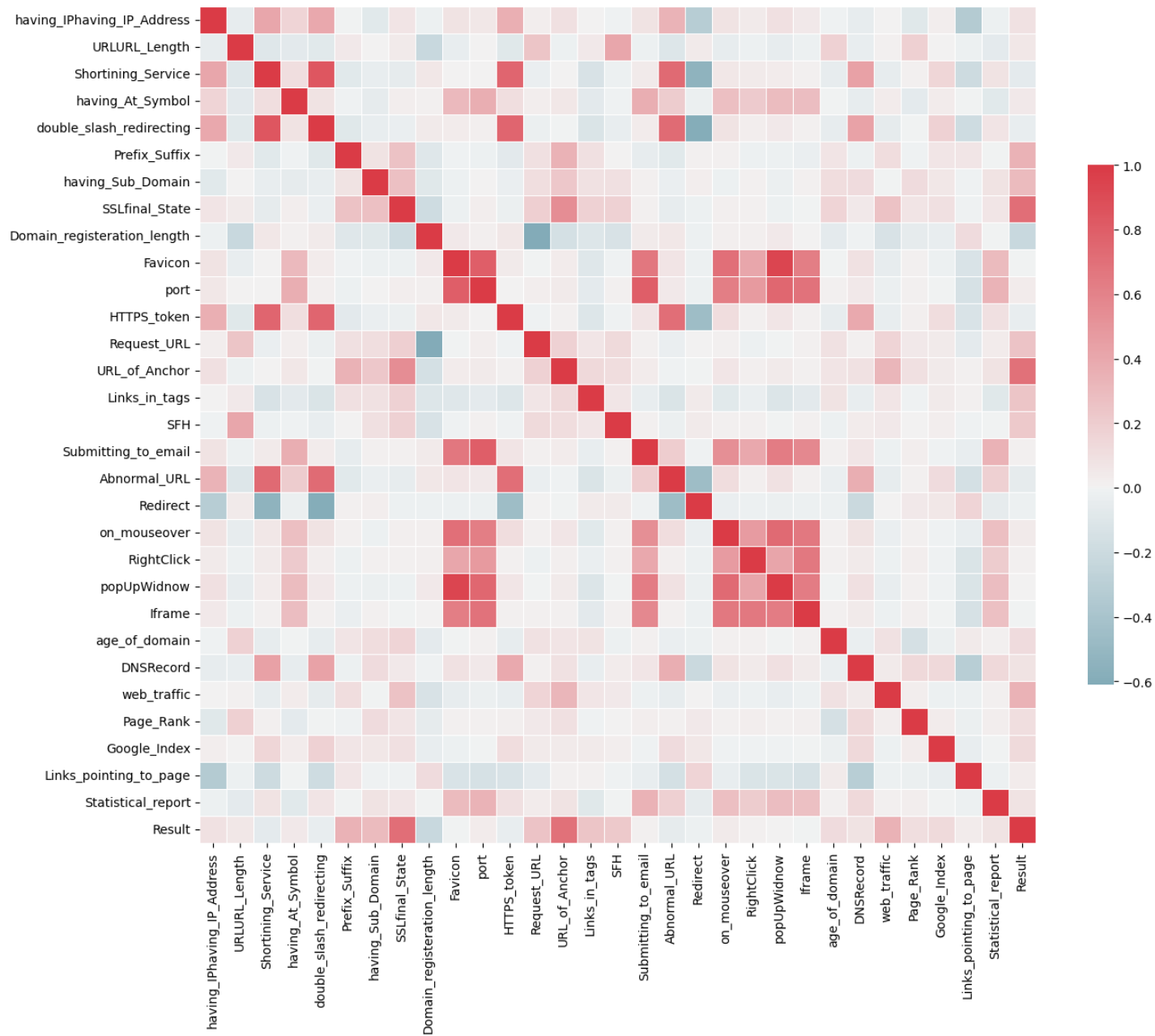
11055 rows x 31 columns

Ох уж ці науковці, намудрили із назвою колонок

Маємо 6157 легітимних та 4898 фішингових сайтів

```
print("number of 1",len(df[df["Result"]==1]))  
print("number of -1",len(df[df["Result"]==1]))  
number of 1 6157  
number of -1 4898
```

