ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Метод виявлення вторгнень у веб-сеанси користувача

Розробила:
ст. гр. СП$_М$-21-1
Запорожець Н.О.

Керівник роботи:
доц.каф. ЕОМ, к.т.н.
Мартовицький В.О.

Мета та задачі роботи

Мета роботи – розробка методів та засобів виявлення вторгнень, які б давали змогу ідентифікувати користувачів інформаційної комп'ютерної системи за їх поведінкою.

Задачі роботи:

1) огляд літературних джерел, присвячених темі дослідження, та аналіз існуючих підходів до вирішення проблеми;
2) огляд особливостей та класифікації систем виявлення та попередження вторгнень у роботу комп'ютерних систем;
3) розробка моделі ідентифікації користувачів за їх поведінкою в системі з використанням методів машинного навчання;
4) експериментальне дослідження запропонованого підходу до ідентифікації користувачів системи.

3

Системи виявлення і попередження вторгнень

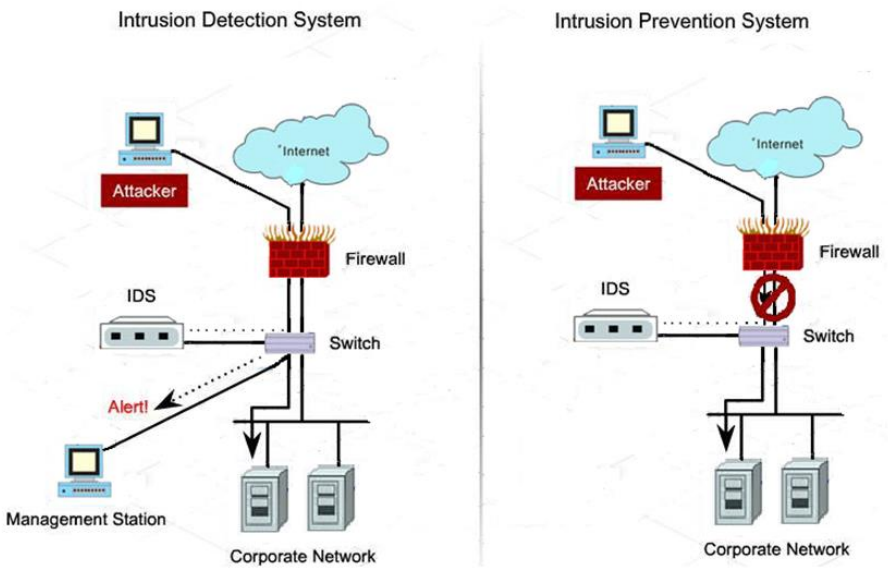
IDS (*Intrusion Detection System*) – система виявлення вторгнень, здійснює моніторинг загроз.

IPS (*Intrusion Prevention System*) – система попередження вторгнень, займається запобіганням загрозам.

IDS/IPS системи – програмні та апаратні інструменти для захисту мереж від несанкціонованого доступу. Вони здатні автоматично виявляти факти вторгнень та запобігати їм, сповіщаючи відповідальних фахівців.

