Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №8

Аналіз даних з використанням мови Python

Тема: Класифікація в scikit-learn

Варіант: 1

Виконав

студент групи ІП-11:

Панченко С. В.

Перевірив:

Тимофєєва Ю. С

Київ 2023

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Написати програму, яка навчає та тестує модель, що виконує задачу бінарної класифікації відповідно до варіанту, оцінити модель за допомогою відповідних метрик та спробувати її покращити (підібрати інші гіперпараметри або використати інший алгоритм класифікації, або змінити комбінацію ознак).

Оформити звіт. Звіт повинен містити:

— титульний лист;

— код програми;

— результати виконання коду.

Продемонструвати роботу програми та відповісти на питання стосовно теоретичних відомостей та роботи програми.

Варіант 1: lab2/possum.csv. Класифікувати опосумів за статтю, використовуючи будь-які незалежні змінні.

2 ЗАВДАННЯ

Написати програму,яка здійснює попередню обробку даних, навчає та тестує(при потребі) модель, що виконує завдання відповідно до варіанту, оцінити модель за допомогою відповідних метрик.

Оформити звіт. Звіт повинен містити:

—титульний лист;

—код програми;

—результати виконання коду.

Продемонструвати роботу програми та відповісти на питання стосовно теоретичних відомостей та роботи програми.

Варіант 1: lab2/Crime.csv. Використати будь-яку комбінацію незалежних ознак (не менше двох), щоб спрогнозувати рівень злочинності в даний час.

3 ВИКОНАННЯ

3.1 Завантаження даних

Імпортуємо модулі pandas, numpy, matplotlib.pyplot, seaborn. Завантажимо датафрейм та виведемо інформацію про нього. Видалимо колонку "Unnamed: 0" та "case", що позначає індекси, оскільки вони вже були автоматично створені.

```
In [238…
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('data/possum.csv')
df.drop(['Unnamed: 0', 'case'], axis=1, inplace=True)
df
```

Out[238]:

	site	Pop	sex	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye	chest	belly
0	1	Vic	m	8.0	94.1	60.4	89.0	36.0	74.5	54.5	15.2	28.0	36.0
1	1	Vic	f	6.0	92.5	57.6	91.5	36.5	72.5	51.2	16.0	28.5	33.0
2	1	Vic	f	6.0	94.0	60.0	95.5	39.0	75.4	51.9	15.5	30.0	34.0
3	1	Vic	f	6.0	93.2	57.1	92.0	38.0	76.1	52.2	15.2	28.0	34.0
4	1	Vic	f	2.0	91.5	56.3	85.5	36.0	71.0	53.2	15.1	28.5	33.0
…	…	…	…	…	…	…	…	…	…	…	…	…	…
96	7	other	m	1.0	89.5	56.0	81.5	36.5	66.0	46.8	14.8	23.0	27.0
97	7	other	m	1.0	88.6	54.7	82.5	39.0	64.4	48.0	14.0	25.0	33.0
98	7	other	f	6.0	92.4	55.0	89.0	38.0	63.5	45.4	13.0	25.0	30.0
99	7	other	m	4.0	91.5	55.2	82.5	36.5	62.9	45.9	15.4	25.0	29.0
100	7	other	f	3.0	93.6	59.9	89.0	40.0	67.6	46.0	14.8	28.5	33.5

101 rows × 13 columns

Рисунок 3.1 - Завантажений датафрейм

3.2 Перетворення даних

Перевіримо датасет на наявність пустих значень. Як побачимо, пустих значень нема.

```
In [239…  df.isna().any()

Out[239]: site        False
          Pop         False
          sex         False
          age         False
          hdlngth     False
          skullw      False
          totlngth    False
          taill       False
          footlgth    False
          earconch    False
          eye         False
          chest       False
          belly       False
          dtype: bool
```

Рисунок 3.2 - Перевірка на присутність пустих значень

Перетворимо категоріальні змінні у числові за допомогою класу LabelEncoder з модуля sklearn.preprocessing.

```
In [240…  from sklearn.preprocessing import LabelEncoder
          df.Pop = LabelEncoder().fit_transform(df.Pop)
          df.sex = LabelEncoder().fit_transform(df.sex)
          df
```

Out[240]:

site	Pop	sex	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye	chest	belly

Рисунок 3.3 - Перетворення категоріальних змінних в числові

Виконаємо масштабування даних. Для цього імпортуємо з sklearn.preprocessing клас MinMaxScaler.

```
In [241…  from sklearn.preprocessing import MinMaxScaler
          df.loc[:, 'age':] = MinMaxScaler().fit_transform(df.loc[:, 'age':])
          df
```