Загальна картина кореляції виглядає так:




Також присутній додатковий файл у зручному вигляді для опису (але він мені не потрібен):


<pre># df.describe().transpose().to_csv("nba.csv") df.describe().transpose()</pre>											
	count	mean	std	min	25%	50%	75%	max			
having_IPhaving_IP_Address	11055.0	0.313795	0.949534	-1.0	-1.0	1.0	1.0	1.0			
URLURL_Length	11055.0	-0.633198	0.786095	-1.0	-1.0	-1.0	-1.0	1.0			
Shortining_Service	11055.0	0.738761	0.673998	-1.0	1.0	1.0	1.0	1.0			
having_At_Symbol	11055.0	0.700588	0.713598	-1.0	1.0	1.0	1.0	1.0			
double_slash_redirecting	11055.0	0.741474	0.671011	-1.0	1.0	1.0	1.0	1.0			
Prefix_Suffix	11055.0	-0.734962	0.678139	-1.0	-1.0	-1.0	-1.0	1.0			
having_Sub_Domain	11055.0	0.063953	0.817518	-1.0	-1.0	0.0	1.0	1.0			
SSLfinal_State	11055.0	0.250927	0.911892	-1.0	-1.0	1.0	1.0	1.0			
Domain_registration_length	11055.0	-0.336771	0.941629	-1.0	-1.0	-1.0	1.0	1.0			
Favicon	11055.0	0.628584	0.777777	-1.0	1.0	1.0	1.0	1.0			
port	11055.0	0.728268	0.685324	-1.0	1.0	1.0	1.0	1.0			
HTTPS_token	11055.0	0.675079	0.737779	-1.0	1.0	1.0	1.0	1.0			
Request_URL	11055.0	0.186793	0.982444	-1.0	-1.0	1.0	1.0	1.0			
URL_of_Anchor	11055.0	-0.076526	0.715138	-1.0	-1.0	0.0	0.0	1.0			
Links_in_tags	11055.0	-0.118137	0.763973	-1.0	-1.0	0.0	0.0	1.0			
SFH	11055.0	-0.595749	0.759143	-1.0	-1.0	-1.0	-1.0	1.0			

Submitting_to_email	11055.0	0.635640	0.772021	-1.0	1.0	1.0	1.0	1.0
Abnormal_URL	11055.0	0.705292	0.708949	-1.0	1.0	1.0	1.0	1.0
Redirect	11055.0	0.115694	0.319872	0.0	0.0	0.0	0.0	1.0
on_mouseover	11055.0	0.762099	0.647490	-1.0	1.0	1.0	1.0	1.0
RightClick	11055.0	0.913885	0.405991	-1.0	1.0	1.0	1.0	1.0
popUpWidnow	11055.0	0.613388	0.789818	-1.0	1.0	1.0	1.0	1.0
Iframe	11055.0	0.816915	0.576784	-1.0	1.0	1.0	1.0	1.0
age_of_domain	11055.0	0.061239	0.998168	-1.0	-1.0	1.0	1.0	1.0
DNSRecord	11055.0	0.377114	0.926209	-1.0	-1.0	1.0	1.0	1.0
web_traffic	11055.0	0.287291	0.827733	-1.0	0.0	1.0	1.0	1.0
Page_Rank	11055.0	-0.483673	0.875289	-1.0	-1.0	-1.0	1.0	1.0
Google_Index	11055.0	0.721574	0.692369	-1.0	1.0	1.0	1.0	1.0
Links_pointing_to_page	11055.0	0.344007	0.569944	-1.0	0.0	0.0	1.0	1.0
Statistical_report	11055.0	0.719584	0.694437	-1.0	1.0	1.0	1.0	1.0
Result	11055.0	0.113885	0.993539	-1.0	-1.0	1.0	1.0	1.0