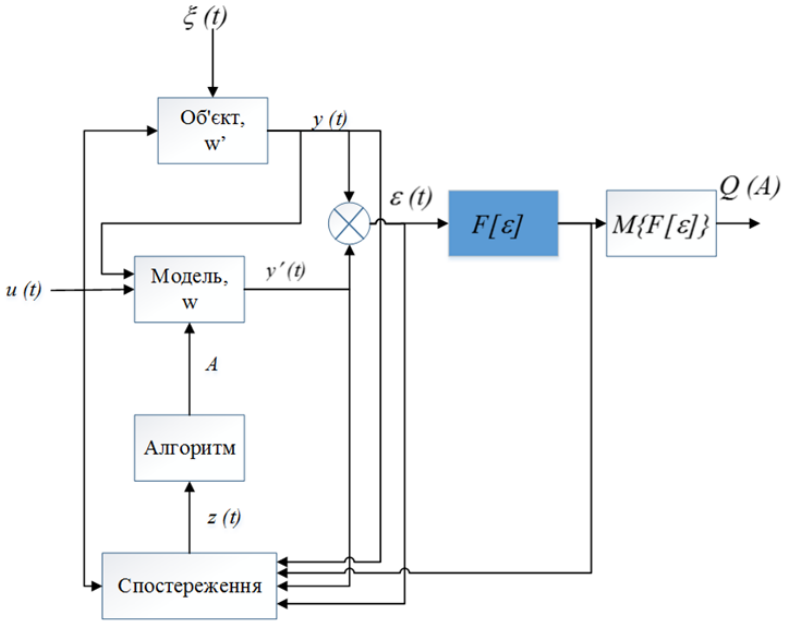
4

Поведінка IDS та IPS під час атаки



5

Функціональна схема ідентифікації



6

Модель ідентифікації користувача

Бінарний класифікатор для виявлення легітимного користувача або порушника, що використовує логістичну регресію

$$a(x,w) = sign\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right) = sign\langle x,w\rangle, \qquad (1)$$

де w_j – вага j-ї ознаки;
w_0 – поріг прийняття рішення;
$w = (w_0, w_1, \ldots, w_n)$ – вектор ваг;
$\langle x,w\rangle$ – скалярний добуток ознак об'єкта на вектор ваг.

Критерій ідентифікації

$$Q(w,X) = \frac{1}{m}\sum_{i=1}^{m} \log(1+\exp(-y_i\langle x_i,w\rangle)) + \frac{\lambda_2}{2}\|w\|^2 \to \min_w. \qquad (2)$$

7

Модель ідентифікації користувача

Оптимальний вектор параметрів моделі будемо шукати за допомогою градієнтного спуску:

$$w^{(k+1)} = w^{(k)} - \alpha \nabla_w Q(w, X). \qquad (3)$$

Градієнт за об'єктом x_i визначається за формулою

$$\nabla_w Q(w, x_i) = -\frac{y_i x_i}{1 + \exp(-y_i \langle w, x_i \rangle)} + \lambda_2 w. \qquad (4)$$

Критерії зупинки (використовуються одночасно):

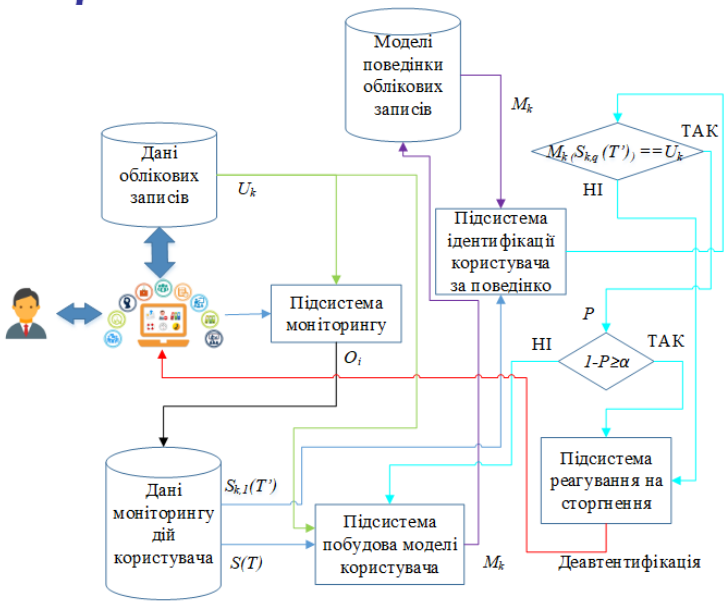
- перевірка на евклідову норму різниці вагів на двох сусідніх ітераціях (наприклад, менше деякого малого числа порядку 10^{-6});
- досягнення максимальної кількості ітерацій (наприклад, 1000).

8

Схема процесу побудови моделі поведінки користувача

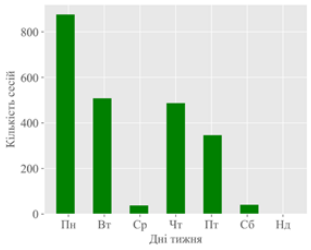
9

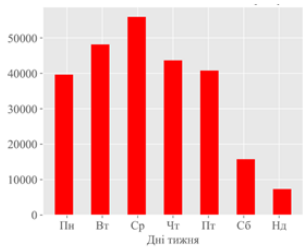
Схема додаткового захисту системи від вторгнення



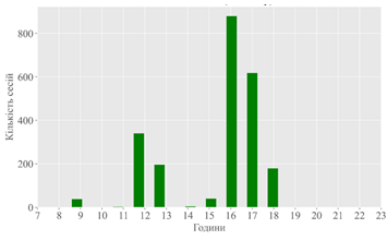
10

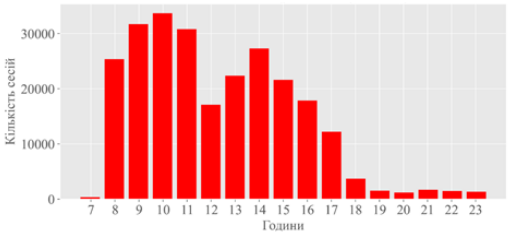
Результати експериментальних досліджень





Розподіл сесій по днях тижня для користувача Еліс та інших користувачів



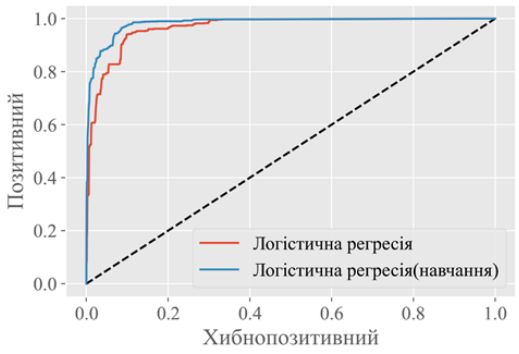


Розподіл сесій по годинах для користувача Еліс та інших користувачів

11

Оцінка якості моделей

Методи побудови моделей ідентифікації	Оцінка якості на валідаційній вибірці	Оцінка якості на тестовій вибірці
Логістична регресія	0.983091	0.966114
Алгоритм k-найближчих сусідів	0.983124	0.956038
Випадковий ліс	0.983183	0.967171
Метод опорних векторів	0.983519	0.965642



ROC-крива для логістичної регресії

12

Висновки

1. Розроблено функціональну модель процесу ідентифікації користувачів за їх поведінкою в системі, що дозволяє створити додаткові засоби захисту користувачів системи у випадку крадіжки їх даних автентифікації. Слід виділити такі переваги запропонованої моделі:
   - незалежність від кількості користувачів в системі, оскільки даний підхід використовує модель оцінки поведінки авторизованого користувача і поточні характеристики користувача;
   - можливість виявлення прихованих закономірностей в поведінці користувача, це досягається за рахунок використання методів машинного навчання;
   - адаптація до зміни поведінки користувачів.
2. Проведено експериментальне дослідження щодо запропонованого підходу ідентифікації користувача за його поведінкою в системі. Побудовані моделі поведінки користувача з використанням методів машинного навчання показали оцінку якості ідентифікації більше 0,95.Не обов'язково використовувати складні моделі при описі поведінки користувача, головне сформувати інформативні ознаки та правильно сформувати валідаційну та тестову вибірки.

ДОДАТОК Б

Код програми

```
#!/usr/bin/env python
# coding: utf-8

# Підключення необхідних бібліотек

# In[1]:

import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression #
LogisticRegression
from sklearn.model_selection import train_test_split # to split
from sklearn.model_selection import TimeSeriesSplit # to split
from sklearn.metrics import roc_curve # ROC curve
from sklearn.metrics import roc_auc_score # ROC score
from sklearn.model_selection import cross_val_score # cross
validation score
from sklearn.preprocessing import StandardScaler #
Standartization
from sklearn.preprocessing import MinMaxScaler # Standartization
from sklearn.model_selection import GridSearchCV # to find best
parameters
from sklearn.ensemble import RandomForestClassifier # Random
Forest
from sklearn.ensemble import GradientBoostingClassifier #
Gradient Boosting
from sklearn.ensemble import BaggingClassifier # Bagging
from sklearn.ensemble import VotingClassifier # Ensemble models
(voting and mean)
from sklearn.metrics import confusion_matrix # Confusion Matrix
from sklearn.svm import SVC # SVM
from sklearn.svm import LinearSVC # SVM
from sklearn.pipeline import make_pipeline # Pipepline

import xgboost as xgb # Gradient Boosting
from mlxtend.classifier import StackingClassifier # Stacking
from mlxtend.classifier import StackingCVClassifier # Stacking
import lightgbm as lgb # Microsoft Gradient Boosting
import matplotlib.pyplot as plt # to plot
plt.style.use('ggplot')
import seaborn as sns # to plot
import numpy as np # to count
import pandas as pd # DataFrames
from scipy import stats # Statistics
import pickle # to read pickle files
from IPython.display import display # Display tables
```

```
import time # time checking
import re # regular expressions
import eli5 # to check features importance
from collections import Counter # to count
pd.set_option('display.max_columns', None) # to display max
columns in DataFrames
pd.options.mode.chained_assignment = None  # don't lool at
mistakes


# In[2]:

import os
for dirname, _, filenames in os.walk('data'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


# In[3]:

def load_train_or_test():
    i = input('train or test ')
    if i == 'test' or i =='train':
        name = f"{i}_sessions.csv"
        df = pd.read_csv(f"data" + name)
        return df
    else:
        pass


# In[4]:

def norm(data):
    return (data)/(max(data)-min(data)) # to norm


# In[5]:

df_train = pd.read_csv("data/train_sessions.csv")


# In[6]:

df_train.head()


# In[7]:

df_train["target"].value_counts(ascending=True)


# In[8]:
```

```
def time_operations(df):
    if 'target' in df.columns:
        df['number_of_sites'] = ((df.notna().sum(axis=1)) / 2) -
1
    else:
        df['number_of_sites'] = ((df.notna().sum(axis=1)) / 2) -
0.5
    for col in [f"time{i}" for i in range(1,11)]:
        df[col] = pd.to_datetime(df[col])
    df['session_strt'] = df[[f"time{i}" for i in
range(1,11)]].min(axis=1)
    df['session_end'] = df[[f"time{i}" for i in
range(1,11)]].max(axis=1)
    df['session_len'] = (df['session_end'] -
df['session_strt']).dt.seconds
    df['weekday'] = df['session_strt'].dt.dayofweek
    df['weekday'] = df['weekday'].astype('category')
    df['start_hour'] = df['session_strt'].dt.hour
    df['start_hour'] = df['start_hour'].astype('category')
    df['end_hour'] = df['session_end'].dt.hour
    df['end_hour'] = df['end_hour'].astype('category')
    df['day'] = df['session_strt'].dt.day
    df['day'] = df['day'].astype('category')
    df['minute'] = df['session_strt'].dt.minute
    df['month'] = df['session_strt'].dt.month
    df = df.sort_values('session_strt', ignore_index=True)
    df = df.reset_index()
    df = df.rename(columns={'index':'time_index'})
    df = df.sort_values('session_id', ignore_index=True)
    return df


# In[9]:

df_train = time_operations(df_train)


# In[10]:

df_train.head()


# In[11]:

df_train.info()


# In[12]:

plt.figure(figsize=(10,7))
plt.hist(df_train.weekday, bins=7,rwidth=0.9,color='green')
plt.rcParams['font.size'] = '14'
plt.rcParams['font.family'] = 'Times New Roman'
```

```
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.xticks(ticks=[i - 0.5 for i in np.linspace(1,6,7)],
labels=['Пн','Вт','Ср','Чт','Пт','Сб','Нд'])
plt.xlabel('Дні тижня', fontname='Times New Roman')
plt.ylabel('Кількість сесій', fontname='Times New Roman')
#plt.title('Розподіл сесій до дням тіжня', fontname='Times New
Roman',fontsize=15, color = 'black')
plt.show()


# In[13]:

plt.figure(figsize=(5.5,4.5))

plt.rcParams['font.size'] = '14'
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.hist(df_train[df_train.target==1].weekday, bins=7,
color='green', label='Alice', rwidth=0.5)
plt.xticks(ticks=[i - 0.5 for i in
np.linspace(1,6,7)],labels=['Mo','Tu','Wd','Th','Fr','Sat','Sun'
])
plt.xlabel('Weekdays', fontname='Times New Roman',fontsize=14)
plt.ylabel('Number of sessions', fontname='Times New
Roman',fontsize=14)
plt.title('Distribution of sessions by day of the week for
Alice', fontname='Times New Roman',fontsize=14)

plt.savefig("Рис. 4_En.pdf", dpi=400, format="pdf")
plt.show()


# In[14]:

plt.figure(figsize=(6.5,4.5))
plt.rcParams['font.size'] = '14'
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.hist(df_train[df_train.target==0].weekday, bins=7,
color='red', label='Alice', rwidth=0.5)
plt.xticks(ticks=[i - 0.5 for i in
```

```
np.linspace(1,6,7)],labels=['Mo','Tu','Wd','Th','Fr','Sat','Sun'
])
plt.xlabel('Weekdays', fontname='Times New Roman',fontsize=14)
plt.ylabel('Number of sessions', fontname='Times New
Roman',fontsize=14)
plt.title('Distribution of sessions by day of the week for other
users', fontname='Times New Roman',fontsize=14)

plt.savefig("Рис. 5_En.pdf", dpi=400, format="pdf")
plt.show()


# In[15]:

plt.figure(figsize=(10,6))
plt.rcParams['font.size'] = 17
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.hist(df_train[df_train.target==1].start_hour, bins=20,
color='green', label='Alice', rwidth=1)
plt.xticks(ticks=[i + 0.5 for i in
np.linspace(7,22,17)],labels=['7','8'
,'9','10','11','12','13','14','15','16','17',
                  '18','19','20','21','22','23'])
plt.xlabel('Hour', fontname='Times New Roman',fontsize=17)
plt.ylabel('Number of sessions', fontname='Times New
Roman',fontsize=17)
plt.title("Alice's timetable", fontname='Times New
Roman',fontsize=17)

plt.savefig("Рис. 6_En.pdf", dpi=400, format="pdf")
plt.show()


# In[16]:

plt.figure(figsize=(10,4.5))
plt.rcParams['font.size'] = 17
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.hist(df_train[df_train.target==0].start_hour, bins=17,
color='red', label='Alice', rwidth=0.8)
plt.xticks(ticks=[i + 0.5 for i in
np.linspace(7,22,17)],labels=['7','8'
,'9','10','11','12','13','14','15','16','17',
```

```
                  '18','19','20','21','22','23'])
plt.xlabel('Hour', fontname='Times New Roman',fontsize=17)
plt.ylabel('Number of sessions', fontname='Times New
Roman',fontsize=17)
plt.title('Timetable for other users', fontname='Times New
Roman',fontsize=17)

plt.savefig("Рис. 7_En.pdf", dpi=400, format="pdf")
plt.show()


# In[17]:

plt.figure(figsize=(10,6))
plt.rcParams['font.size'] = 17
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.hist(np.log(df_train[df_train.target==1].session_len + 1),
bins=30, color='green', label='Alice', rwidth=0.5)
plt.xlabel('log(session_len)', fontname='Times New
Roman',fontsize=17)
plt.ylabel('Number of sessions', fontname='Times New
Roman',fontsize=17)
#plt.title('Розклад дня Еліс (за годину)', fontname='Times New
Roman',fontsize=17)

plt.savefig("Рис. 8_En.pdf", dpi=400, format="pdf")
plt.show()


# In[18]:

plt.figure(figsize=(10,6))
plt.rcParams['font.size'] = 17
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.hist(np.log(df_train[df_train.target==0].session_len + 1),
bins=30, color='red', label='Alice', rwidth=0.5)
plt.xlabel('log(session_len)', fontname='Times New
Roman',fontsize=17)
plt.ylabel('Number of sessions', fontname='Times New
Roman',fontsize=17)
#plt.title('Розклад дня Еліс (за годину)', fontname='Times New
Roman',fontsize=17)
```

```
plt.savefig("Рис. 9_En.pdf", dpi=400, format="pdf")
plt.show()


# In[19]:

def alice_sites(train_df):
    for j in range(1,11):
      if j == 1:
        df =
train_df[['session_id','site{i}'.format(i=j),'target']]
        df.columns = ['session_id','site', 'target']
      if j<10:
        df_2 = train_df[['session_id','site{i}'.format(i=j+1),
'target']]
        df_2.columns = ['session_id','site', 'target']
        df = pd.concat([df, df_2], ignore_index=True)
    sites = df.groupby('site')['target'].mean()
    return sites


# In[20]:

def time_treshold_save_zero(df, date='2014-03-01'):
    df['day'] = df['session_strt'].dt.dayofyear
    train_df = df[df.session_strt < date]
    test_df = df[df.session_strt >= date]
    return train_df, test_df

def add_alice_sites(df, sites, alice_size = 0.06):
    mask = sites[sites > alice_size].index
    df['alice_site'] = 0
    for col in [f"site{i}" for i in range(1,11)]:
        alice_mask = df[col].isin(mask)
        df.alice_site[alice_mask] = 1
    return df


# In[21]:

def add_more_alice_sites(df, sites, a_s_2 = 0.3, a_s_3 = 0.01,
a_s_4 = 0.5):
    count = 1
    for j in [a_s_2, a_s_3, a_s_4]:
        count += 1
        mask = sites[sites > j].index
        df[f'alice_site_{count}'] = 0
        for col in [f"site{i}" for i in range(1,11)]:
            alice_mask = df[col].isin(mask)
            df.loc[alice_mask, f'alice_site_{count}'] = 1
    return df
```

```
# In[22]:

train_df, test_df = time_treshold_save_zero(df_train)


# In[23]:

sites = alice_sites(train_df)


# In[24]:

train_df = add_alice_sites(train_df, sites, alice_size = 0.06)


# In[25]:

train_df = add_more_alice_sites(train_df, sites)


# In[26]:

work_df = train_df.groupby('day')['alice_site'].sum()
work_df


# In[27]:

good_days = work_df[(work_df > 70) == True].index
good_days


# In[28]:

bad_days = work_df[(work_df <= 70) == True].index
bad_days


# In[29]:

df_train[df_train.target==1].start_hour.value_counts()


# In[30]:

df_train[(df_train.target==1) & (df_train.start_hour==11)]


# In[31]:

df_train[(df_train.target==1) & (df_train.start_hour==14)]
```

```
# In[32]:

df_train[(df_train.target==1) &
(df_train.start_hour==18)].sort_values('minute')


# In[33]:

def remove_days(df, border=70):

    df = df.set_index('session_id')

    work_df = df.groupby('day')['alice_site'].sum()
    good_days = work_df[(work_df > border) == True].index
    bad_days = work_df[(work_df <= border) == True].index
    good_session_id = df[df.day.isin(good_days)].index
    bad_session_id = df[df.day.isin(bad_days)].index

    bad_hours_array = np.array([7,8,10,11,14,19,20,21,22,23])
    good_hours_array = np.array([12,13,16,17])
    good_hour_session_id =
df[df.start_hour.isin(good_hours_array)].index
    bad_hour_session_id =
df[df.start_hour.isin(bad_hours_array)].index

    rare_hours_array = np.array([9,18,15])

    nine_oclock_bad_minutes = np.arange(30,60)
    fiften_oclock_bad_minutes = np.arange(20,50)
    six_oclock_bad_minutes = np.arange(25,60)

    nine_oclock_good_minutes = np.arange(0,30)
    fifteen_oclock_good_minutes = np.append(np.arange(0,20),
np.arange(50,60))
    six_oclock_good_minutes = np.arange(0,25)

    bad_nine_oclock_session_id = df[(df.start_hour == 9) &
(df.minute.isin(nine_oclock_bad_minutes))].index
    bad_fifteen_oclock_session_id = df[(df.start_hour == 15) &
(df.minute.isin(fiften_oclock_bad_minutes))].index
    bad_six_oclock_session_id = df[(df.start_hour == 18) &
(df.minute.isin(six_oclock_bad_minutes))].index

    good_nine_oclock_session_id = df[(df.start_hour == 9) &
(df.day.isin(good_days)) &
(df.minute.isin(nine_oclock_good_minutes))].index
    good_fifteen_oclock_session_id = df[(df.start_hour == 15) &
(df.day.isin(good_days)) &
(df.minute.isin(fifteen_oclock_good_minutes))].index
    good_six_oclock_session_id = df[(df.start_hour == 18) &
(df.day.isin(good_days)) &
(df.minute.isin(six_oclock_good_minutes))].index
```

```
    good_session_id = np.intersect1d(good_session_id,
good_hour_session_id)

    good_minutes_session_id =
np.concatenate((good_nine_oclock_session_id,

good_fifteen_oclock_session_id, good_six_oclock_session_id),
axis=None)

    bad_minutes_session_id =
np.concatenate((bad_nine_oclock_session_id,

bad_fifteen_oclock_session_id, bad_six_oclock_session_id),
axis=None)

    good_session_id = np.unique(np.concatenate((good_session_id,
good_minutes_session_id), axis=None))

    bad_session_id = np.unique(np.concatenate((bad_session_id,
bad_hour_session_id), axis=None))
    bad_session_id = np.unique(np.concatenate((bad_session_id,
bad_minutes_session_id), axis=None))


    df = df.drop('minute', axis=1)
    df = df.loc[good_session_id]
    df = df.reset_index()

    return df, good_session_id, bad_session_id


# In[34]:

def get_dummies(train_df, more_features):
    l1 = more_features
    l2 =
['number_of_sites','session_len','weekday','start_hour',
'end_hour', 'time_index', 'session_id']
    col_names = l1 + l2
    train_df = train_df[col_names]
    train_df = train_df.set_index('session_id')
    train_df = pd.get_dummies(train_df,
columns=['weekday','start_hour', 'end_hour', 'number_of_sites'],
                              drop_first=True)
    train_df = train_df.dropna()
    return train_df


# In[35]:

train_df_short, good_session_id_train, bad_session_id_train =
remove_days(train_df, border=70)
```

```
test_df = add_alice_sites(test_df, sites, alice_size = 0.06)
test_df = add_more_alice_sites(test_df, sites)

test_df_short, good_session_id_test, bad_session_id_test =
remove_days(test_df, border=70)


# In[36]:

more_features = ['alice_site', 'alice_site_2', 'alice_site_3',
'alice_site_4', 'target']


train_df_short = get_dummies(train_df_short, more_features)
test_df_short = get_dummies(test_df_short, more_features)


# In[37]:

train_df_short.head()


# In[39]:

X_train = train_df_short.drop('target', axis=1)
X_train = X_train.drop(['start_hour_8',
'end_hour_8','start_hour_10','end_hour_10','start_hour_11','end_
hour_11','start_hour_12','end_hour_12',

'start_hour_14','end_hour_14','start_hour_19','end_hour_19','sta
rt_hour_20','end_hour_20','start_hour_21','end_hour_21',

'start_hour_22','end_hour_22','start_hour_23','end_hour_23','tim
e_index', 'alice_site_4'], axis=1)

y_train = train_df_short.target

y_train_valid = train_df.target

X_test = test_df_short.drop('target', axis=1)
X_test = X_test.drop(['start_hour_8',
'end_hour_8','start_hour_10','end_hour_10','start_hour_11','end_
hour_11','start_hour_12','end_hour_12',

'start_hour_14','end_hour_14','start_hour_19','end_hour_19','sta
rt_hour_20','end_hour_20','start_hour_21','end_hour_21',

'start_hour_22','end_hour_22','start_hour_23','end_hour_23','tim
e_index', 'alice_site_4'], axis=1)
y_test = test_df.target
```

```
log = LogisticRegression(C=4, multi_class='ovr', penalty='l1',
solver='liblinear', random_state=126)
log.fit(X_train, y_train)

# TEST PART
# predict data with Alice (DataFrame with good_sessions_id)
y_pred_prob_lg = log.predict_proba(X_test)[:,1]
final_df_test_start = pd.DataFrame(y_pred_prob_lg, columns =
['target'], index=good_session_id_test)

# prepare data without Alice (DataFrame with bad_sessions_id
filled by zeros)
empty_result = np.zeros((len(bad_session_id_test),), dtype=int)
empty_df = pd.DataFrame(empty_result, columns = ['target'],
index=bad_session_id_test)

# concat two DataFrames
final_df_test = pd.concat([final_df_test_start, empty_df],
axis=0)
y_pred = np.array(final_df_test.sort_index().target)

# plot the ROC Curve
fpr, tpr, trhesholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(10,6))
plt.rcParams['font.size'] = 14
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['text.color'] = 'black'
plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.plot([0, 1], [0, 1],'k--')
plt.plot(fpr, tpr, label='Logistic regression')
plt.xlabel('False-positive')
plt.ylabel('Positive')
#plt.title('Logistic Regression ROC Curve')Крива ROC логістичної
регресії
plt.legend(fontsize=14)



# TRAIN PART
# predict data with Alice (DataFrame with good_sessions_id)
y_pred_prob_lg_train = log.predict_proba(X_train)[:,1]
final_df_train_start = pd.DataFrame(y_pred_prob_lg_train,
columns = ['target'], index=good_session_id_train)

# prepare data without Alice (DataFrame with bad_sessions_id
filled by zeros)
empty_result = np.zeros((len(bad_session_id_train),), dtype=int)
empty_df = pd.DataFrame(empty_result, columns = ['target'],
index=bad_session_id_train)
```

```
# concat two DataFrames
final_df_train = pd.concat([final_df_train_start, empty_df],
axis=0)
y_pred_train = np.array(final_df_train.sort_index().target)

# plot the ROC Curve
fpr, tpr, trhesholds = roc_curve(y_train_valid, y_pred_train)
plt.plot([0, 1], [0, 1],'k--')
plt.plot(fpr, tpr, label='Logistic regression (training)')
plt.legend()

plt.savefig("Рис. 10_En.pdf", dpi=400, format="pdf")
plt.show()

print(roc_auc_score(y_test, y_pred))
print(roc_auc_score(y_train_valid, y_pred_train))


# In[ ]:
```