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 №3

Прикладні задачі машинного навчання

Тема: Класифікація, регресія і кластеризація з використанням бібліотеки scikit-learn

Виконав
студент групи ІП-11:
Панченко С. В.

Перевірив:
Нестерук А. О

Київ 2023

ЗМІСТ

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – дослідити класифікацію, регресію і кластеризацію з використанням бібліотеки scikit-learn.

ЗАВДАННЯ

1. Повторити дії описані в пункті «Часові ряди і проста лінійна регресія частина 2» даної лабораторної роботи та порівняти з результатом попередньої лабораторної роботи.
2. Аналогічно з прикладом з лекції 7 згенеруйте набір даних та класифікуйте його використавши класифікатор SVC (слайд 95).
3. Порівняти декілька класифікаційних оцінювачів наприклад KNeighborsClassifier, SVC та GaussianNB для вбудованого в scikit-learn одного набору даних (вибрати довільний за бажанням).
4. Оцінити за формулою, якими могли б бути показники до 1895 року.
5. Зробити звіт про роботу.

ВИКОНАННЯ

Повторити дії описані в пункті «Часові ряди і проста лінійна регресія частина 2» даної лабораторної роботи та порівняти з результатом попередньої лабораторної роботи.

Завантажимо дані з ave_hi_nyc_jan_1895-2018.csv у датафрейм. Переназвемо колонки та застосуємо цілочисельне ділення, поділивши значенння років на 100.

In [111…
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
nyc = pd.read_csv('data/ave_hi_nyc_jan_1895-2018.csv')
nyc.columns = ['Date', 'Temperature', 'Anomaly']
nyc.Date = nyc.Date.floordiv(100)
nyc
```

Out[111]:

	Date	Temperature	Anomaly
0	1895	34.2	-3.2
1	1896	34.7	-2.7
2	1897	35.5	-1.9
3	1898	39.6	2.2
4	1899	36.4	-1.0
...	...	...	...
119	2014	35.5	-1.9
120	2015	36.1	-1.3
121	2016	40.8	3.4
122	2017	42.8	5.4
123	2018	38.7	1.3

124 rows × 3 columns

Рисунок 3.1 - Завантаження датасету

Розіб'ємо дані на навчальні та тестові. Оскільки оцінювачі scikit-learn вимагають, щоб в якості навчальних і тестових даних використовувалися двовимірні масиви, то застосуємо метод reshape та передамо в нього значення -1, 1б щоб перетворити їх з одновимірного масиву з n елементами в двовимірний масив з n рядками і одним стовпцем.

In [112…
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    nyc.Date.values.reshape(-1, 1),
    nyc.Temperature.values,
    random_state=11)
```

Рисунок 3.2 - Розбиття даних для навчання і тестування

Для перевірки пропорції навчальних і тестових даних (75% до 25%) задамо розміри X_train і X_test.

```
In [113… x_train.shape, x_test.shape

Out[113]: ((93, 1), (31, 1))
```

Рисунок 3.3 - Роміри навчальних і тестових даних (75% до 25%)

За допомогою оцінювача LinearRegression та простої лінійної регресії, що окремим випадком множинної лінійної регресії, навчимо модель.

```
In [114… from sklearn.linear_model import LinearRegression
         linear_regression = LinearRegression()
         linear_regression.fit(X=x_train, y=y_train)
```

Рисунок 3.4 - Тренування моделі

Виведемо кут нахилу, який зберігається в атрибуті coeff_ оцінювача.

```
In [115… linear_regression.coef_

Out[115]: array([0.01939167])
```

Рисунок 3.5 - Кут нахилу

Виведемо точку перетину, яка зберігається в атрибуті intercept_ оцінювача (m у формулі).

```
In [116… linear_regression.intercept_

Out[116]: -0.30779820252656265
```

Рисунок 3.6 - Точка перетину

Проведемо тестування моделі за даними з X_test і перевіримо прогнози по набору даних, виводячи прогнозовані і очікувані значення для кожного п'ятого елементу.

```
In [117…
predicted = linear_regression.predict(x_test)
expected = y_test
for p, e in zip(predicted[::5], expected[::5]):
    print(f'predicted: {p:.2f}, expected {e:.2f}')
```

```
predicted: 37.86, expected 31.70
predicted: 38.69, expected 34.80
predicted: 37.00, expected 39.40
predicted: 37.25, expected 45.70
predicted: 38.05, expected 32.30
predicted: 37.64, expected 33.80
predicted: 36.94, expected 39.70
```