На цьому опис датасета від авторів закінчився. Я трохи допрацював тут аналіз даних та додав певну візуалізацію



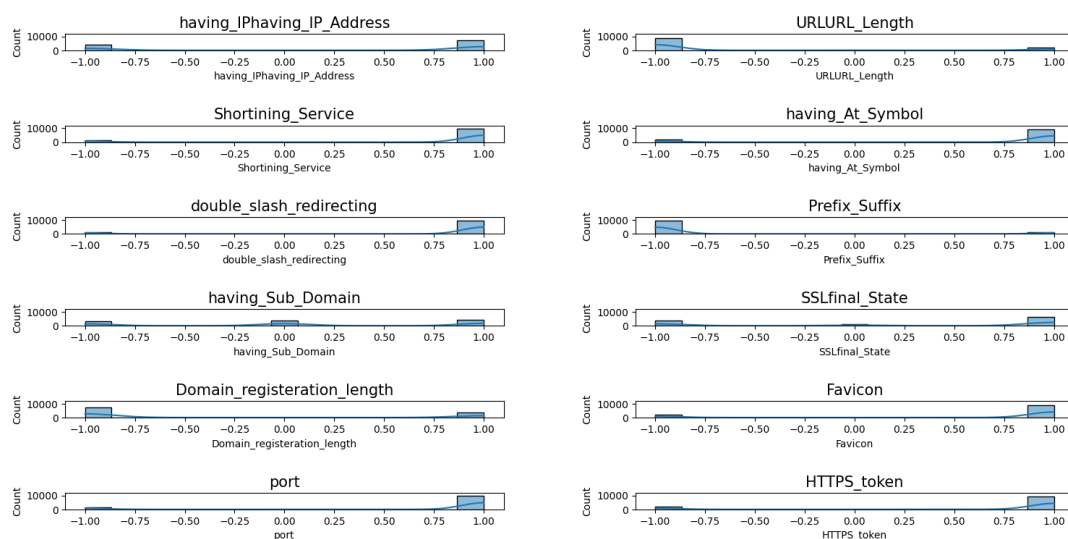
```
df.info()
```

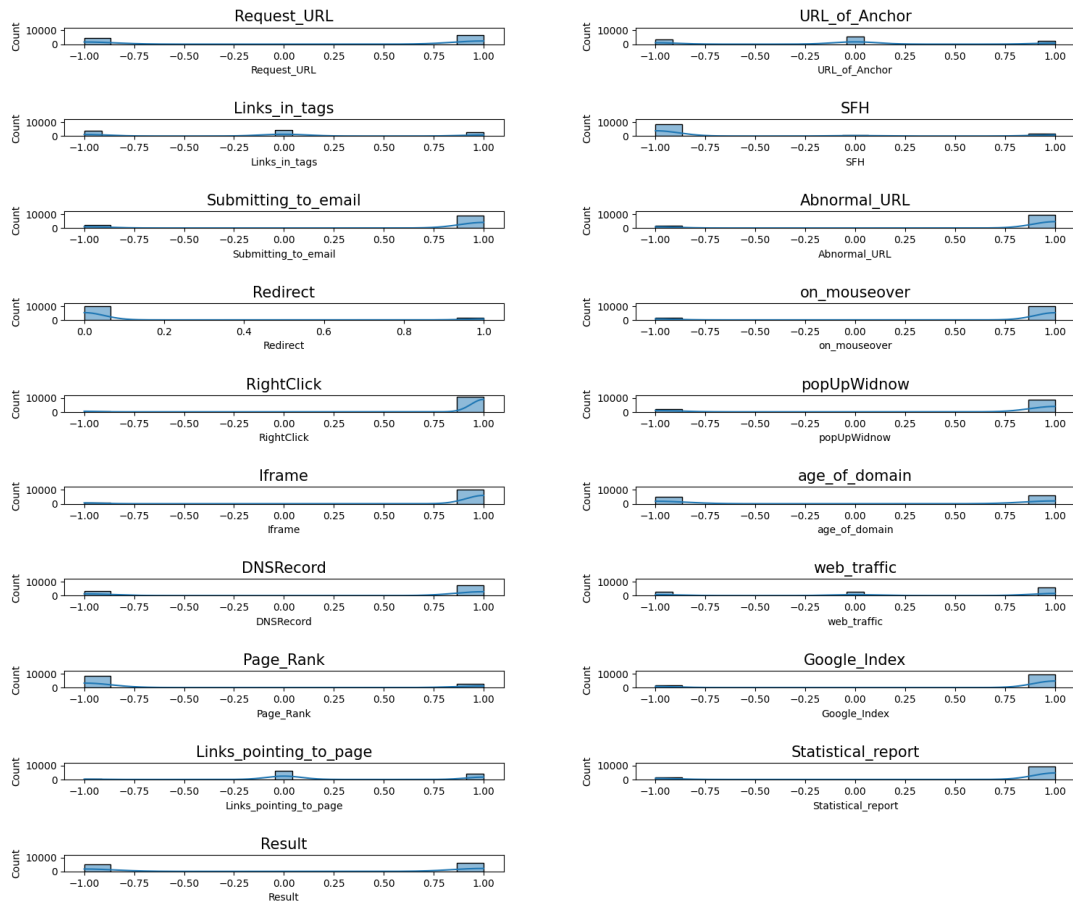


```
<class 'pandas.core.frame.DataFrame'>
Index: 11055 entries, 1 to 11055
Data columns (total 31 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   having_IPhaving_IP_Address               11055 non-null  int64
1   URLURL_Length                           11055 non-null  int64
2   Shortining_Service                       11055 non-null  int64
3   having_At_Symbol                         11055 non-null  int64
4   double_slash_redirecting                 11055 non-null  int64
5   Prefix_Suffix                            11055 non-null  int64
6   having_Sub_Domain                       11055 non-null  int64
7   SSLfinal_State                          11055 non-null  int64
8   Domain_registration_length               11055 non-null  int64
9   Favicon                                 11055 non-null  int64
10  port                                    11055 non-null  int64
11  HTTPS_token                             11055 non-null  int64
12  Request_URL                             11055 non-null  int64
13  URL_of_Anchor                           11055 non-null  int64
14  Links_in_tags                           11055 non-null  int64
15  SFH                                      11055 non-null  int64
16  Submitting_to_email                     11055 non-null  int64
17  Abnormal_URL                            11055 non-null  int64
18  Redirect                                11055 non-null  int64
19  on_mouseover                            11055 non-null  int64