Out[241]:

	site	Pop	sex	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye	chest	belly
0	1	0	1	0.875	0.563107	0.559140	0.651163	0.363636	0.806818	0.885906	0.48	0.60	0.733333
1	1	0	0	0.625	0.485437	0.408602	0.767442	0.409091	0.693182	0.664430	0.64	0.65	0.533333
2	1	0	0	0.625	0.558252	0.537634	0.953488	0.636364	0.857955	0.711409	0.54	0.80	0.600000
3	1	0	0	0.625	0.519417	0.381720	0.790698	0.545455	0.897727	0.731544	0.48	0.60	0.600000
4	1	0	0	0.125	0.436893	0.338710	0.488372	0.363636	0.607955	0.798658	0.46	0.65	0.533333
…	…	…	…	…	…	…	…	…	…	…	…	…	…
96	7	1	1	0.000	0.339806	0.322581	0.302326	0.409091	0.323864	0.369128	0.40	0.10	0.133333
97	7	1	1	0.000	0.296117	0.252688	0.348837	0.636364	0.232955	0.449664	0.24	0.30	0.533333
98	7	1	0	0.625	0.480583	0.268817	0.651163	0.545455	0.181818	0.275168	0.04	0.30	0.333333
99	7	1	1	0.375	0.436893	0.279570	0.348837	0.409091	0.147727	0.308725	0.52	0.30	0.266667
100	7	1	0	0.250	0.538835	0.532258	0.651163	0.727273	0.414773	0.315436	0.40	0.65	0.566667

101 rows × 13 columns

Рисунок 3.4 - Масштабування даних

Переставимо колонку "sex" на перше місце.

```
In [242…
name = 'sex'
col = df.pop(name)
df.insert(0, name, col)
df
```

Out[242]:

	sex	site	Pop	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye	chest	belly
0	1	1	0	0.875	0.563107	0.559140	0.651163	0.363636	0.806818	0.885906	0.48	0.60	0.733333
1	0	1	0	0.625	0.485437	0.408602	0.767442	0.409091	0.693182	0.664430	0.64	0.65	0.533333
2	0	1	0	0.625	0.558252	0.537634	0.953488	0.636364	0.857955	0.711409	0.54	0.80	0.600000
3	0	1	0	0.625	0.519417	0.381720	0.790698	0.545455	0.897727	0.731544	0.48	0.60	0.600000
4	0	1	0	0.125	0.436893	0.338710	0.488372	0.363636	0.607955	0.798658	0.46	0.65	0.533333
...	...	...	...	...	...	...	...	...	...	...	...	...	...
96	1	7	1	0.000	0.339806	0.322581	0.302326	0.409091	0.323864	0.369128	0.40	0.10	0.133333
97	1	7	1	0.000	0.296117	0.252688	0.348837	0.636364	0.232955	0.449664	0.24	0.30	0.533333
98	0	7	1	0.625	0.480583	0.268817	0.651163	0.545455	0.181818	0.275168	0.04	0.30	0.333333
99	1	7	1	0.375	0.436893	0.279570	0.348837	0.409091	0.147727	0.308725	0.52	0.30	0.266667
100	0	7	1	0.250	0.538835	0.532258	0.651163	0.727273	0.414773	0.315436	0.40	0.65	0.566667

101 rows × 13 columns

Рисунок 3.5 - Переміщення колонки

Розділимо дані на аргументи та значення.

```
In [243…
df_x, df_y =  df.iloc[:, 1:], df.iloc[:, 0]
```

Рисунок 3.6 - Розділення даних на аргументи та значення

Ділимо дані на тренувальні та тестові для подальшої роботи. Імпортуємо модуль sklearn.model_selection та застосуємо функцію train_test_split. Розділимо набір даних на 80% навчальних та 20% тестових.

```
In [244…
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    df_x, df_y, test_size=0.2, train_size=0.8, random_state=11)
x_train.shape, x_test.shape
```

```
Out[244]: ((80, 12), (21, 12))
```

Рисунок 3.7 - Поділ інформації на тренувальну та тестову

3.3 KNN

Було обрано чотири методи K-Nearest Neighbors, Logistic Regression, Random Forest, SVM.

Зберігатимемо результати тестування моделей у списку results.

```
In [245…  results = []
```

Рисунок 3.8 - Список результатів

Для виконання роботи методу KNN імпортуємо sklearn.neighbors.KNeighborsClassifier та sklearn.model_selection.GridSearchCV.

```
In [246…  from sklearn.neighbors import KNeighborsClassifier
          from sklearn.model_selection import GridSearchCV
```

Рисунок 3.9 - Імпортування модулів

Визначимо, які варіанти параметрів найкраще вирішують дану задачу.

```
In [247…  classificator = KNeighborsClassifier()
          params = {'n_neighbors': range(1, 60)}
          grid_search = GridSearchCV(classificator, params, cv=10, verbose=1)
          grid_search.fit(x_train, y_train)
          knn = grid_search.best_estimator_
          knn

          Fitting 10 folds for each of 59 candidates, totalling 590 fits
Out[247]: ▼ KNeighborsClassifier
          KNeighborsClassifier(n_neighbors=1)
```

Рисунок 3.10 - Визначення найкращого параметра

Натренуємо модель з найкращим параметром.

```
In [248…  knn.fit(x_train, y_train)

Out[248]:  ▼ KNeighborsClassifier
           KNeighborsClassifier(n_neighbors=1)
```

Рисунок 3.11 - Тренування моделі K-Nearest Neighbors

Визначимо точність моделі на тренувальних та тестових даних.

```
In [249…  train_score = round(knn.score(x_train, y_train), 5)
          test_score = round(knn.score(x_test, y_test), 5)
          results.append({'method': 'knn', 'score': train_score, 'type': 'train'})
          results.append({'method': 'knn', 'score': test_score, 'type': 'test'})
          print(f'Train accuracy: {train_score}')
          print(f'Test accuracy: {test_score}')

          Train accuracy: 1.0
          Test accuracy: 0.47619
```

Рисунок 3.12 - Точність моделі K-Nearest Neighbors

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей. Для цього застосуємо sklearn.metrics.plot_confusion_matrix.

```
In [250…  from sklearn.metrics import confusion_matrix
          def conf_mat(model, x_test, y_test):
              y_predicted = model.predict(x_test)
              cm = confusion_matrix(y_test, y_predicted)
              plt.figure(figsize = (8,5))
              sns.heatmap(cm, annot=True, fmt=".1f")
              plt.xlabel('Predicted')
```

Рисунок 3.13 - Функція побудови матриці невідповідностей

Матриця має два рядки та дві колонки: перший ряд і перша колонка - це істинно позитивні значення: опосум має стать A, і модель визначила цю стать. Перший ряд і друга колонка - хибно позитивні, тобто опосум має стать A, але модель вказала на стать B; другий ряд і перша колонка - хибно негативні, тобто опосум має стать B, але модель вказала на стать A; другий ряд і друга колонка - істинно негативні, тобто опосум має стать B, і модель вказала на стать B.

```
In [251… conf_mat(knn, x_test, y_test)
```

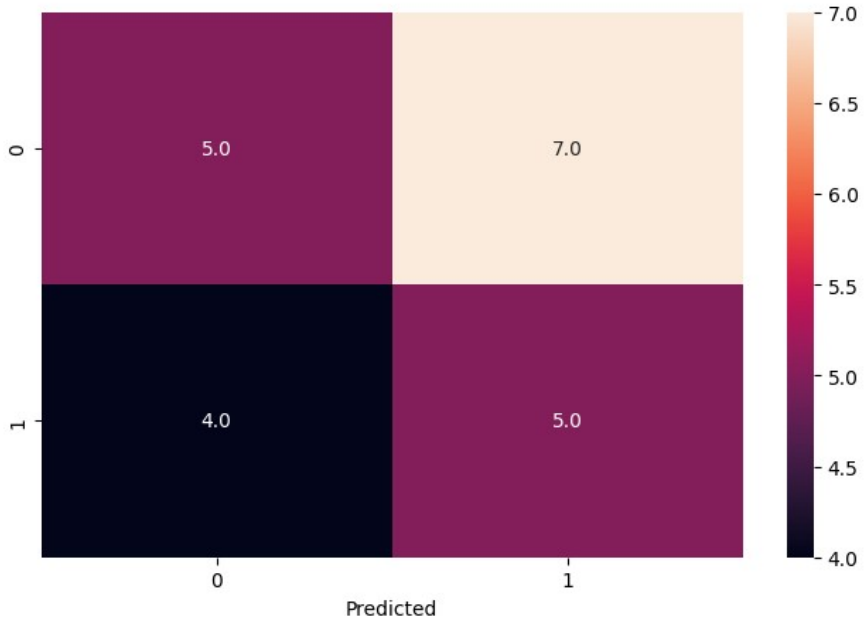

Рисунок 3.14 - Матриця невідповідностей для K-Nearest Neighbors

Побудуємо графік ROC(Receiver Operating Characteristic), що є графіком істинно позитивної відносної частоти проти хибно позитивної частоти. Це показує компроміс між чутливістю та специфічністю. Для цього імпортуємо sklearn.metrics.roc_curve та sklearn.metrics.roc_auc_score. До того ж визначимо AUC(Area Under the ROC Curve), що є мірою того, наскільки добре модель може розрізняти позитивні та негативні результати. Він коливається від 0 до 1, де 1 є найкращим класифікатором, а 0,5 – випадковим класифікатором. AUC корисний під час порівняння продуктивності різних класифікаторів на одному наборі даних, бо дає єдине число, яке підсумовує загальну продуктивність.

```
In [252… from sklearn.metrics import roc_curve, roc_auc_score
def roc(model, x_test, y_test):
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, _ = roc_curve(y_test,  y_pred_proba)
    auc = roc_auc_score(y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="Area = "+str(auc)+')')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc=4)
    plt.show()
```

Рисунок 3.15 - Імпортування модуля та визначення функції roc

Побудуємо ROC для K-Nearest Neighbors.

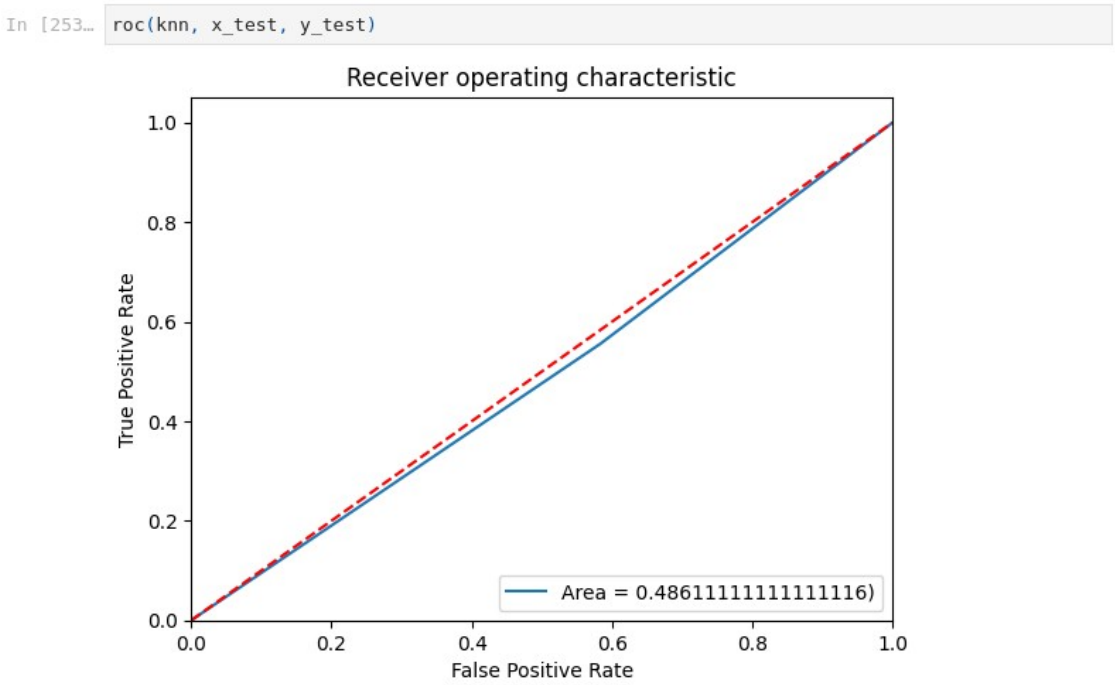

Рисунок 3.16 - Графік ROC для K-Nearest Neighbors

3.4 Logistic Regression

Для виконання роботи методу Logistic Regression імпортуємо sklearn.linear_model.LogisticRegression. Визначимо найкращі параметри моделі, передавши в неї параметри регуляризації.

In [254…

```
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
c = np.logspace(-4, 4, 60)
penalty = ['l1', 'l2']
params = dict(C=c, penalty=penalty)
log_reg = GridSearchCV(logisticRegr, params, cv=10, verbose=1)
log_reg.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 120 candidates, totalling 1200 fits
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Рисунок 3.17 - Тренування моделі Logistic Regression

Визначимо точність моделі на тренувальних та тестових даних.

```
In [255…
train_score = round(log_reg.score(x_train, y_train), 5)
test_score = round(log_reg.score(x_test, y_test), 5)
results.append({'method': 'logress', 'score': train_score, 'type': 'train'})
results.append({'method': 'logress', 'score': test_score, 'type': 'test'})
print(f'Train accuracy: {train_score}')
print(f'Test accuracy: {test_score}')

Train accuracy: 0.775
Test accuracy: 0.42857
```

Рисунок 3.18 - Точність моделі Logistic Regression

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

```
In [256…
conf_mat(log_reg, x_test, y_test)
```

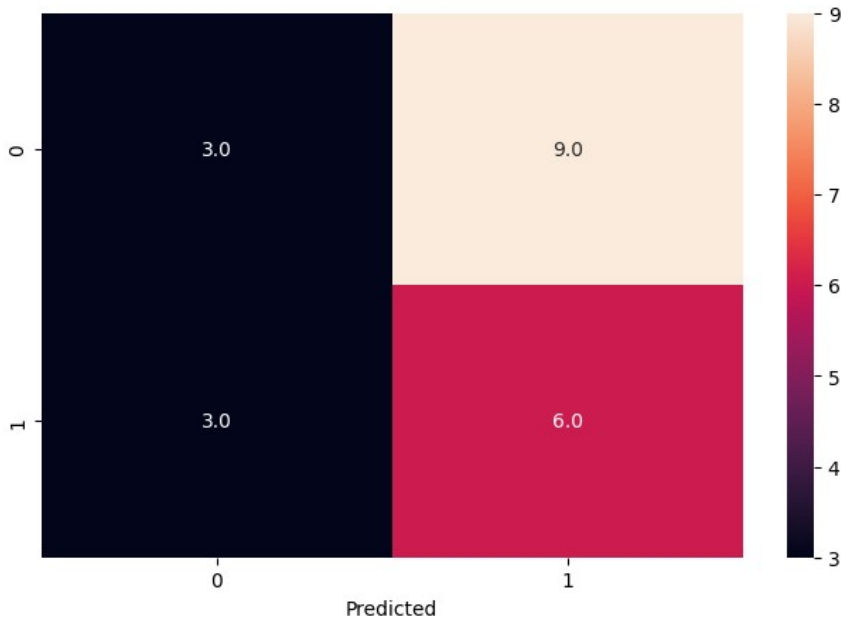

Рисунок 3.19 - Матриця невідповідностей для Logistic Regression

Побудуємо графік ROC для Logistic Regression.

```
In [257… roc(log_reg, x_test, y_test)
```

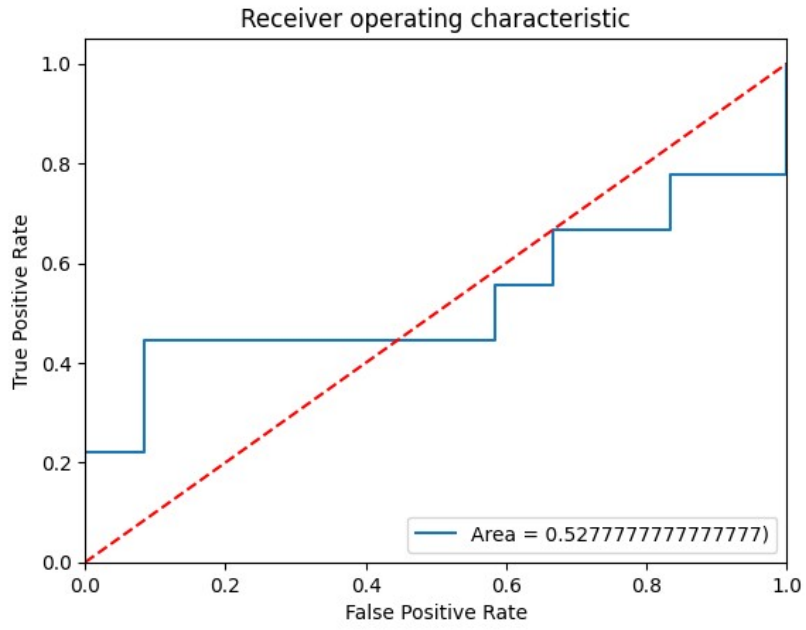

Рисунок 3.20 - Графік ROC для Logistic Regression

3.5 Random Forest

Для виконання роботи методу Random Forest імпортуємо sklearn.ensemble.RandomForestClassifier. Визначимо найкращі параметри для моделі. У випадку Random Forest параметри включають кількість дерев рішень та кількість характеристик, які враховуються кожним деревом під час поділу вузла і використовуються для поділу кожного вузла, отриманого під час навчання. Імпортуємо sklearn.model_selection.RandomizedSearchCV.

```
In [258… from sklearn.ensemble import RandomForestClassifier
         # Кількість дерев
         n_estimators = [int(x) for x in np.linspace(start = 10, stop = 300, num = 60)]
         params = {'n_estimators': n_estimators}
         rf = RandomForestClassifier()
         rf_random = GridSearchCV(rf, param_grid=params, cv=3, n_jobs=5)
         rf_random.fit(x_train, y_train)
```

```
Out[258]: ▸ GridSearchCV
          ▸ estimator: RandomForestClassifier
              ▸ RandomForestClassifier
```

Рисунок 3.21 - Тренування моделі Random Forest

Визначимо точність моделі на тренувальних та тестових даних.

```
In [259…
train_score = round(rf_random.score(x_train, y_train), 5)
test_score = round(rf_random.score(x_test, y_test), 5)
results.append({'method': 'rf', 'score': train_score, 'type': 'train'})
results.append({'method': 'rf', 'score': test_score, 'type': 'test'})
print(f'Train accuracy: {train_score}')
print(f'Test accuracy: {test_score}')

Train accuracy: 1.0
Test accuracy: 0.33333
```

Рисунок 3.22 - Точність моделі Random Forest

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

```
In [260…
conf_mat(rf_random, x_test, y_test)
```

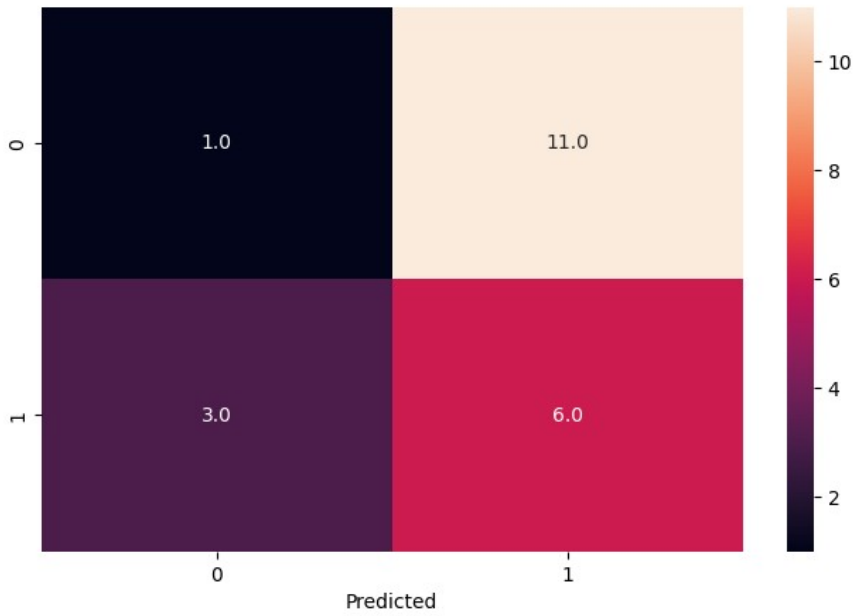

Рисунок 3.23 - Матриця невідповідностей для Random Forest

Побудуємо графік ROC для Logistic Regression.

```
In [261… roc(rf_random, x_test, y_test)
```

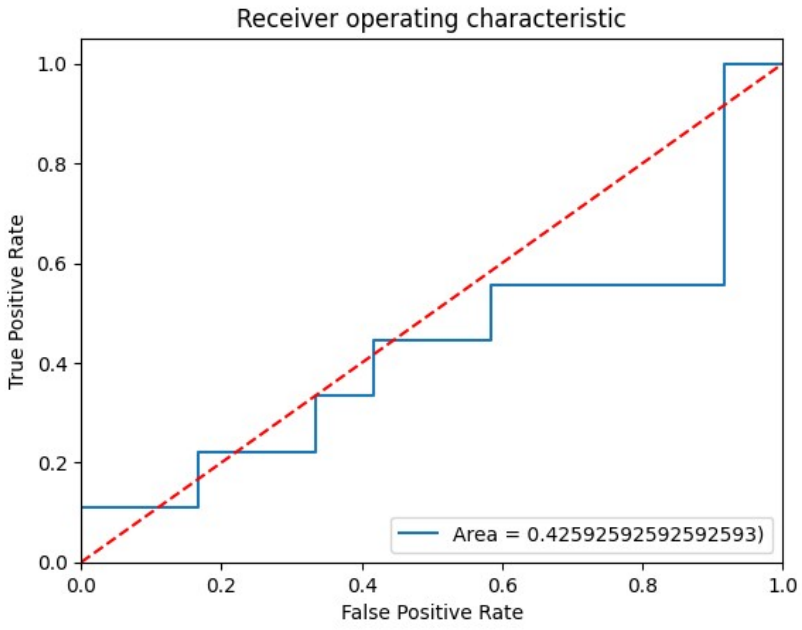

Рисунок 3.24 - Графік ROC для Random Forest

3.6 SVM

Для виконання роботи методу SVM імпортуємо sklearn.svm.SVC.

```
In [262… from sklearn.svm import SVC
c = [1, 5]
params = {'C': c, 'kernel': ['rbf', 'linear']}
svc = SVC(gamma='auto', probability=True)
svc_model = GridSearchCV(svc, param_grid=params, cv=3, n_jobs=5)
svc_model.fit(x_train, y_train)
```

Out[262]:

Рисунок 3.25 - Тренування моделі SVM

Визначимо точність моделі на тренувальних та тестових даних.

```
In [263…
train_score = round(svc_model.score(x_train, y_train), 5)
test_score = round(svc_model.score(x_test, y_test), 5)
results.append({'method': 'svm', 'score': train_score, 'type': 'train'})
results.append({'method': 'svm', 'score': test_score, 'type': 'test'})
print(f'Train accuracy: {train_score}')
print(f'Test accuracy: {test_score}')

Train accuracy: 0.8125
Test accuracy: 0.47619
```

Рисунок 3.26 - Точність моделі SVM

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

```
In [264…
conf_mat(svc_model, x_test, y_test)
```

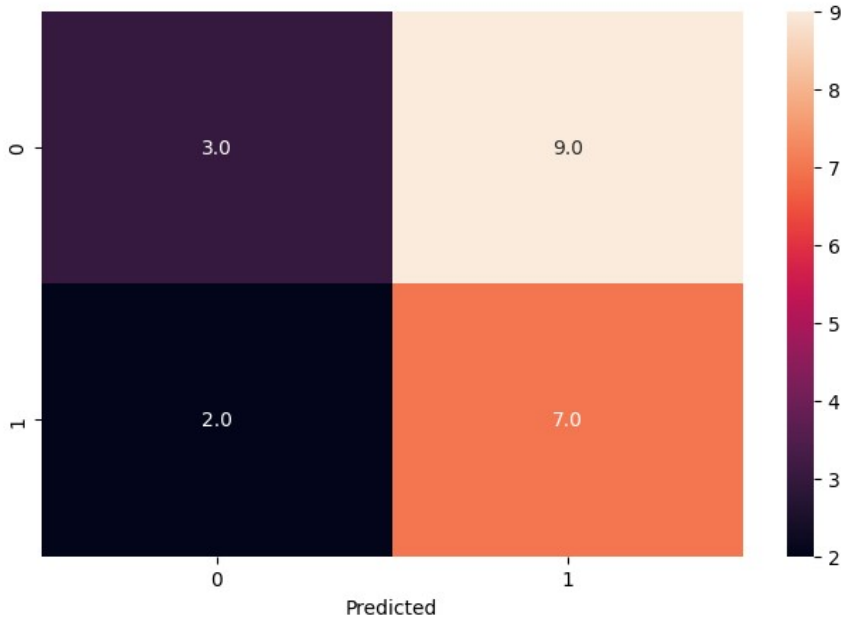

Рисунок 3.27 - Матриця невідповідностей для SVM

Побудуємо графік ROC для SVM.

```
In [265… roc(svc_model, x_test, y_test)
```

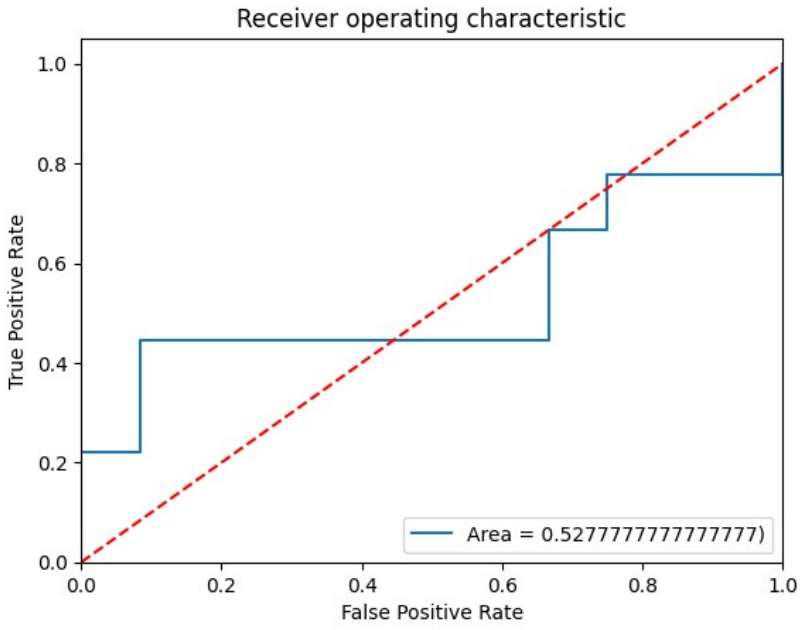

Рисунок 3.28 - Графік ROC для SVM

3.7 Порівняння результатів

Проаналізувавши окремо кожен із методів, проведемо порівняння даних методів.

```
In [266… df_score = pd.DataFrame(results, columns=['method','score','type'])
df_score
```

Out[266]:

	method	score	type
0	knn	1.00000	train
1	knn	0.47619	test
2	logress	0.77500	train
3	logress	0.42857	test
4	rf	1.00000	train
5	rf	0.33333	test
6	svm	0.81250	train
7	svm	0.47619	test

Рисунок 3.29 - Датафрейм результатів

Для наочності побудуємо гістограму.

```
In [267… sns.barplot(x='method', y='score', hue='type', data=df_score)

Out[267]: <AxesSubplot: xlabel='method', ylabel='score'>
```

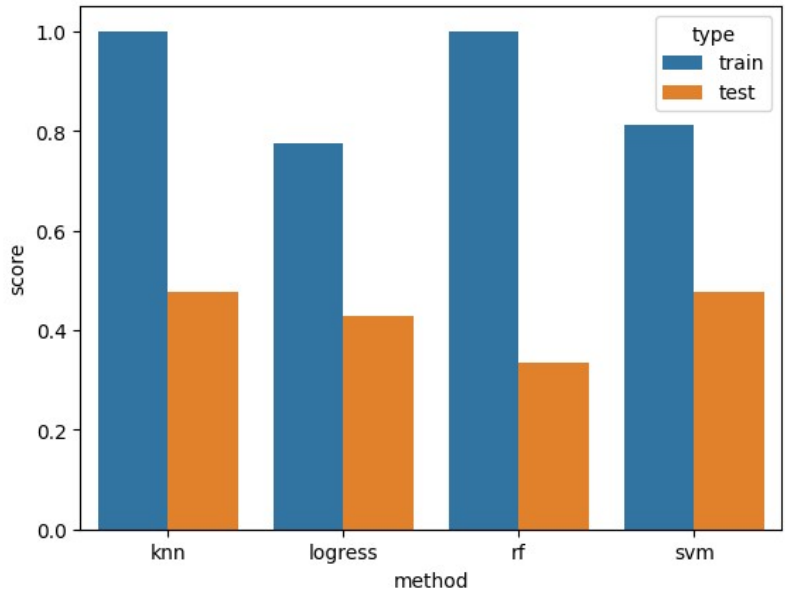

Рисунок 3.30 - Результати моделей

Як бачимо, усіма методами отримали доволі низьку точність. Найкраще себе показали KNN та SVM. Найгірше - Random Forest. Такі результати пояснюються тим, що а датасеті надзвичайно мало тренувальних даних, і їхню кількість треба суттєво збільшувати для досягнення вищих показників точності.

4 ВИСНОВОК

Під час виконання даної лабораторної роботи я ознайомився з побудовою моделей для вирішення задачі класифікації в scikit-learn, оцінкою та способами налаштування цих моделей.

Після аналізу даних встановлено, що методи SVM та KNN показують найкращі результати на тестових даних з точністю близько 47,6%. На тренувальних даних найкраще відпрацював Random Forest з точністю 100,0%, але на тестових даних його точність була найгіршою з результатом 33,33%. З іншої сторони, Logistic Regression показав себе найгірше на тренувальних даних з результатом 77,5%, але на тестових гірше за SVM та KNN і краще за Random Forest із 42,86%.

Оскільки датасет має надто мало даних, тому в результаті й отримали таку низьку точність. Для поліпшення ситуації треба мати достатньо великі набори даних.