Рисунок 3.7 - Тестування моделі

За допомогою кута нахилу і точки перетину зробимо прогнози для середньої температури в січні 2019 року, а також оцінки середньої температури в січні 1890 року.

```
In [118…
m = linear_regression.coef_
b = linear_regression.intercept_
predict = lambda x: m * x + b
```

Рисунок 3.8 - Лямбда вираз формули y = mx + b

Спрогнозуємо значення за 2019 рік.

```
In [119…
predict(2019)
```

```
Out[119]: array([38.84399018])
```

Рисунок 3.9 - Прогноз на 2019 рік

Спрогнозуємо значення за 1890 рік.

```
In [120…
predict(1890)
```

```
Out[120]: array([36.34246432])
```

Рисунок 3.10 - Прогноз на 1890 рік

Візуалізуємо набір даних з регресійними прямими.

Почнемо зі створення масиву, що містить мінімальні і максимальні значення дати з nyc.Date. Вони стануть координатами x початкової і кінцевої точок регресійної прямої.

```
In [121…  x = np.array([min(nyc.Date.values), max(nyc.Date.values)])
          x

Out[121]: array([1895, 2018])
```

Рисунок 3.11 - Масив даних

Передамо масив x функції predict та отримаємо пронозовані значення, які будуть використовуватися в якості координат y.

```
In [122…  y = predict(x)
          y

Out[122]: array([36.43942269, 38.82459851])
```

Рисунок 3.12 - Спрогнозовані значення

Побудуємо діаграму розкиду даних за допомогою функції scatterplot бібліотеки Seaborn і функції plot бібліотеки Matplotlib. Для виведення точок даних скористаємося методом scatterplot з колекцією DataFrame з ім'ям nyc. Змінимо масштаб осі. Зобразимо регресію.

```
In [135…  import seaborn as sns
          fig, ax = plt.subplots(1, 1, figsize=(6, 5))
          sns.scatterplot(data=nyc, x='Date', ax=ax,
                          y='Temperature', hue='Temperature',
                          palette='winter', legend=False)
          ax.set_ylim(10, 70)
          ax.plot(x, y)

Out[135]: [<matplotlib.lines.Line2D at 0x7f3a776c80d0>]
```

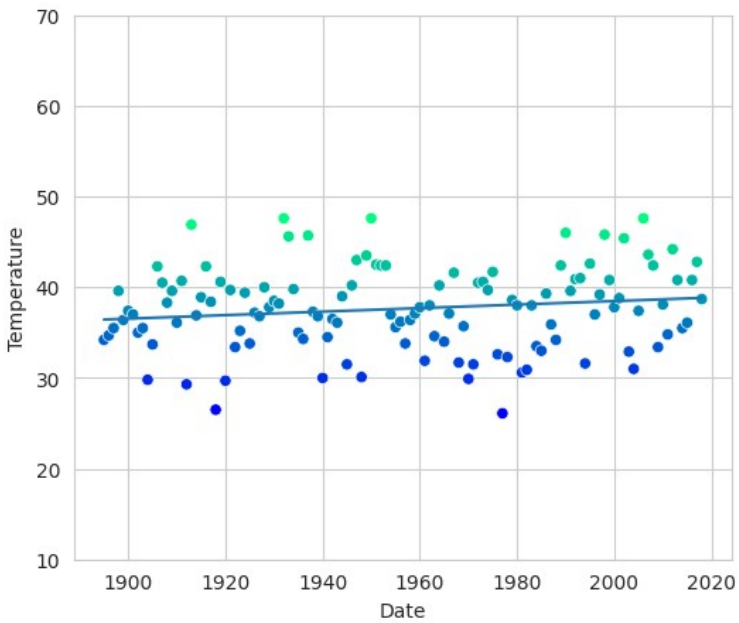

Рисунок 3.13 - Візуалізація з допомогою LinearRegression

Поглянемо на візуалізацію з минулої лабораторної та переконаємося, що обидві ідентичні.

```
In [137…
sns.set_style('whitegrid')
fig, ax = plt.subplots(1, 1, figsize=(6, 5))
sns.regplot(x=nyc.Date, y=nyc.Temperature, ax=ax)
ax.set_ylim(10, 70)
```

```
Out[137]: (10.0, 70.0)
```