20  RightClick                              11055 non-null  int64
21  popUpWidnow                             11055 non-null  int64
22  Iframe                                  11055 non-null  int64
23  age_of_domain                           11055 non-null  int64
24  DNSRecord                               11055 non-null  int64
25  web_traffic                             11055 non-null  int64
26  Page_Rank                               11055 non-null  int64
27  Google_Index                            11055 non-null  int64
28  Links_pointing_to_page                   11055 non-null  int64
29  Statistical_report                       11055 non-null  int64
30  Result                                  11055 non-null  int64

dtypes: int64(31)
memory usage: 2.7 MB
```

Далі вниз розподіл кількості колонок:





Очевидно, що дані нелінійні, тому автор і використовував **нелінійні підходи**
Графіки залежностей для усіх колонок для всіх нема сенсу (будуть точки)

- All_Classifair.ipynb

Після бібліотек, автори завантажують дані у рандомному порядку, хоча автори вказали що використали фіксований seed

Я не зміг знайти оригінальний seed, тому буду відносно аналізувати

```
[46] df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/dataset.csv",index_col=0)
df = sklearn.utils.shuffle(df, random_state=0)
X = df.drop("Result",axis=1).values
X = preprocessing.scale(X)
y = df["Result"].values
df.head()
```

index	having_IPhaving_IP_Address	URLURL_length	Shortning_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_State	Domain_registration_length
227	1	-1	1	1	1	-1	-1	1	1
2253	1	-1	1	1	1	-1	0	-1	-1
2647	1	-1	1	1	1	-1	0	1	-1
6445	1	-1	1	1	1	-1	0	-1	-1
1388	1	-1	1	1	1	-1	0	1	-1

5 rows x 31 columns

Також окрім метрик проводять 10фолд крос-валідацій

```
▼ Evaluation metrics
```

- Specifying evaluation metrics for classification models
- Using 10 fold-cross-validation for evaluating

```
scoring = {'accuracy': 'accuracy',  
          'recall': 'recall',  
          'precision': 'precision',  
          'f1': 'f1'}  
fold_count=10
```

Далі застосовуються модельки. Оригінальний вивід був у вигляді пайтон словника. Я допрацював та зробив табличками:

<div><div>▼ Decision Tree</div><div><pre>dtree_clf=DecisionTreeClassifier() cross_val_scores = cross_validate(dtree_clf, X, y, cv=fold_count, scoring=scoring) dtree_score = mean_score(cross_val_scores) pd.DataFrame([dtree_score])</pre></div><div><div><div>fit_time</div><div>score_time</div><div>test_accuracy</div><div>test_recall</div><div>test_precision</div><div>test_f1</div></div><table><tr><td>0</td><td>0.037794</td><td>0.014377</td><td>0.965173</td><td>0.971088</td><td>0.96663</td><td>0.968806</td></tr></table></div></div>	0	0.037794	0.014377	0.965173	0.971088	0.96663	0.968806	<div><div>▼ Random Forest</div><div><pre>rforest_clf=RandomForestClassifier() cross_val_scores = cross_validate(rforest_clf, X, y, cv=fold_count, scoring=scoring) rforest_clf_score = mean_score(cross_val_scores) pd.DataFrame([rforest_clf_score])</pre></div><div><div><div>fit_time</div><div>score_time</div><div>test_accuracy</div><div>test_recall</div><div>test_precision</div><div>test_f1</div></div><table><tr><td>0</td><td>0.651733</td><td>0.028811</td><td>0.973315</td><td>0.981972</td><td>0.97055</td><td>0.976189</td></tr></table></div></div>	0	0.651733	0.028811	0.973315	0.981972	0.97055	0.976189
0	0.037794	0.014377	0.965173	0.971088	0.96663	0.968806									
0	0.651733	0.028811	0.973315	0.981972	0.97055	0.976189									
<div><div>▼ Ada_Booster</div><div><pre>[52] adaboost_clf=AdaBoostClassifier() cross_val_scores = cross_validate(adaboost_clf, X, y, cv=fold_count, scoring=scoring) adaboost_clf_score = mean_score(cross_val_scores) pd.DataFrame([adaboost_clf_score])</pre></div><div><div><div>fit_time</div><div>score_time</div><div>test_accuracy</div><div>test_recall</div><div>test_precision</div><div>test_f1</div></div><table><tr><td>0</td><td>0.387293</td><td>0.02391</td><td>0.935777</td><td>0.954529</td><td>0.93194</td><td>0.943032</td></tr></table></div></div>		0	0.387293	0.02391	0.935777	0.954529	0.93194	0.943032							
0	0.387293	0.02391	0.935777	0.954529	0.93194	0.943032									

* для XGBClassifier стикнувся із проблемою, що він не сприймає [-1 1] як [0 1] у цільовій змінній y

```
[13] import numpy as np  
np.unique(y)  
  
array([-1, 1])
```

Перетворимо її та спробуємо запустити, воно спрацює. Тому продовжуємо запускати моделі та дивитись на точності:

▼ Gradient Boosting With XGBoost

```
yy = np.where(y == -1, 0, y)  
  
XGB_clf=XGBClassifier()  
cross_val_scores = cross_validate(XGB_clf, X, yy, cv=fold_count, scoring=scoring)  
XGB_clf_score = mean_score(cross_val_scores)  
pd.DataFrame([XGB_clf_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	0.304987	0.019681	0.971868	0.979863	0.970019	0.974874

▼ Gradient_Booster