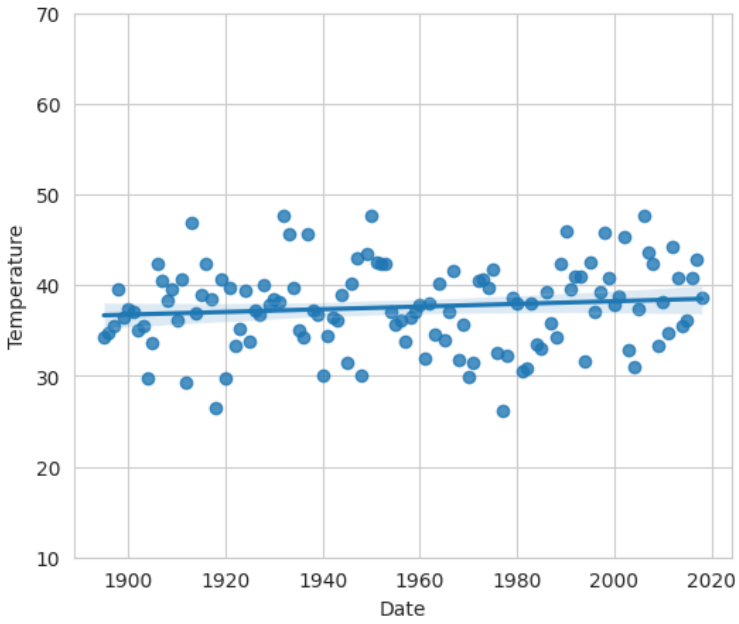

Рисунок 3.14 - Візуалізація минулої лабораторної

Як бачимо, візуалізації практично ідентичні.

з прикладом з лекції 7 згенеруйте набір даних та класифікуйте його використавши класифікатор SVC (слайд 95)

Для початку імпортуємо ListedColormap з matplotlib.colors та класифікатор SVC з sklearn.svm.

```
In [125…
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap
```

Рисунок 3.15 - Імпортування модулів

За допомогою бібліотеки NumPy згенеруємо набір даних, передавши

початковий seed 1 та використавши функції np.random.randn(згенерує матрицю з 200-ми рядками та 2-ма стовпцями), np.logical_xor(застосовує операцію виключного або), np.where(приймає в себе логічну операцію xor та повертає значення 1, якщо True, -1, якщо False).

```
In [126…  x_xor = np.random.randn(200, 2)
          y_xor = np.logical_xor(x_xor[:, 0] > 0, x_xor[:, 1] > 0)
          y_xor = np.where(y_xor, 1, -1)
```

Рисунок 3.16 - Генерація даних

Зобразимо згенеровані дані. За допомогою функції scatter, спочатку синіми хрестиками зобразимо ті точки, для яких y_xor == 1. Потім ті точки, для яких y_xor == -1. Значення 0 та 1 у других індексах - це відповідно порядкові номери стовпчиків.

```
In [140…  fig, ax = plt.subplots(1, 1, figsize=(6, 5))
          ax.scatter(x_xor[y_xor == 1, 0], x_xor[y_xor == 1, 1],
                     c='b', marker='x', label='1')
          ax.scatter(x_xor[y_xor == -1, 0], x_xor[y_xor == -1, 1],
                     c='r', marker='s', label='-1')
          ax.set_xlim([-3, 3])
          ax.set_ylim([-3, 3])
          ax.legend(loc='best')
```

```
Out[140]: <matplotlib.legend.Legend at 0x7f3a7727bdc0>
```

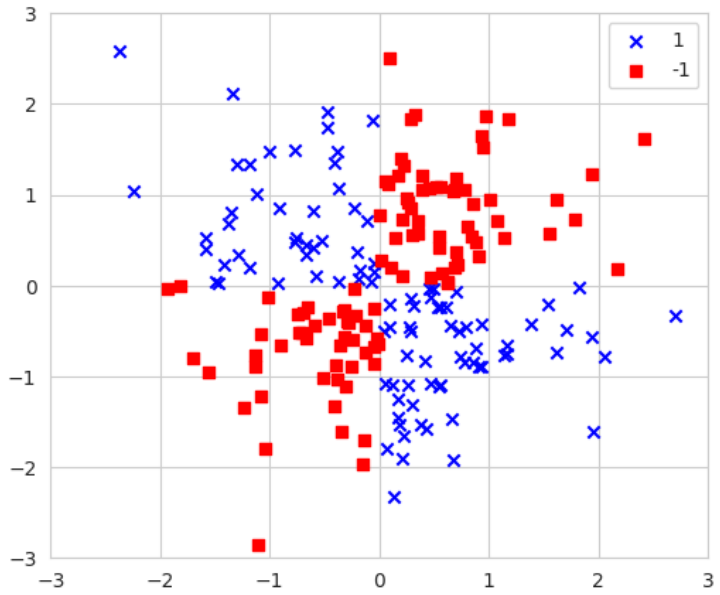

Рисунок 3.17 - Візуалізація згенерованих даних

Запишемо функцію plot_decision_regions, яка приймає в параметри: X -

двовимірний масив аргуметів, y - одновимірний масив значень, test_idx=None - тестовий індекс, resolution=0.02 - масштаб.

Рефактиремо функцію, надану в лекції. "unq" - масив унікальних значень "y", а "unql" - довжина "unq".

Викличемо IndexError, якщо передані масиви "markers" або "colors" менші за разміром ніж "unq".

cmap - екземпляр класу ListedColormap, у який ми передали кольори до індекса "unql"; x1_min, x1_max - відповідно мінімальне і максимальне значення значення першого стовпчика, зміщені на 1 для кращої видимості; x2_min, x2_max - для другого стовпчика відповідно.

xx1, xx2 - масиви рівномірно розподілених значень між мінімальним та максимальним з кроком в resolution за допомогою функції numpy.arange; xx1, xx2 - це перетворені попередні xx1 та xx2 за допомогою функції numpy.meshgrid, яка робить сітку індексів з одновимірних масивів; xx1_flat, xx2_flat - сплющені до одновимірних двовимірні масив xx1, xx2 за допомогою методу ravel; xx_t - матриця, стовпчиками якої є xx1_flat та xx2_flat.

z - спрогнозовані класифікатором значення, у метод predict якого передаємо xx_t.

За допомогою plt.contourf зображаємо контури класів, передаючи у функцію xx1, xx2 та кольорову мапу. За допомогою xlim та ylim, у які передаємо мінімальні та максимальні значення, обмежуємо візуалізацію для зручного сприйняття.

У циклі для кожного класу вимальовуємо свої точки, передаючи у функцію scatter їхню позицію, колір, маркер, кольор контуру, позначку.

Якщо тестовий індекс не пустий, то вимальовуємо точки з x_test.

In [148…
```
def plot_decision_regions(x, y, classifier, markers, colors, w, h,
                          test_idx=None, resolution=0.02):
    unq = np.unique(y)
    unql = len(unq)
    if len(markers) < unql or len(colors) < unql:
        raise IndexError
    fig, ax = plt.subplots(1, 1, figsize=(w, h))
    cmap = ListedColormap(colors[:unql])
    x1_min, x1_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    x2_min, x2_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx1 = np.arange(x1_min, x1_max, resolution)
    xx2 = np.arange(x2_min, x2_max, resolution)
    xx1, xx2 = np.meshgrid(xx1, xx2)
    xx1_flat, xx2_flat = xx1.ravel(), xx2.ravel()
    xx_t = np.array([xx1_flat, xx2_flat]).T
    z = classifier.predict(xx_t)
    z = z.reshape(xx1.shape)
    ax.contourf(xx1, xx2, z, alpha=0.3, cmap=cmap)
    ax.set_xlim(xx1.min(), xx1.max())
    ax.set_ylim(xx2.min(), xx2.max())
    for idx, c1 in enumerate(unq):
        params = dict(c=colors[idx], marker=markers[idx], label=c1,
                      edgecolor='black')
        ax.scatter(x=x[y == c1, 0], y=x[y == c1, 1], **params)
    if not test_idx:
        return ax
    params = dict(marker='o', edgecolor='black', s=100)
    x_test = x[test_idx, :]
    ax.scatter(x_test[:, 0], x_test[:, 1], **params)
    return ax
```

Рисунок 3.18 - Функція plot_decision_regions

Зобразимо результати.

In [149…
```
markers = ('s', 'x', 'o', '^', 'v')
colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
svm = SVC(kernel='rbf', random_state=1, gamma=0.1, C=10)
svm.fit(x_xor, y_xor)
ax = plot_decision_regions(x_xor, y_xor, svm, markers, colors, 7, 5)
```

```
/tmp/ipykernel_43616/1281675792.py:24: UserWarning: You passed a edgecolor/edgecolors ('bl
ack') for an unfilled marker ('x').  Matplotlib is ignoring the edgecolor in favor of the
facecolor.  This behavior may change in the future.
  ax.scatter(x=x[y == c1, 0], y=x[y == c1, 1], **params)
```

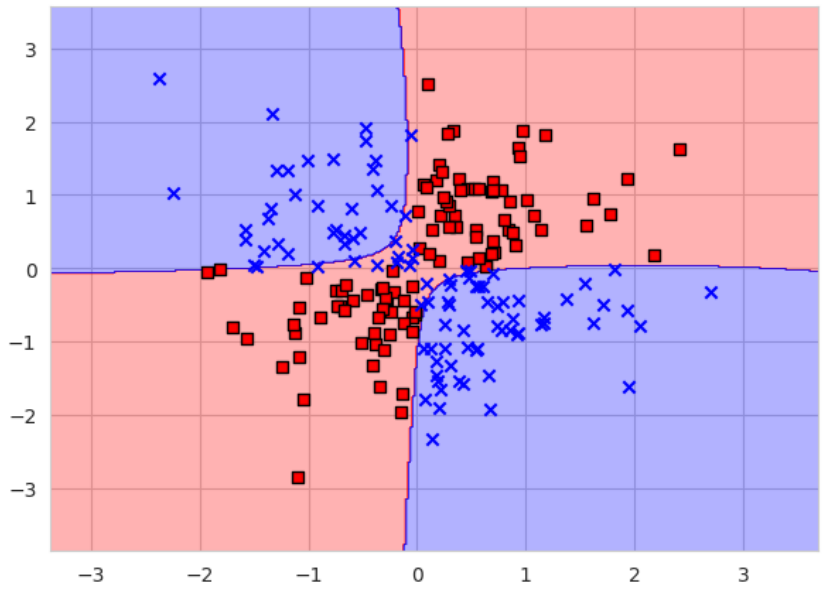

Рисунок 3.19 - Візуалізація роботи класифікатора SVC

декілька класифікаційних оцінювачів наприклад KNeighborsClassifier, SVC та GaussianNB для вбудованого в scikit-learn одного набору даних (вибрати довільний за бажанням)

Обиремо датасет вин та виведемо його.

```
In [210…
from sklearn.datasets import load_wine
pd.options.display.max_rows = 10
pd.options.display.max_columns = 7
wine = load_wine()
wine_df = pd.DataFrame(wine.data, columns=wine.feature_names)
wine_df
```

Out[210]:

	alcohol	malic_acid	ash	...	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	...	1.04	3.92	1065.0
1	13.20	1.78	2.14	...	1.05	3.40	1050.0
2	13.16	2.36	2.67	...	1.03	3.17	1185.0
3	14.37	1.95	2.50	...	0.86	3.45	1480.0
4	13.24	2.59	2.87	...	1.04	2.93	735.0
...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	...	0.64	1.74	740.0
174	13.40	3.91	2.48	...	0.70	1.56	750.0
175	13.27	4.28	2.26	...	0.59	1.56	835.0
176	13.17	2.59	2.37	...	0.60	1.62	840.0
177	14.13	4.10	2.74	...	0.61	1.60	560.0

178 rows × 13 columns

Рисунок 3.20 - Датасет вин

Розділемо датасет на тренувальну та тестові частини. Нехай буде 80% тренувальних та 20% тестових даних.

```
In [189…
x_train, x_test, y_train, y_test = train_test_split(
    wine.data, wine.target, test_size=0.2, train_size=0.8, random_state=11)
```

Рисунок 3.21 - Навчальні та тестові дані

Ініціалізуємо список results, у який будемо додавати результати моделей.

```
In [190…
results = []
```

Рисунок 3.22 - Список результатів моделей

Спочатку застосуємо модель K-Nearest Neighbors та за допомогою

GridSearchCV знайдемо оптимальну модель.

```
In [191…
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

Рисунок 3.23 - Імпортування модулів

Визначимо, які варіанти параметрів найкраще вирішують дану задачу, підбираючи оптимальну кількість сусідів.

```
In [195…
classificator = KNeighborsClassifier()
params = {'n_neighbors': range(1, 10)}
grid_search = GridSearchCV(classificator, params)
grid_search.fit(x_train, y_train)
knn = grid_search.best_estimator_
knn
```

```
Out[195]:
▼ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

Рисунок 3.24 - Визначення найкращого параметра

Визначимо точність моделі на тренувальних та тестових даних

```
In [196…
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: 0.80556
```

Рисунок 3.25 - Точність моделі K-Nearest Neighbors

Візуалізуємо отримані результати. Побудуємо графік, де маємо сині плюси - прогнозовані значення, жовті кружечки - фактичні значення. Якщо жовті кружечки не закриваються синіми плюсами, то це означає, що в цих місцях модел допустила помилку.

```
In [202…
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.axes.get_yaxis().set_visible(False)
x_test_df = pd.DataFrame(x_test)
ax.plot(x_test_df.index, y_test, 'yo', label='True result')
ax.plot(x_test_df.index, knn.predict(x_test),
         'b+', label='Predicted result')
ax.legend(loc='upper right', shadow=True)
```

```
Out[202]: <matplotlib.legend.Legend at 0x7f3a770fd390>
```

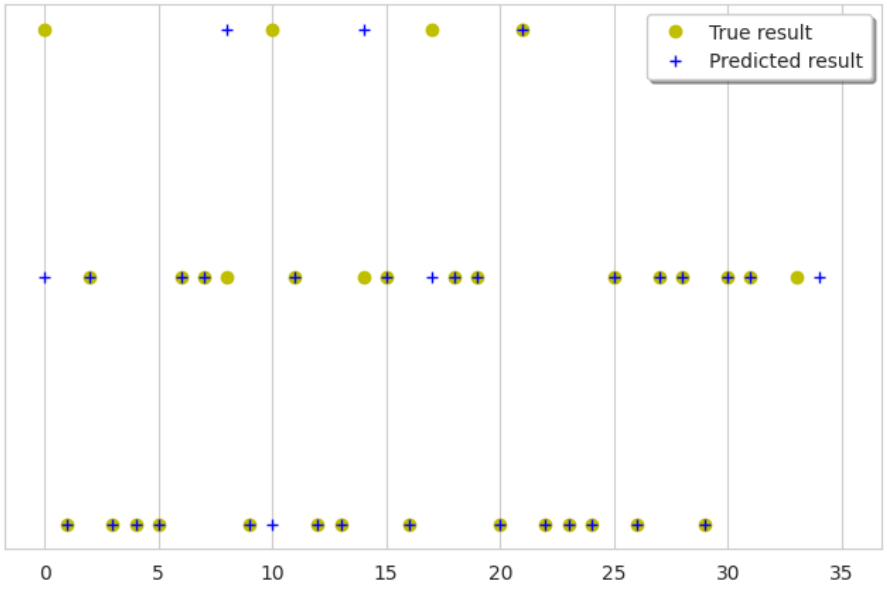

Рисунок 3.26 - Візуалізація точності результатів K-Nearest Neigbors

Далі застосуємо модель SVC. Ядрами для SVC будуть "rbf", "linear". Задамо параметр C, який контролює трейд-офф між гладкістю вирішальної границі та правильною класифікацією точок. Де чим більше C, тим краще класифікуються точки.

```
In [203…
from sklearn.svm import SVC
params = {'C': [1, 5], 'kernel': ['rbf', 'linear']}
svc = SVC(gamma='auto', probability=True)
svc_model = GridSearchCV(svc, param_grid=params)
svc_model.fit(x_train, y_train)
```

```
Out[203]:
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

Рисунок 3.27 - Тренування моделі SVM

Визначимо точність моделі на тренувальних та тестових даних.

```
In [204…
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: 1.0
Test accuracy: 0.97222
```

Рисунок 3.28 - Точність моделі SVM

Візуалізуємо отримані результати.

```
In [205…
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.axes.get_yaxis().set_visible(False)
x_test_df = pd.DataFrame(x_test)
ax.plot(x_test_df.index, y_test, 'yo', label='True result')
ax.plot(x_test_df.index, svc_model.predict(x_test),
         'b+', label='Predicted result')
ax.legend(loc='upper right', shadow=True)
```

```
Out[205]: <matplotlib.legend.Legend at 0x7f3a770fc820>
```

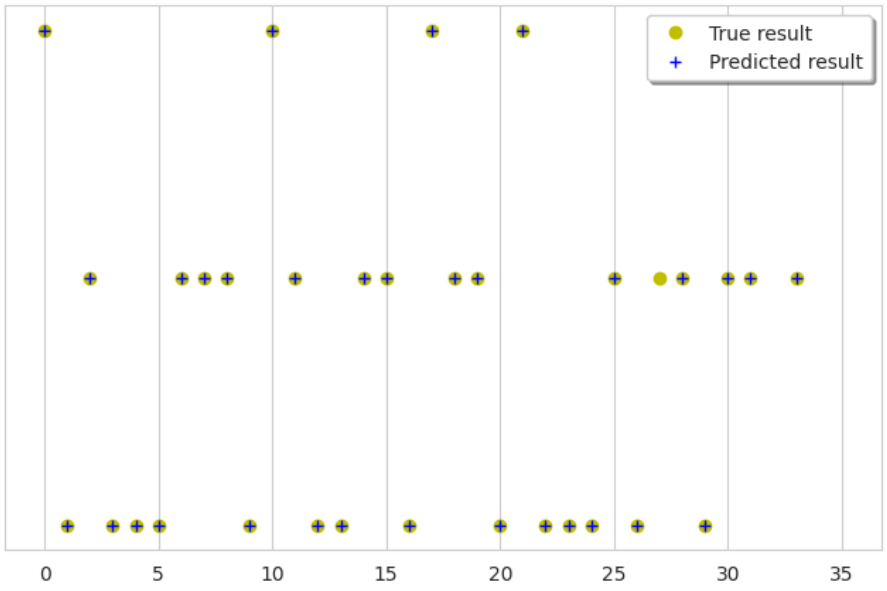

Рисунок 3.29 - Візуалізація точності результатів SVC

Далі застосуємо модель Random Forest. n-estimators - кількість дерев у лісі. За допомогою функції linspace утворимо рівномірно розподілений список кількість дерев у заданих межах.

```
In [206…
from sklearn.ensemble import RandomForestClassifier
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 300, num = 60)]
params = {'n_estimators': n_estimators}
rf = RandomForestClassifier()
rf_random = GridSearchCV(rf, param_grid=params)
rf_random.fit(x_train, y_train)
```

Out[206]:

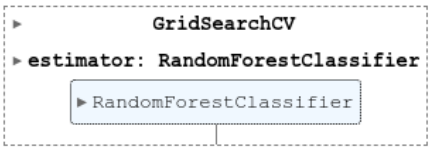

Рисунок 3.30 - Тренування моделі Random Forest

Визначимо точність моделі на тренувальних та тестових даних.

```
In [207…
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: 0.97222
```

Рисунок 3.31 - Точність моделі Random Forest

Візуалізуємо отримані результати.

```
In [208…
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.axes.get_yaxis().set_visible(False)
x_test_df = pd.DataFrame(x_test)
ax.plot(x_test_df.index, y_test, 'yo', label='True result')
ax.plot(x_test_df.index, rf_random.predict(x_test),
         'b+', label='Predicted result')
ax.legend(loc='upper right', shadow=True)
```

Out[208]: <matplotlib.legend.Legend at 0x7f3a76fad390>

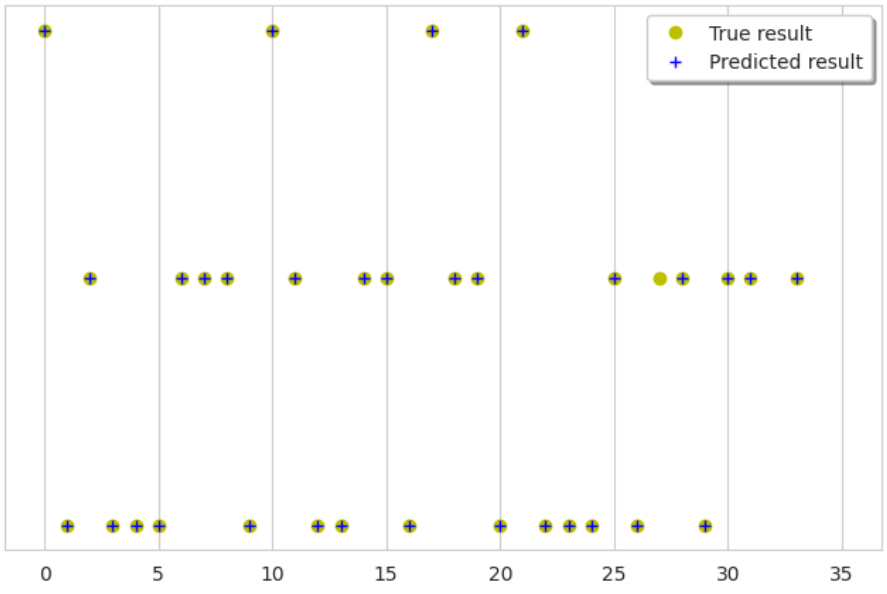

Рисунок 3.32 - Візуалізація точності результатів Random Forest

Порівняємо отримані результати моделей. Утворимо датафрейм результатів.

```
In [211… df_score = pd.DataFrame(results, columns=['method','score','type'])
df_score
```

Out[211]:

	method	score	type
0	knn	1.00000	train
1	knn	0.80556	test
2	knn	1.00000	train
3	knn	0.80556	test
4	svm	1.00000	train
5	svm	0.97222	test
6	rf	1.00000	train
7	rf	0.97222	test

Рисунок 3.33 - Датафрейм результатів

Для наочності побудуємо гістограму.

```
In [213… fig, ax = plt.subplots(1, 1, figsize=(8, 5))
sns.barplot(x='method', y='score', hue='type', data=df_score, ax=ax)
```

```
Out[213]: <AxesSubplot: xlabel='method', ylabel='score'>
```

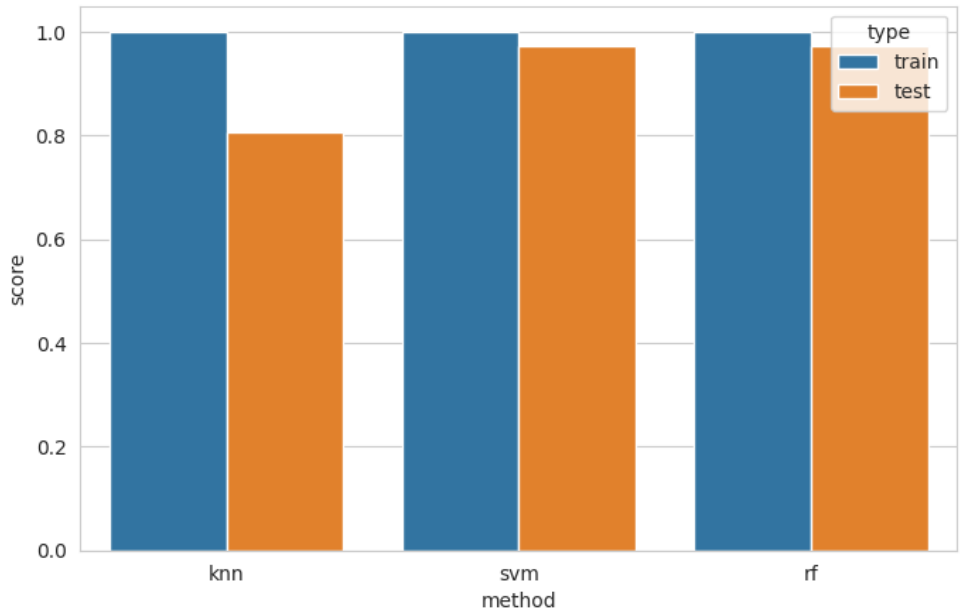

Рисунок 3.34 - Результати моделей

З огляду бачимо, що на тренувальних даних усі методи відпрацювали однаково. Однак на тестових даних KNN показує себе найгірше. Бачимо, що SVC

та RandomForest показали себе однаково на тестових даних, але треба зауважити, що вибірка була доволі малою, тому може бути не видно суттєвої різниці.

ВИСНОВОК

Під час виконання цієї лабораторної роботи здобув базові навички класифікації, регресії і кластеризації з використанням бібліотеки scikit-learn. Було створено лінійну регресію з допомогою класу LinearRegression та порівняно її з попередньою з другої лабораторної роботи, результати виявилися ідентичними. Потім було згенеровано та класифіковано згенерований набір даних з використанням класифікатора SVC. У кінці порівняв три класифікатори: K-Neare st Neighbors, SVC та Random Forest. KNN показав себе найгірше на тестових даних, SVC та Random Forest виявили однакову точність.