```
gradientBooster_clf=GradientBoostingClassifier()  
cross_val_scores = cross_validate(gradientBooster_clf,X, y, cv=fold_count, scoring=scoring)  
gradientBooster_clf_score = mean_score(cross_val_scores)  
pd.DataFrame([gradientBooster_clf_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	1.274029	0.013127	0.949617	0.963137	0.947339	0.955139

▼ Histogram-Based Gradient Boosting

```
histGradientBooster_clf = HistGradientBoostingClassifier()  
cross_val_scores = cross_validate(histGradientBooster_clf,X, y, cv=fold_count, scoring=scoring)  
histGradientBooster_clf_score = mean_score(cross_val_scores)  
pd.DataFrame([histGradientBooster_clf_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	0.736664	0.024391	0.969155	0.976127	0.968808	0.972422

▼ LightGBM for Classification

```
LGBM_clf = LGBMClassifier()  
cross_val_scores = cross_validate(LGBM_clf,X, y, cv=fold_count, scoring=scoring)  
LGBM_clf_score = mean_score(cross_val_scores)  
pd.DataFrame([LGBM_clf_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	0.218835	0.01739	0.968612	0.97629	0.967713	0.971954

Gradient Boosting with CatBoost

```
catBoost_classifier = CatBoostClassifier(verbose=0, n_estimators=100)
cross_val_scores = cross_validate(catBoost_classifier, X, y, cv=fold_count, scoring=scoring)
catBoost_classifier_score = mean_score(cross_val_scores)
pd.DataFrame([catBoost_classifier_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	0.720222	0.026574	0.971055	0.980836	0.967729	0.974202

KNN

```
KNeighbors_clf=KNeighborsClassifier(3)
cross_val_scores = cross_validate(KNeighbors_clf, X, y, cv=fold_count, scoring=scoring)
KNeighbors_clf_score = mean_score(cross_val_scores)
pd.DataFrame([KNeighbors_clf_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	0.003825	0.092669	0.951425	0.960372	0.952831	0.956559

Logistic Regression

```
logistic_clf=LogisticRegression(random_state=1)
cross_val_scores = cross_validate(logistic_clf, X, y, cv=fold_count, scoring=scoring)
logistic_clf_score = mean_score(cross_val_scores)
pd.DataFrame([logistic_clf_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	0.061339	0.018659	0.927094	0.944296	0.926363	0.935183

NN

```
neural_clf=MLPClassifier(hidden_layer_sizes=(33,),max_iter=500)
cross_val_scores = cross_validate(neural_clf, X, y, cv=fold_count, scoring=scoring)
neural_clf_score = mean_score(cross_val_scores)
pd.DataFrame([neural_clf_score])
```

	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	12.141479	0.014895	0.969606	0.976611	0.969136	0.972821

SVM

```
##linear
linear_clf = svm.SVC(kernel='linear')
cross_val_scores = cross_validate(linear_clf, X, y, cv=fold_count, scoring=scoring)
linear_svc_clf_score = mean_score(cross_val_scores)
print("linear SVM:")
print(pd.DataFrame([linear_svc_clf_score]))

##poly
poly_clf = svm.SVC(kernel='poly')
cross_val_scores = cross_validate(poly_clf, X, y, cv=fold_count, scoring=scoring)
poly_svc_clf_score = mean_score(cross_val_scores)
print("poly SVM:")
print(pd.DataFrame([poly_svc_clf_score]))

##rbf
rbf_clf = svm.SVC(kernel='rbf')
cross_val_scores = cross_validate(rbf_clf, X, y, cv=fold_count, scoring=scoring)
rbf_svc_clf_score = mean_score(cross_val_scores)
print("rbf SVM:")
print(pd.DataFrame([rbf_svc_clf_score]))

##sigmoid
sigmoid_clf = svm.SVC(kernel='sigmoid')
cross_val_scores = cross_validate(sigmoid_clf, X, y, cv=fold_count, scoring=scoring)
sigmoid_svc_clf_score = mean_score(cross_val_scores)
print("sigmoid SVM:")
print(pd.DataFrame([sigmoid_svc_clf_score]))
```

linear SVM:						
	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	2.929038	0.092372	0.927817	0.946731	0.925456	0.935933
poly SVM:						
	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	1.867397	0.127072	0.950069	0.970606	0.941693	0.955880
rbf SVM:						
	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	1.869217	0.210418	0.952874	0.969957	0.946819	0.958217
sigmoid SVM:						
	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
0	2.266515	0.174657	0.828766	0.847329	0.845607	0.846426

Отримані значення не сильно відрізняються від тих, що залишені в оригінальному ноутбучі, на 0.0001 одиниці

Помітив, що моделей більше ніж в таблиці в статті (15 проти 12)

Організую мої отримані результати в загальну таблицю, при цьому відсортую за спаданням **test_accuracy**:

	Model	fit_time	score_time	test_accuracy	test_recall	test_precision	test_f1
2	Random Forest	0.651733	0.028811	0.973315	0.981972	0.970550	0.976189
11	XGBoost	0.304987	0.019681	0.971868	0.979863	0.970019	0.974874
12	Gradient CatBoost	0.720222	0.026574	0.971055	0.980836	0.967729	0.974202
5	Neural Network (MLP)	12.141479	0.014895	0.969606	0.976611	0.969136	0.972821
14	Histogram-Based GB	0.736664	0.024391	0.969155	0.976127	0.968808	0.972422
13	LightGBM	0.218835	0.017390	0.968612	0.976290	0.967713	0.971954
1	Decision Tree	0.037794	0.014377	0.965173	0.971088	0.966630	0.968806
8	SVM rbf	1.869217	0.210418	0.952874	0.969957	0.946819	0.958217
4	KNN	0.003825	0.092669	0.951425	0.960372	0.952831	0.956559
7	SVM poly	1.867397	0.127072	0.950069	0.970606	0.941693	0.955880
10	Gradient Boosting	1.274029	0.013127	0.949617	0.963137	0.947339	0.955139
3	Ada Booster	0.387293	0.023910	0.935777	0.954529	0.931940	0.943032
6	SVM linear	2.929038	0.092372	0.927817	0.946731	0.925456	0.935933
0	Logistic Regression	0.061339	0.018659	0.927094	0.944296	0.926363	0.935183
9	SVM sigmoid	2.266515	0.174657	0.828766	0.847329	0.845607	0.846426

Найкращі модельки це **RandomForest, XGBoost**, Gradient CatBoost, MLP

Мої перші 2 найкращі модельки співпали із думкою авторів, тому робота пророблена +- правильно

4. За бажанням (на додаткові +2 бали) запропонувати можливі шляхи покращення отриманих результатів у статті (покращення попередньої обробки даних, використання додаткових джерел даних, удосконалення використаної моделі тощо)

Було додано візуалізацію до даних, щоправда безкорисна

Було удосконалено вивід інформації про перформанс моделей та їх загальне порівняння в одну таблицю в ноутбучі

Можна було б для більшості моделей погратись із гіперпараметрами щоб знайти найбільш вдалу точність. Але маючи багатьом моделей, це не дуже доцільно, тимпаче в цьому випадку підняття точності буде взагалі некритичним

5. Оформити свою роботу у вигляді протоколу з описом розглянутої статті, висновками та посиланням на власний GitHub з відповідним кодом

Pls don't look at my GitHub (nothing to find there, like, literally, pls)

https://github.com/inimitable-carp2099/ML_lab6_phishing

Дякую за увагу

Висновок: в данній ЛР я розглянув статтю присвячену застосуванню машинного навчання для виявлення фішингових вебсайтів. Дані було обрані із рейтингу сайтів та промічені за 30 параметрами, які можуть вказувати та потенційну небезпеку. Було застосовано близько 15 моделей для навчання. Серед них найефективнішими виявились 2: RandomForest та XGBoost. Також

були і інші рішення, які мали схожі точності. Загалом, робота показала просте але ефективне використання МН у справі кібезбезпеки

Кошенятко після 6 ЛР:

