МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Івано-Франківський національний технічний університет нафти і газу

Кафедра комп'ютеризованого машинобудування

В. Б. Копей, О. Р. Онисько

ОСНОВИ ШТУЧНОГО ІНТЕЛЕКТУ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Івано-Франківськ
2024

УДК 004.85
К 65

Рецензент:

Кіт Г. В., кандидат технічних наук, доцент, завідувач кафедри інформаційних технологій та програмування Івано–Франківської філії Відкритого міжнародного університету розвитку людини "Україна"

Рекомендовано методичною радою університету
(протокол №6/1 від 20 червня 2024 р.)

К65 **Копей В. Б., Онисько О. Р.** Основи штучного інтелекту : лабораторний практикум / В. Б. Копей, О. Р. Онисько – Івано-Франківськ : ІФНТУНГ, 2024. – 59 с.

МВ 02070855-20468-2024

Лабораторний практикум містить приклади використання популярного Python-пакету scikit-learn в задачах машинного навчання. Розглянуто задачі: машинного навчання з учителем (регресія та класифікація), машинного навчання без учителя (кластеризація та зменшення розмірності), запобігання перенавчання моделей, підготовки даних, кодування і відбору ознак, оптимізації гіперпараметрів. Призначено для вивчення дисципліни «Основи штучного інтелекту» під час підготовки фахівців другого (магістерського) рівня освіти.

УДК 004.85

МВ 02070855-20468-2024

© Копей В. Б., 2024
© ІФНТУНГ, 2024

ЗМІСТ

ВСТУП

Штучний інтелект в широкому розумінні – це інтелект машин, тобто їхня здатність виконувати творчі інтелектуальні функції, як це робить людина. Як галузь знань штучний інтелект має чимало підходів і напрямків [1], зокрема напрямок машинного навчання (МН), що вивчає методи створення алгоритмів, які здатні навчатися. МН поділяють на дедуктивне та індуктивне.

Дедуктивне машинне навчання основане на **формалізації** знань експертів у вигляді бази знань [1].

Індуктивне машинне навчання [2, 3] (навчання за прецедентами) основане на виявленні **загальних** емпіричних закономірностей (зв'язків, залежностей, моделей) за **частковими** даними (прецедентами). Наприклад є часткові дані про середню температуру повітря, отримані шляхом вимірювання (таблиця 1).

Таблиця 1 – Опис прецедентів (вибірка для навчання)

Місяць року, x	1	2	3	4	5
Температура повітря, y (°C)	3	3	7	13	15

Результатом МН може бути регресійна модель у вигляді функції

$$ax+b \approx y,$$

де x – незалежна змінна (ознака), y - залежна змінна, a і b - параметри моделі. Правильно побудована модель (a = 3,4; b = −2) дозволяє прогнозувати **приблизну** температуру повітря у перші п'ять місяці року. Наприклад у третій: 3,4·3−2=8,2 °C. Зверніть увагу, що для цієї моделі прогноз на інші місяці може бути зовсім не правильний. Наприклад у дванадцятий: 3,4·12−2=38,8 °C.

У методичних вказівках наведено приклади програм мовою Python [4-6] для вивчення основ популярного пакету для машинного навчання scikit-learn [7]. Приклади основані на версії scikit-learn 0.19 та вільно доступні на GitHub за адресою https://github.com/vkopey/sklearn_examples. Для запуску прикладів в новіших версіях scikit-learn (наприклад в середовищі Google Colab)

може знадобитись незначна модифікація коду. Необхідну для цього інформацію можна легко знайти в Інтернеті, зокрема на stackoverflow.com.

Глибшу інформацію про scikit-learn та машинне навчання можна отримати з офіційної документації [7] та книг [1-3, 8-14].

1 ЛІНІЙНА РЕГРЕСІЯ

Регресійні моделі та моделі класифікації є моделями машинного навчання **з учителем**, коли для створення моделі використовуються значення незалежної змінної *x* з відповідними значеннями залежної змінної *y* (табл. 1). Алгоритм ніби навчає модель: «цьому значенню *x* відповідає це значення *y*, а цьому – це, і т.д.». У регресійної моделі прогноз є неперервною величиною, а у моделі класифікації – дискретною.

В прикладі linreg.py за допомогою пакету scikit-learn та класу `LinearRegression` побудовано одновимірну **лінійну регресійну модель** даних, поданих масивами `x` та `y`. Функція `fit()` шукає лінійну залежність у вигляді $ax+b$ **методом найменших квадратів**. Програма виводить значення коефіцієнтів a і b, коефіцієнта детермінації R^2 та будує графік з даними та моделлю. Якщо значення R^2 близьке до 1 то, модель добре описує дані та може бути використана для прогнозів. Якщо значення R^2 близьке до 0, то слід шукати більш якісну модель, наприклад, серед поліноміальних нелінійних моделей.

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# дані:
x = np.array([8,0,3,4,9,7,1,6,3,9])
y = np.array([9,0,2,6,9,8,2,9,4,9])
x=x[:, None] # або x.reshape(-1,1) або x.reshape(10,1)
```

```
model = LinearRegression() # модель: лінійна регресія
model.fit(x, y) # підігнати модель (навчання або пошук
коефіцієнтів)
print "a=%f b=%f"%(model.coef_[0], model.intercept_) #
коефіцієнти моделі
print "R2=%f"%model.score(x,y) # коефіцієнт детермінації

X = np.linspace(0, 10, 100) # нові дані X
Y = model.predict(X[:, None]) # прогноз для X

plt.scatter(x, y) # емпіричні дані
plt.plot(X, Y) # модель
plt.xlabel('x'),plt.ylabel('y')
plt.show()
```

Результат:

```
a=1.020833 b=0.695833
R2=0.896431
```

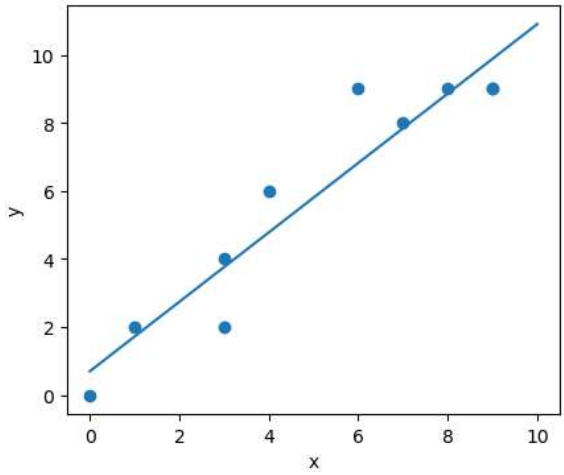


Рисунок 1 – Лінійна регресія

2 ДВОВИМІРНА ЛІНІЙНА РЕГРЕСІЯ

Наступний приклад linreg2D.py теж використовує модель лінійної регресії, але замість однієї ознаки є дві. Тому масив `x` містить два рядки (стовпчики, після транспонування), які відповідають ознакам x_0 та x_1. Функція `fit()` шукає лінійну залежність у вигляді ax_0+bx_1+c методом найменших квадратів. Програма виводить значення коефіцієнтів (a, b, c), коефіцієнта детермінації R^2 та будує графік з даними та моделлю.

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = np.array([[0,1,2,0,1,2,0,1,2],
              [0,0,0,1,1,1,2,2,2]])
y = np.array([0,1,2,1,2,3,2,3,9])
x=x.T
model = LinearRegression()
model.fit(x, y)

X=np.mgrid[0:3:0.5,0:3:0.5]
X_=X.reshape((2,X.size//2))
Y = model.predict(X_.T)
print model.coef_, model.intercept_, model.score(x,y)

from mpl_toolkits.mplot3d import axes3d
fig = plt.figure() # створити фігуру
ax = fig.add_subplot(111, projection='3d') # додати графік
3D
Y=Y.reshape(X.shape[1:])
ax.scatter(x[:,0], x[:,1], y) # показати емпіричні точки
ax.plot_wireframe(X[0], X[1], Y, rstride=1, cstride=1) #
показати теоретичну поверхню
```

```
ax.set_xlabel('X0'),ax.set_ylabel('X1'),ax.set_zlabel('Y')
plt.show()
```

Результати:

```
[1.83333333 1.83333333] -1.1111111111111125
0.7438524590163935
```

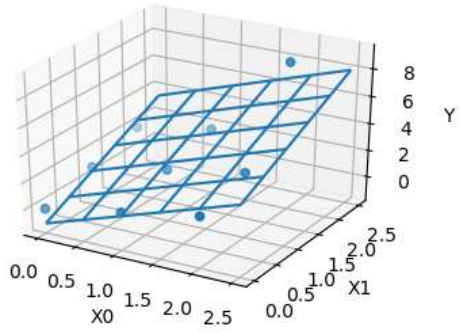


Рисунок 2 – Двовимірна лінійна регресія

3 РЕГУЛЯРИЗАЦІЯ ЛІНІЙНИХ МОДЕЛЕЙ

Перенавчанням (overfitting) називають явище, коли модель добре прогнозує дані, що використовувались для навчання, але погано прогнозує нові дані. З метою зменшення перенавчання намагаються зменшити коефіцієнти лінійної моделі (кут нахилу прямої) шляхом збільшення параметра **регуляризації** `alpha`>=0 (`alpha`=0 відповідає звичайній LinearRegression) [8]. Зменшення кута нахилу прямої робить модель більш «скромною» і подібною на модель середнього (горизонтальна пряма), яка взагалі не схильна до перенавчання, проте схильна до **недонавчання**. В прикладі linregRidgeLasso.py показано побудову лінійних моделей різними методами регуляризації: `Ridge` (регуляризація Тихонова `L1`), `Lasso` (`L2`) і `ElasticNet` (комбінована `L1 i L2`).

Для пошуку шляхом перехресної перевірки найкращого значення `alpha` можуть бути застосовані `RidgeCV` або `ElasticNetCV`.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.linear_model import Ridge, Lasso, RidgeCV,
ElasticNet

# дані
x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])

# лінійна регресія методом регуляризації Тихонова
model = Ridge(alpha = 0.5)
model.fit(x, y)
print model.coef_, model.intercept_, model.score(x,y)
##
# інший метод регуляризації - деякі коефіцієнти моделі
можуть бути рівні 0
model = Lasso(alpha = 0.5)
model.fit(x, y)
print model.coef_, model.intercept_, model.score(x,y)
##
# комбінована L1 і L2 регуляризація
# якщо l1_ratio = 1 то це L1 регуляризація
model = ElasticNet(alpha=0.5, l1_ratio = 0.5)
model.fit(x, y)
print model.coef_, model.intercept_, model.score(x,y)
##
# лінійна регресія Ridge (автоматично знаходить найкраще
alpha)
model = RidgeCV(alphas=[0.1, 1.0, 10.0])
model.fit(x, y)
print model.coef_, model.intercept_, model.score(x,y)
print model.alpha_
# див. також ElasticNetCV
```

Результати:

```
[1.01554404] 0.7222797927461135 0.8964066389985597
[0.96875] 0.9562500000000007 0.8940972222222223
[0.96954315] 0.9522842639593909 0.8941677516143532
[1.01030928] 0.7484536082475426 0.8963354311493872
1.0
```

4 НЕЛІНІЙНА РЕГРЕСІЯ

В прикладі nlinreg.py показано побудову нелінійної моделі (**поліном** другого степеня). Щоб зрозуміти, як це працює, виведіть значення масиву поліноміальних ознак `x_poly`. Він містить три ознаки: x^0, x^1, x^2. Ці ознаки використовуються для навчання лінійної моделі `LinearRegression`. Функція `fit()` шукає лінійну залежність у вигляді $ax^0+bx^1+cx^2+d$ методом найменших квадратів. Програма виводить значення коефіцієнтів (a, b, c, d), коефіцієнта детермінації R^2 та будує графік з даними та моделлю.

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

x = np.array([8,0,3,4,9,7,1,6,3,9])
y = np.array([9,0,2,6,9,8,2,9,4,9])
x=x[:, None]

# поліном степні 2 з вільним членом
poly = PolynomialFeatures(degree=2)
x_poly = poly.fit_transform(x)
poly.get_feature_names()

model = LinearRegression()
```

```
model.fit(x_poly, y)
print model.coef_, model.intercept_, model.score(x_poly,y)

X = np.linspace(0, 10, 1000)
X_poly = poly.transform(X[:, None])
Y = model.predict(X_poly)

plt.scatter(x, y)
plt.plot(X, Y)
plt.ylabel('y'),plt.xlabel('x')
plt.show()
```

Результати:

```
[ 0. 1.70728993 -0.07178631] -0.2526432943795145
0.9233988148307714
```

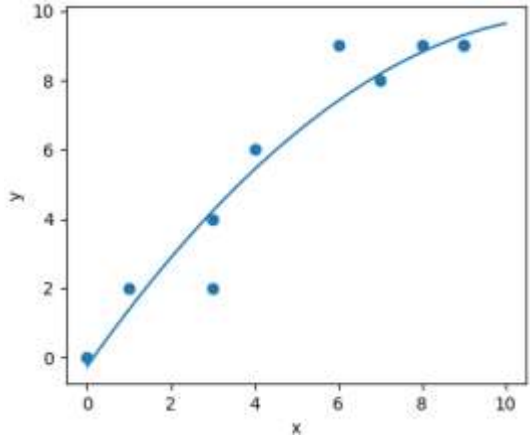


Рисунок 3 – Нелінійна регресія

5 ДВОВИМІРНА НЕЛІНІЙНА РЕГРЕСІЯ

В прикладі nlinreg2D.py показано побудову двовимірної нелінійної моделі (**поліном** другого степеня). Тут масив поліноміальних ознак `x_poly` містить ознаки: 1, x_0, x_1, x_0^2, x_0x_1, x_1^2. Ці ознаки використовуються для навчання лінійної моделі

`LinearRegression`. Функція `fit()` шукає лінійну залежність у вигляді $a \cdot 1 + bx_0 + cx_1 + dx_0^2 + ex_0x_1 + fx_1^2 + g$ методом найменших квадратів. Програма виводить значення коефіцієнтів (*a, b, c, d, e, f, g*), коефіцієнта детермінації R^2 та будує графік з даними та моделлю.

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

x = np.array([[0,1,2,0,1,2,0,1,2],
              [0,0,0,1,1,1,2,2,2]])
y = np.array([0,1,2,1,2,3,2,3,9])
x=x.T

# поліном степні 2 з вільним членом
poly = PolynomialFeatures(degree=2)
x_poly = poly.fit_transform(x)
poly.get_feature_names() # назви поліноміальних ознак

model = LinearRegression()
model.fit(x_poly, y)
print model.coef_, model.intercept_, model.score(x_poly,y)

X=np.mgrid[0:3:0.5,0:3:0.5]
X_=X.reshape((2,X.size//2))
X_poly = poly.transform(X_.T)
Y = model.predict(X_poly)

from mpl_toolkits.mplot3d import axes3d
fig = plt.figure() # створити фігуру
ax = fig.add_subplot(111, projection='3d') # додати графік
3D
```

```
Y=Y.reshape(X.shape[1:])
ax.scatter(x[:,0], x[:,1], y) # показати емпіричні точки
ax.plot_wireframe(X[0], X[1], Y, rstride=1, cstride=1) #
показати теоретичну поверхню
ax.set_xlabel('X0'),ax.set_ylabel('X1'),ax.set_zlabel('Y')
plt.show()
```

Результати:

```
[ 0. -1.08333333 -1.08333333 0.83333333 1.25 0.83333333]
0.6944444444444431 0.9103483606557377
```

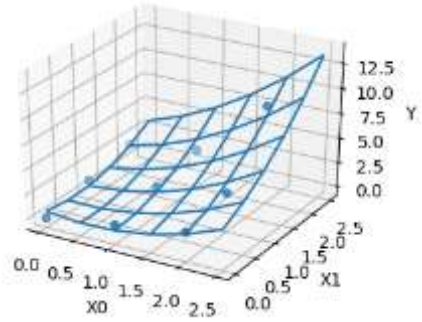


Рисунок 4а – Двовимірна нелінійна регресія

6 РЕГРЕСІЙНІ МОДЕЛІ

В прикладі regress.py на одній вибірці навчаються різні регресійні моделі `LinearRegression` (лінійна), `KNeighborsRegressor` (k-сусідів), `SVR` (опорних векторів), `DecisionTreeRegressor` (дерева рішень), `GradientBoostingRegressor` (градієнтного підсилення), `MLPRegressor` (багатошарового перцептрона). Для кожної моделі програма виводить прогноз для x=5 та коефіцієнт детермінації R^2.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])
# на практиці застосовуйте train_test_split

M=[] # моделі
M+=[LinearRegression()]
M+=[KNeighborsRegressor()]
M+=[SVR()]
M+=[DecisionTreeRegressor()]
M+=[GradientBoostingRegressor()]
M+=[MLPRegressor()]

for model in M:
    model.fit(x, y)
    print model.predict([[5]]), model.score(x,y)
```

Результати:

```
[5.8] 0.8964307048984468
[5.8] 0.8068100358422938
[7.15985159] 0.5402822781027095
[6.] 0.982078853046595
[5.99998327] 0.9820788506534793
[5.84756539] 0.9022676253692934
```

7 РЕГРЕСІЙНА МОДЕЛЬ «ДЕРЕВО РІШЕНЬ»

Приклад decisionTreeRegress.py будує одновимірну регресійну модель на основі **дерева рішень** з максимальною глибиною 2. Програма виводить прогноз для x=5, коефіцієнт детермінації R^2,

важливість ознак (доцільно для багатьох ознак), будує дерево та показує шлях рішення для *x*=5. Для візуалізації дерева (рис. 4б) необхідно установити програму Graphviz, або скористатись відповідним онлайн-сервісом.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.tree import DecisionTreeRegressor,
export_graphviz

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])

model= DecisionTreeRegressor(max_depth=2)
model.fit(x, y)
print model.predict([[5]]), model.score(x,y)
print model.feature_importances_

from StringIO import StringIO
f = StringIO()
export_graphviz(model, out_file=f)
print f.getvalue()
path=model.decision_path([[5]]).toarray()
print path
```

Результати:

```
[6.] 0.9689366786140979
[1.]
array([[1, 0, 0, 0, 1, 1, 0]])
```

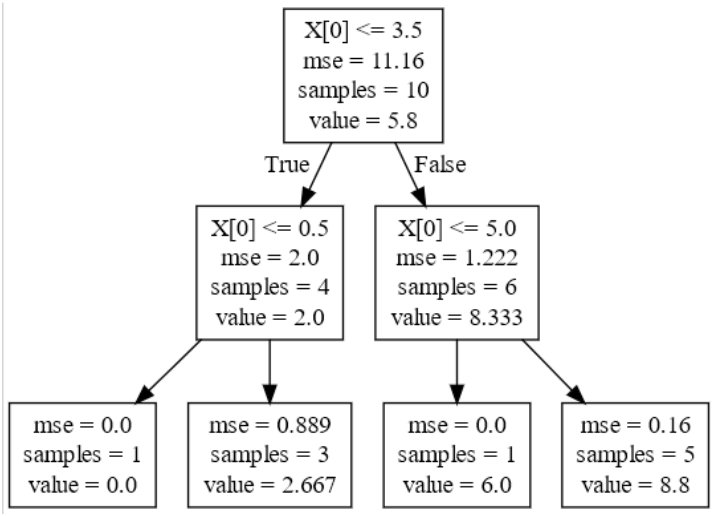


Рисунок 4б – Дерево рішень

8 РЕГРЕСІЙНА МОДЕЛЬ «БАГАТОШАРОВИЙ ПЕРЦЕПТРОН»

В прикладі MLPRegress.py побудовано регресійні моделі на основі нейронних мереж (**багатошарових перцептронів**) з одним скритим шаром розміром 1 (рис. 5а) і 2 (рис. 5б). Програма виводить вільні члени та коефіцієнти моделей.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.neural_network import MLPRegressor

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])

# 1 скритий шар розміром 1
model= MLPRegressor(solver='lbfgs', alpha=0,
hidden_layer_sizes=[1], activation='tanh')
```

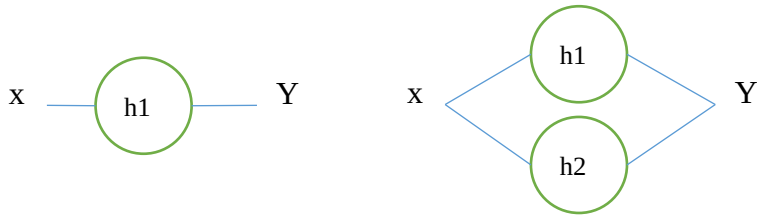
```
model.fit(x, y)
print model.intercepts_# вільні члени
print model.coefs_# коефіцієнти
Y=model.predict(x)

# модель:
h1=np.tanh(-2.73092729+0.75195748*x)
Yf=4.85558639+4.00237182*h1

# 1 скритий шар розміром 2
model= MLPRegressor(solver='lbfgs', alpha=0,
hidden_layer_sizes=[2], activation='tanh')
model.fit(x, y)
print model.intercepts_
print model.coefs_
Y=model.predict(x)

# модель:
h1=np.tanh(13.87883208-3.47790122*x)
h2=np.tanh(-0.45782002+1.03259976*x)
Yf=-2.89208082*h1+2.111439*h2+3.79645258
```

Результати:



а б

Рисунок 5 – Багатошаровий перцептрон (одна ознака) з одним скритим шаром розміром 1 (а) і 2 (б)

9 ДВОВИМІРНА РЕГРЕСІЙНА МОДЕЛЬ «БАГАТОШАРОВИЙ ПЕРЦЕПТРОН»

В прикладі MLPRegress2D.py побудовано двовимірну регресійну модель на основі **багатошарового перцептрону** з одним скритим шаром розміром 1 (рис. 6а). Щоб побудувати модель з одним скритим шаром розміром 2 (рис. 6б) змініть `hidden_layer_sizes=[2]`. Програма виводить вільні члени та коефіцієнти моделей.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.neural_network import MLPRegressor

x = np.array([[0,1,2,0,1,2,0,1,2],
              [0,0,0,1,1,1,2,2,2]])
y = np.array([0,1,2,1,2,3,2,3,9])
x=x.T

# 1 скритий шар розміром 1
model= MLPRegressor(solver='lbfgs', alpha=0,
hidden_layer_sizes=[1], activation='tanh')
model.fit(x, y)
print model.intercepts_ # вільні члени
print model.coefs_ # коефіцієнти
Y=model.predict(x)

# модель:
h1=np.tanh(4.6427557-0.54709987*x[:,0]-0.54759059*x[:,1])
Yf=560.91609523-560.27879759*h1
```

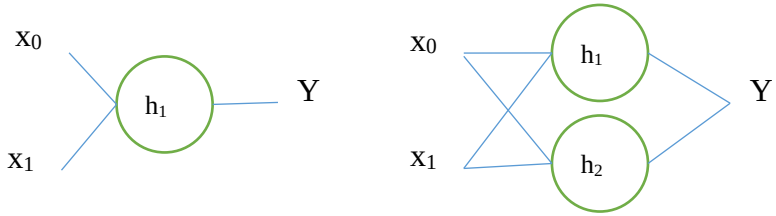


Рисунок 6 – Багатошаровий перцептрон (дві ознаки) з одним скритим шаром розміром 1 (а) і 2 (б)

10 ПЕРЕВІРКА МОДЕЛІ НА ТЕСТОВИХ ДАНИХ

В прикладі traintestsplit.py лінійна регресійна модель перевіряється на тестових даних. Усі дані були поділені на **дані для навчання** (75%) та **тестові дані** (25%). Дані для навчання використовувались для створення моделі, а тестові дані – тільки для її перевірки. Перевірка на тестових даних показує, що реальна якість моделі не є високою. Програма виводить оцінку моделі на даних для навчання, оцінку моделі на тестових даних та оцінку моделі на усіх даних. У більшості практичних випадків перевірку моделі потрібно виконувати тільки на тестових даних.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.25)

model = LinearRegression()
```

```
model.fit(x_train, y_train)
print model.coef_, model.intercept_
print model.score(x_train, y_train)

y_pred = model.predict(x_test)
print model.score(x_test, y_test) # точність моделі на
тестових даних
print r2_score(y_test, y_pred) # або
print model.score(x, y)
#plt.scatter(expected, predicted)
```

Результати:

```
[1.] 0.857142857142855
0.9083969465648857
0.552295918367347
0.552295918367347
0.8957647575159096
```

11 ПЕРЕХРЕСНА ПЕРЕВІРКА

Однократна перевірка моделі на тестових даних не є надійною. В прикладі crossValidation.py показано способи багатократної перевірки – перехресної перевірки. Для **тьохблокової перехресної перевірки** функція `cross_val_score` ділить усі дані на три частини: 1, 2, 3. Перший раз модель будується за даними 1, 2 і перевіряється за даними 3. Другий раз модель будується за даними 1, 3 і перевіряється за даними 2. Третій раз модель будується за даними 2, 3 і перевіряється за даними 1. Таким чином, отримано три оцінки і можна підрахувати середню. **Перехресна перевірка з випадковими перестановками** теж перевіряє модель три рази, але щоразу дані діляться випадково навпіл на навчаючі та тестові. Програма виводить оцінки цих перехресних перевірок та індекси навчаючих і тестових даних для перехресної перевірки з випадковими перестановками.

```
# -*- coding: utf-8 -*-
```

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])
model = LinearRegression()

# перехресна перевірка - це удосконалення train_test_split
+ score
s=cross_val_score(model, x, y, cv=3)
print s, s.mean()

# перехресна перевірка з випадковими перестановками
from sklearn.model_selection import ShuffleSplit
cv = ShuffleSplit(test_size=0.5, n_splits=3) # спробуйте
test_size=3
s=cross_val_score(model, x, y, cv=cv)
print s, s.mean()
for train_index, test_index in cv.split(x):
    print train_index, test_index # вивести індекси даних
для CV
```

Результати:

```
[0.81862845 0.93487684 0.61472012] 0.7894084682862825
[0.74747434 0.82090904 0.89331962] 0.8205676642838404
[0 2 6 8 9] [5 7 4 3 1]
[7 0 9 8 1] [3 6 4 5 2]
[3 6 1 4 9] [7 0 2 8 5]
```

12 КРИВІ ПЕРЕВІРКИ ТА НАВЧАННЯ

В прикладі validationCurve.py згенеровано дані за залежністю $0{,}2x^2+1$, до яких додано випадковий шум (рис. 7). Побудовано **криві перевірки** (рис. 8) для поліноміальних моделей (степінь

полінома 0-7) – залежності якості моделі від гіперпараметра моделі (степеня полінома). Якість обчислено за навчаючими даними (суцільна лінія) і тестовими (штрихова лінія). За штриховою лінією видно, що найвищої якості модель досягає в точці `degree`=2. **Криві навчання** для моделі `degree`=2 показано на рис. 9. Це залежності якості моделі від кількості даних для навчання (`train_size`). Якість також обчислено за навчаючими даними (суцільна лінія) і тестовими (штрихова лінія). За штриховою лінією видно, що якість моделі стає достатньо високою, коли `train_size` ≥ 8.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import validation_curve
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt

# випадкові дані
# x з 0 до 10
x = 10*np.random.random((30, 1))
# y = a*x**2 + b з шумом
y = 0.2*x**2+1 + 2*np.random.normal(size=x.shape)
plt.scatter(x, y)
plt.xlabel('x'), plt.ylabel('y')
plt.show(); plt.figure()

# будуємо криві перевірки
model = make_pipeline(PolynomialFeatures(),
LinearRegression())
degree = np.arange(0, 8)
train_scores, test_scores = validation_curve(model, x, y,
'polynomialfeatures__degree', degree, cv=3)
```

```
plt.plot(degree, np.mean(train_scores, 1), 'o-') # оцінка
навчання
plt.plot(degree, np.mean(test_scores, 1), 'o--') # оцінка
перевірки
plt.xlabel('degree'),plt.ylabel('score')
plt.show(); plt.figure()

# будуємо криві навчання для моделі degree=2
model = make_pipeline(PolynomialFeatures(degree=2),
LinearRegression())
train_sizes, train_scores, test_scores =
learning_curve(model, x, y, cv=3,
train_sizes=np.linspace(0.2, 1, 20))
plt.plot(train_sizes, np.mean(train_scores, 1), 'o-')#
оцінка навчання
plt.plot(train_sizes, np.mean(test_scores, 1), 'o--')#
оцінка перевірки
plt.xlabel('train_size'),plt.ylabel('score')
plt.show()
```

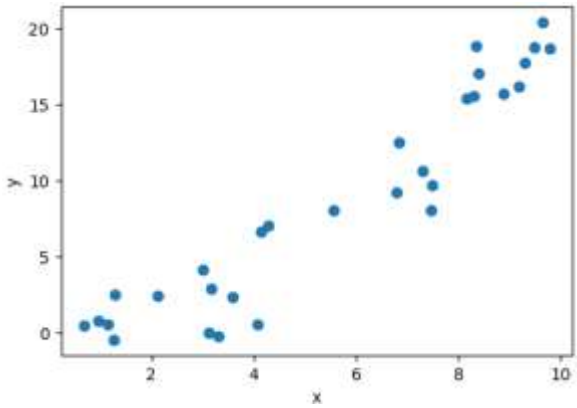


Рисунок 7 – Дані

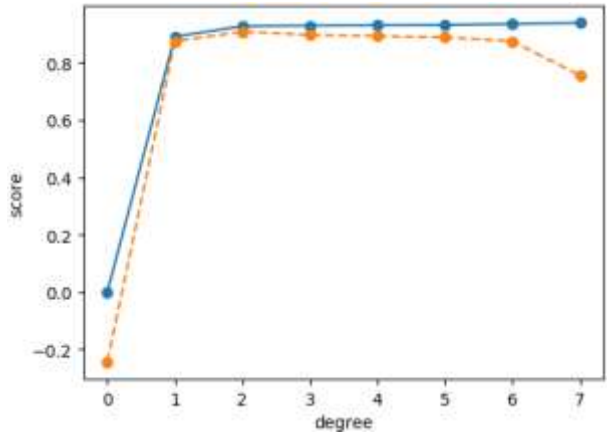


Рисунок 8 – Криві перевірки

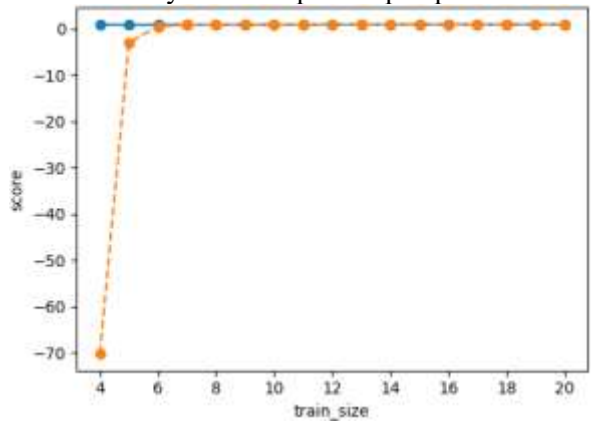


Рисунок 9 - Криві навчання для моделі `degree`=2

13 ЛІНІЙНА КЛАСИФІКАЦІЯ

В прикладі linClassify.py побудовано двовимірну лінійну модель **бінарної класифікації** `LogisticRegression`. На відміну від регресії, залежна змінна класифікації приймає тільки дискретні значення. Для бінарної класифікації залежна змінна *y* може бути рівна тільки нулю або одиниці. Перед побудовою моделі програма з використанням пакету Pandas будує матрицю діаграм розсіювання з

гістограмами для кожної ознаки вибірки для навчання (рис. 10). Після побудови моделі програми візуалізує класи вибірки для навчання і границю прийняття рішень (рис. 11). Темні точки відповідають значенню *y*=0, а світлі значенню *y*=1. Лінія границі прийняття рішень показує, що якщо нова точка буде під цією лінією ліворуч, то модель класифікує її як 0, а якщо вона буде над цією лінією праворуч, то модель класифікує її як 1. Також програма виводить прогноз в точці (5, 10).

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# дві ознаки класів
x=np.array([[0,1,1,2,2,3,2,3,1,3, 6,5,6,7,7,8,7,7,8,5],
            [1,1,3,1,2,2,3,4,4,1, 5,7,6,7,6,7,5,8,8,8]])
# мітки класів (бінарна класифікація)
y=np.array( [0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,1,1,1,1,1] )
x=x.T

# рисуємо матрицю діаграм розсіювання
import pandas as pd
df=pd.DataFrame(x, columns=[0,1])
pd.plotting.scatter_matrix(df, c=y, figsize=(5, 5),
hist_kwds={'bins': 5})
plt.show(); plt.figure()

model=LogisticRegression(C=100) # лінійний класифікатор
model.fit(x,y)
b=model.intercept_ # вільний член
a=model.coef_ # коефіцієнти

# способи прогнозу в точці p
p=np.array([[5,10]])
```

```
print model.predict(p)
print a[0,0]*p[0,0]+a[0,1]*p[0,1]+b > 0
print model.decision_function(p) > 0

plt.scatter(x[:,0], x[:,1], c=y) # візуалізація класів

# рисуємо границю прийняття рішень
x1, x2 = np.meshgrid(np.linspace(0, 10), np.linspace(0,
10))
xx = np.c_[x1.ravel(), x2.ravel()]
d,l = model.decision_function(xx), [0]
#d,l = model.predict_proba(xx)[:, 1], [0.5] # або
plt.contour(x1, x2, d.reshape(x1.shape) ,levels=l,
colors="black")

plt.xlabel('x0'), plt.ylabel('x1')
plt.show()
```

Результати:

```
[1]
[ True]
[ True]
```

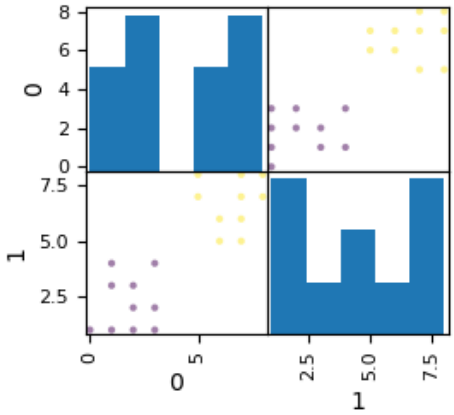


Рисунок 10 - Матриця діаграм розсіювання

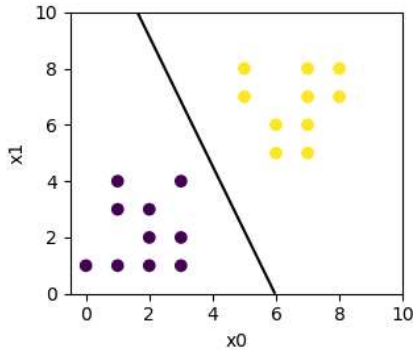


Рисунок 11 – Візуалізація класів та границі прийняття рішень

14 МОДЕЛІ КЛАСИФІКАЦІЇ

Приклад classify.py подібний на приклад regress.py, але будуються різні моделі класифікації: `KNeighborsClassifier` (метод k-сусідів), `LogisticRegression` (логістична регресія), `LinearSVC` (лінійний метод опорних векторів), `GaussianNB` (наївний баєсів класифікатор), `DecisionTreeClassifier` (дерево рішень), `RandomForestClassifier` (випадковий ліс), `SVC` (ядерний метод опорних векторів) та `MLPClassifier` (багатошаровий перцептрон). В цьому прикладі усі дані (рис. 12) випадково поділяються навпіл на дані для навчання та тестові дані. Програма для кожної моделі виводить прогноз на тестових даних та оцінку моделі на тестових даних. Помітно, що найкращою моделлю є `LogisticRegression`, але і у неї є одна помилка (10%), тому її якість 90% (0,9).

```
# -*- coding: utf-8 -*-
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

# дві ознаки класів
x=np.array([[0,1,1,2,2,3,2,3,1,3, 6,5,6,7,7,8,7,7,8,5],
            [1,1,3,1,2,2,3,4,4,8, 5,7,6,7,6,7,5,8,8,1]])
# мітки класів (бінарна класифікація)
y=np.array( [0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,1,1,1,1,1] )
x=x.T

plt.scatter(x[:,0], x[:,1], c=y) # візуалізація класів
plt.xlabel('x0'), plt.ylabel('x1')
plt.show()

x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.5,random_state=11)
print y_test # фактичні тестові класи

M=[] # моделі
M+=[KNeighborsClassifier(n_neighbors=3,
weights='distance')]
# метод k сусідів
# n_neighbors - к-ть сусідів
# weights - функція ваг
M+=[LogisticRegression(C=100, penalty="l1")]
# логістична регресія
# C - параметр регуляризації (менше C - більша
регуляризація). За замовчуванням 1
# penalty - тип регуляризації
M+=[LinearSVC(C=100)]
# лінійний метод опорних векторів
```

```
M+=[GaussianNB()]
# наївний баєсів класифікатор
# див. також MultinomialNB і BernoulliNB
M+=[DecisionTreeClassifier(max_depth=4)]
# дерево рішень
# max_depth - максимальна глибина дерева
M+=[RandomForestClassifier(n_estimators=5)]
# випадковий ліс
# n_estimators - кількість дерев
# див. також GradientBoostingClassifier
M+=[SVC(kernel='rbf', C=10, gamma=0.1)]
# ядерний метод опорних векторів
M+=[MLPClassifier(solver='lbfgs', hidden_layer_sizes=[2],
activation='tanh',alpha=0.1)]
# багатошаровий перцептрон
# hidden_layer_sizes=[2] - кількість елементів в скритому
шарі
# activation - функція активації
# alpha - регуляризація
# застосовуйте StandardScaler

for model in M:
    model.fit(x_train, y_train) # виконати навчання
    print model.predict(x_test), model.score(x_test,
y_test) # спрогнозовані класи
```

Результати

```
[0 0 1 0 1 1 1 0 1 0]
[0 0 1 0 1 0 1 1 1 0] 0.8
[0 0 1 0 1 0 1 0 1 0] 0.9
[0 0 1 0 1 0 1 1 1 0] 0.8
[0 0 1 0 1 0 1 1 1 0] 0.8
[0 0 1 0 1 0 1 1 1 0] 0.8
[0 0 1 0 1 0 1 1 1 0] 0.8
[0 0 1 0 1 0 1 1 1 0] 0.8
```

```
[0 0 1 0 1 0 1 1 1 0] 0.8
```

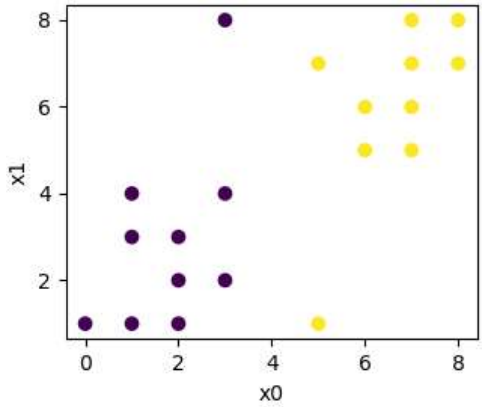


Рисунок 12 – Візуалізація усіх класів

15 МЕТРИКИ КЛАСИФІКАЦІЇ

В прикладі classifyMetrics.py показані різні метрики бінарної класифікації. Нехай клас 0 – Negative (*N*), клас 1 – Positive (*P*). **Матриця помилок** для бінарної класифікації (таблиця 2) показує кількість правильних і помилкових прогнозів за результатами перевірки моделі на тестових даних: *TN* (True Negative) – фактично *N* і спрогнозовано як *N*; *TP* (True Positive) – фактично *P* і прогнозовано як *P*; *FN* (False Negative) – фактично *P*, а прогнозовано як *N*; *FP* (False Positive) – фактично *N*, а прогнозовано як *P*. В даному випадку серед 10 тестових даних є 8 правильних прогнозів (виділено жирним) і 2 помилки.

Таблиця 2 - Матриця помилок для бінарної класифікації

		Прогноз	
		Клас *N*	Клас *P*
Факт.	Клас *N*	***TN*=4**	*FP*=1
	Клас *P*	*FN*=1	***TP*=4**

В залежності від цілей класифікації можуть використовуватись такі метрики:

Правильність (accuracy) – це частка правильно класифікованих $TP+TN$

$$s=(TP+TN)/(TP+TN+FP+FN).$$

Точність (precision) – це частка TP серед усіх прогнозованих P

$$p=TP/(TP+FP).$$

Повнота (recall) – це частка TP серед усіх фактичних P

$$r=TP/(TP+FN).$$

F1-міра (f1-score) – це гармонічне середнє точності і повноти

$$f1=2pr/(p+r).$$

Середня точність класифікатора (average precision score) - це площа під кривою точність-повнота (рис. 13), яка будується для різних порогових значень імовірності приналежності до класу [8].

```
# -*- coding: utf-8 -*-
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# дві ознаки класів
x=np.array([[0,1,1,2,2,3,2,3,1,3, 6,5,6,7,7,8,7,7,8,5],
            [1,1,3,1,2,2,3,4,4,8, 5,7,6,7,6,7,5,8,8,1]])
# мітки класів (бінарна класифікація)
y=np.array( [0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,1,1,1,1,1] )
x=x.T

x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.5,random_state=11)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train) # виконати навчання
Y_test=knn.predict(x_test) # прогнозовані класи
print y_test # фактичні тестові класи
```

```
print Y_test # прогнозовані тестові класи

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, Y_test) # матриця помилок
# нехай клас 0 - Negative (N), клас 1 - Positive (P)
TN=cm[0,0] # фактично 0, прогнозовано 0 (True Negative)
TP=cm[1,1] # фактично 1, прогнозовано 1 (True Positive)
FN=cm[1,0] # фактично 1, прогнозовано 0 (False Negative)
FP=cm[0,1] # фактично 0, прогнозовано 1 (False Positive)

s=(TP+TN)/cm.sum() # правильність - частка правильно
класифікованих
# або
s=knn.score(x_test, y_test) # правильність
p=TP/(TP+FP) # точність - частка TP серед усіх
прогнозованих P
r=TP/(TP+FN) # повнота - частка TP серед усіх фактичних P
f1=2*p*r/(p+r) # F1-міра - гармонічне середнє точності і
повноти
# або
from sklearn.metrics import f1_score
f1=f1_score(y_test, Y_test) # F1-міра

from sklearn.metrics import classification_report
print classification_report(y_test, Y_test) # повний звіт
по класифікації

knn.predict_proba([[4,4]]) # імовірність класу для 1 точки
y_scores=knn.predict_proba(x_test)[:,1] # імовірності
класу 1 тестових даних
Y1=y_scores>0.5 # порогове значення імовірності за
замовчуванням, порівняйте з Y_test
Y2=y_scores>0.7 # тепер інша к-ть точок буде належати
класу 1, порівняйте з Y1
```

```
print classification_report(y_test, Y2) # порівняйте з
попереднім звітом

# крива точності-повноти
# будується для різних порогових значень імовірності
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds =
precision_recall_curve(y_test, y_scores)
plt.plot(precision, recall)
plt.xlabel(u"Точність"), plt.ylabel(u"Повнота")
plt.show()

# середня точність класифікатора (площа під кривою
точності-повноти)
from sklearn.metrics import average_precision_score
print average_precision_score(y_test, y_scores)
# див. також ROC-криві і AUC [Мюллер с.315]
```

Результати:

```
[0 0 1 0 1 1 1 0 1 0]
[0 0 1 0 1 0 1 1 1 0]
             precision    recall  f1-score   support

          0       0.80      0.80      0.80         5
          1       0.80      0.80      0.80         5

avg / total       0.80      0.80      0.80        10

             precision    recall  f1-score   support

          0       0.83      1.00      0.91         5
          1       1.00      0.80      0.89         5

avg / total       0.92      0.90      0.90        10
```

```
0.9
```

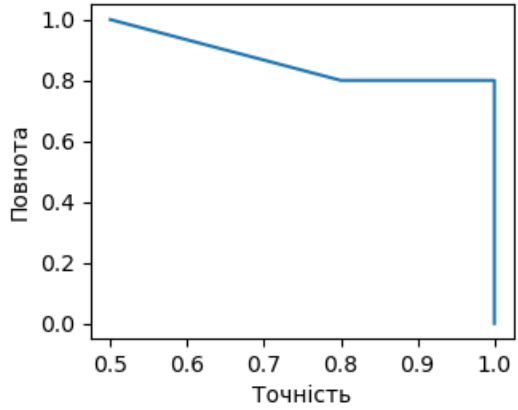


Рисунок 13 - Крива точність-повнота

16 КОДУВАННЯ ОЗНАК

В прикладі categorial.py показано кодування категоріальних ознак в неперервні (використовується пакет Pandas) та кодування неперервних ознак в категоріальні (біннінг).

```
# -*- coding: utf-8 -*-
# кодування категоріальних ознак в неперервні
import numpy as np
import pandas as pd
x1 = [0, 2, 2, 3, 9] # неперервні ознаки
x2 = ['Male', 'Female', 'Male', 'Male', 'Male'] #
категоріальні ознаки
dataSet = zip(x1,x2) # підготувати дані
df = pd.DataFrame(data = dataSet, columns=['X1', 'X2']) #
об'єкт DataFrame
dfc = pd.get_dummies(df) # кодувати категоріальні ознаки
print dfc
X=dfc.values # масив numpy
```

```
# кодування неперервних ознак в категоріальні (біннінг)
bins = np.linspace(0, 10, 6)
x1c = np.digitize(x1, bins=bins) # повертає індекси бінів
print x1c
```

Результат:

```
   X1  X2_Female  X2_Male
0   0          0        1
1   2          1        0
2   2          0        1
3   3          0        1
4   9          0        1

array([1, 2, 2, 2, 5])
```

17 ВІДБІР ОЗНАК

Часто ми не знаємо, які ознаки впливають на значення залежної змінної, а які ні. В прикладі featureSelect.py показано різні способи відбору ознак. Одновимірний відбір ознак (**дисперсійний аналіз**) полягає в тому, що за F-значенням вибираємо певний відсоток найбільш значущих ознак. **Відбір ознак на основі моделі** оснований на можливості деяких моделей безпосередньо вказувати на важливість ознак. Наприклад величина коефіцієнтів лінійних моделей вказує на важливість відповідних ознак (див. приклад 2). **Ітеративний відбір** ознак оснований на тому, що будується послідовність моделей з різною кількістю ознак і визначаються неважливі ознаки. Наприклад функція `RFE` реалізує метод рекурсивного виключення ознак. Рекомендується також застосовувати **експертні знання** для додання нових ознак.

```
# -*- coding: utf-8 -*-
# відбір ознак
import numpy as np
```

```
from sklearn.feature_selection import SelectPercentile,
f_regression
#from sklearn.model_selection import train_test_split

x1 = np.linspace(0,10,110) # закономірна ознака
x2 = 5*np.random.normal(size=x1.shape) # шумова ознака
X = np.vstack([x1, x2]).T # усі ознаки
y = 1+2*x1+1*np.random.normal(size=x1.shape)
# на практиці застосовуйте train_test_split

# одновимірний відбір ознак (дисперсійний аналіз)
# за F-значенням вибираємо 50% найбільш значущих ознак
# f_classif - для класифікації
# f_regression - для регресії
select = SelectPercentile(score_func=f_regression,
percentile=50)
select.fit(X, y)
print select.scores_ # оцінки ознак
print select.pvalues_ # p-значення оцінок (високі
відкидаємо)
print select.get_support() # які ознаки відібрані
# отримуємо новий набір даних без шумових ознак
X_selected = select.transform(X)

# відбір ознак на основі моделі
from sklearn.ensemble import GradientBoostingRegressor
model=GradientBoostingRegressor()
model.fit(X, y)
print model.feature_importances_ # оцінки ознак
# для відбору ознак можна також використовувати
коефіцієнти лінійних моделей і моделі Lasso

# або
from sklearn.feature_selection import SelectFromModel
```

```
select = SelectFromModel(model) # відбір ознак на основі
моделі
select.fit(X, y)
print select.get_support() # які ознаки відібрані

# ітеративний відбір ознак -
# будується послідовність моделей з різною кількістю ознак
from sklearn.feature_selection import RFE
select = RFE(model) # метод рекурсивного виключення ознак
select.fit(X, y)
print select.get_support() # які ознаки відібрані
```

Результати:

```
[3.61696832e+03 1.88058541e+00]
[7.15743323e-85 1.73110222e-01]
[ True False]
[0.5533245 0.4466755]
[ True False]
[ True False]
```

18 ЗМЕНШЕННЯ РОЗМІРНОСТІ

В прикладі dimReduction.py показано різні методи **зменшення розмірності**, які відносять до методів машинного навчання без учителя: PCA, NMF, t-SNE. Аналіз головних компонентів (PCA) оснований на пошуку в даних напрямків максимальної дисперсії – головних компонентів (рис. 14, 15). PCA можна застосувати для видалення шуму з даних. Наприклад, якщо є дві ознаки, одна з яких шумова, то PCA дозволяє залишити один головний компонент (рис. 16). Ще одним методом є факторизація невід'ємних матриць (NMF). В прикладі за допомогою NMF відбувається зменшення розмірності з трьох ознак до двох (рис. 17) шляхом видалення зайвої ознаки. Ще один метод t-SNE намагається знайти двовимірне представлення даних, яке зберігає відстані між точками найкращим чином. Його використовують для двовимірного представлення багатовимірних даних (рис. 18).

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
plt.axis('equal')

from sklearn.decomposition import PCA

# дані з шумом
x1 = np.linspace(10,20,1000)
x2=10+2*x1+1*np.random.normal(size=x1.size)
x = np.vstack([x1, x2]).T # усі ознаки

# аналіз головних компонентів
# шукає напрямки максимальної дисперсії - головні
компоненти
model = PCA(n_components=2) # залишити 2 головних
компонента
model.fit(x) # підгонка моделі
X = model.transform(x) # перетворити дані
C=model.components_ # напрямки максимальної дисперсії
компонентів
a0,b0=C[0] # напрямок (в системі координат x1,x2)
максимальної дисперсії головного компонента
a1,b1=C[1] # напрямок (в системі координат x1,x2)
максимальної дисперсії другого компонента
V=model.explained_variance_ # відповідні дисперсії
S = V**0.5 # стандартні відхилення
m0,m1=model.mean_ # емпіричні середні

plt.scatter(x1, x2, c=x1) # початкові дані
# напрямки максимальної дисперсії головних компонентів
plt.arrow(m0, m1, S[0]*a0, S[0]*b0, width=.1,
head_width=.5, color='k')
```

```
plt.arrow(m0, m1, S[1]*a1, S[1]*b1, width=.1,
head_width=.5, color='k')
plt.xlabel('x1'),plt.ylabel('x2')
plt.show(); plt.figure()

plt.scatter(X[:,0], X[:,1], c=x1) # перетворені дані
plt.xlabel('X1'),plt.ylabel('X2')
plt.show(); plt.figure()

# аналіз головних компонентів для видалення шуму
model = PCA(n_components=1) # залишити 1 головний
компонент
model.fit(x)
X = model.transform(x)
xi=model.inverse_transform(X) # зворотна трансформація в
початковий простір ознак x1, x2 (відміна обертання)
plt.scatter(xi[:,0], xi[:,1], c=x1) # дані без шуму
plt.xlabel('x1'),plt.ylabel('x2')
plt.show(); plt.figure()

# факторизація невід’ємних матриць
# дані з шумом
x1 = np.linspace(10,20,1000)
x2=10+2*x1+1*np.random.normal(size=x1.size)
x3=x1+x2+1*np.random.normal(size=x1.size) # сума x1+x2
# x повинні бути невід’ємні !
x = np.vstack([x1, x2, x3]).T # усі ознаки
from sklearn.decomposition import NMF
model = NMF(n_components=2) #
model.fit(x) # підгонка моделі (x – невід’ємні!)
# дозволяє виділити доданки x3
X = model.transform(x)
X = model.inverse_transform(X)
plt.scatter(X[:,0], X[:,1], c=x1)
plt.xlabel('X1'),plt.ylabel('X2')
```

```
plt.show(); plt.figure()

# t-SNE намагається знайти двовимірне представлення даних,
яке зберігає відстані між точками найкращим чином
# використовують для двовимірного представлення даних
from sklearn.manifold import TSNE
model = TSNE()
X = model.fit_transform(x)
plt.scatter(X[:,0], X[:,1], c=x1) # перетворені дані
plt.xlabel('X1'),plt.ylabel('X2')
plt.show()
```

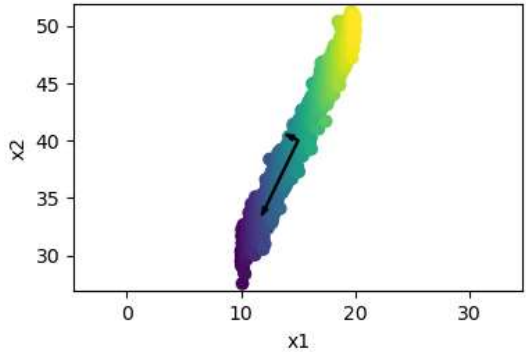


Рисунок 14 - Початкові дані з напрямками максимальної дисперсії головних компонентів

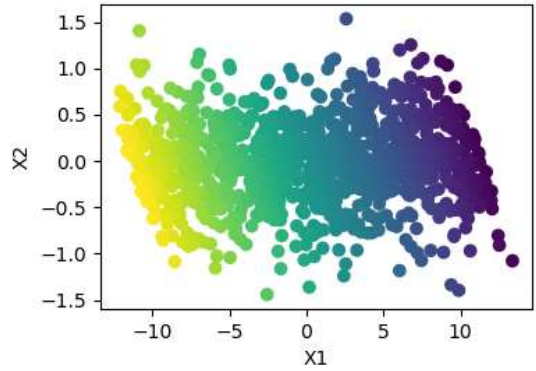


Рисунок 15 - Перетворені дані

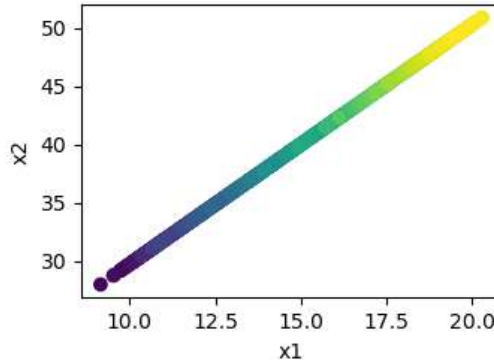


Рисунок 16 - Дані без шуму (один головний компонент)

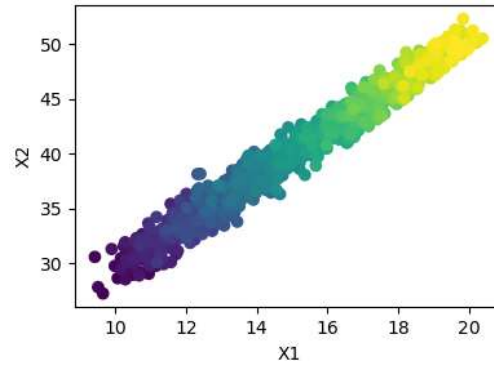


Рисунок 17 - Факторизація невід’ємних матриць

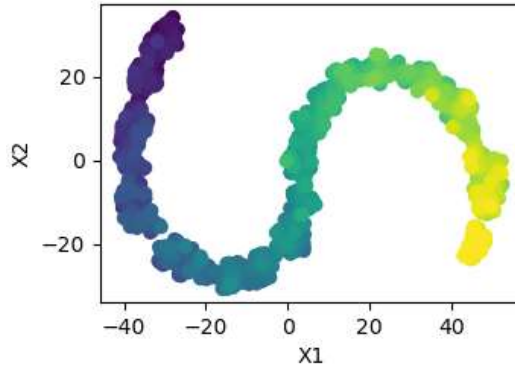


Рисунок 18 – Метод t-SNE

19 КЛАСТЕРИЗАЦІЯ

Кластеризація - це метод машинного навчання без учителя, оснований на пошуку груп (кластерів) схожих об'єктів. В прикладі clastering.py показано такі методи кластеризації як метод k-середніх, метод агломеративної кластеризації та метод DBSCAN. Алгоритм методу k-середніх обчислює центри ваги кластерів. Агломеративна кластеризація об'єднує подібні кластери в агломерації. DBSCAN - це оснований на щільності алгоритм кластеризації просторових даних з наявністю шуму, який шукає ядрові точки в щільних зонах. Дані для кластеризації показано на рис. 19. Програма виводить мітки кластерів, які обчислені цими методами. Зверніть увагу, що усі ці методи роблять помилку в точках 10 і 20. Існують також інші методи кластеризації, наприклад ієрархічної [8].

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
plt.axis('equal')
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN

# дані
# дві ознаки класів
x=np.array([[0,1,1,2,2,3,2,3,1,3, 6,5,6,7,7,8,7,7,8,5],
            [1,1,3,1,2,2,3,4,4,8, 5,7,6,7,6,7,5,8,8,1]])
# мітки класів
y=np.array( [0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,1,1,1,1,1] )
x=x.T

plt.scatter(x[:,0], x[:,1], c=y) # візуалізація класів
plt.xlabel('x0'), plt.ylabel('x1')
plt.show()
```

```
m=KMeans(n_clusters=2) # метод k-середніх
# алгоритм обчислює центри ваги кластерів
m.fit(x)
print m.labels_
print m.predict([[1,2]]) # прогноз в новій точці
print m.cluster_centers_ # центри кластерів

m=AgglomerativeClustering(n_clusters=2, linkage='ward') #
агломеративна кластеризація
# об’єднує подібні кластери в агломерації
# linkage - критерій порівняння кластерів
# агломеративні методи не мають методу predict
m.fit(x)
print m.labels_
# див. також ієрархічну класифікацію [Мюллер, с. 201]
#from scipy.cluster.hierarchy import dendrogram, ward

m=DBSCAN(eps=2.0, min_samples=5) # оснований на щільності
алгоритм кластеризації просторових даних з наявністю шуму
# шукає ядрові точки в щільних зонах
# точка є ядровою, якщо min_samples точок знаходяться в її
околі радіусом eps
# ядрові точки в околі eps утворюють кластер
m.fit(x)
print m.labels_ # шумові точки позначаються (-1)

#оцінка якості кластеризації
from sklearn.metrics.cluster import adjusted_rand_score
print adjusted_rand_score(y, m.labels_)
```

Результати:

```
[1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1]
[1]
[[6.4 6.7]
 [2.  2.2]]
```

```
[1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1]
[ 0  0  0  0  0  0  0  0  0 -1  1  1  1  1  1  1  1  1 -1]
0.7975322490185082
```

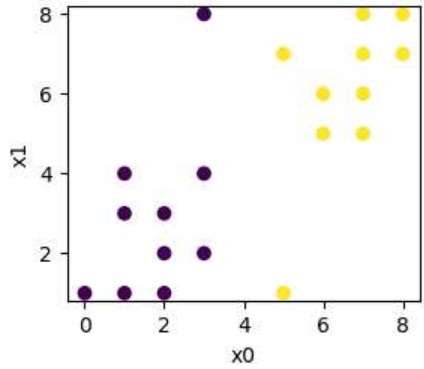


Рисунок 19 – Дані для кластеризації

20 ПІДГОТОВКА ДАНИХ

Деякі моделі потребують спеціальної підготовки даних. В прикладі preprocess.py показано методи **масштабування даних**: `MinMaxScaler` – масштабує в діапазон від 0 до 1, `StandardScaler` – масштабує так, щоб середнє було 0, а дисперсія 1, `RobustScaler` – те саме що `StandardScaler`, але ігнорує викиди (спостереження, які значно відрізняються від інших).

Більшість моделей працює краще, якщо ознаки і залежна змінна мають нормальний розподіл. Часто (особливо під час обробки дискретних даних) функції `log` і `exp` дозволяють досягти більш симетричного розподілу [8]. Застосовуйте їх для лінійних моделей, але не для моделей на основі дерев.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.preprocessing import StandardScaler,
RobustScaler, MinMaxScaler

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
```

```
y = np.array([9,0,2,6,9,8,2,9,4,9])

S=[]
S+=[MinMaxScaler()] # масштабує в діапазоні 0..1
S+=[StandardScaler()] # середнє 0, дисперсія 1
S+=[RobustScaler()] # те саме що StandardScaler, але
ігнорує викиди
for scaler in S:
    scaler.fit(x) # отримати модель для масштабування
    x_scaled=scaler.transform(x) # масштабувати
    print x_scaled
    print scaler.inverse_transform(x_scaled) # зворотне
перетворення

# Увага! Завжди застосовуйте fit для навчаючих даних
# і потім transform для навчаючих і тестових даних
```

Результати (масиви транспоновано):

```
[[0.88888889 0. 0.33333333 0.44444444 1. 0.77777778 0.11111111
0.66666667 0.33333333 1.]]
[[8. 0. 3. 4. 9. 7. 1. 6. 3. 9.]]
[[ 0.96824584 -1.61374306 -0.64549722 -0.32274861 1.29099445
0.64549722 -1.29099445 0.32274861 -0.64549722 1.29099445]]
[[8. 0. 3. 4. 9. 7. 1. 6. 3. 9.]]
[[ 0.63157895 -1.05263158 -0.42105263 -0.21052632 0.84210526
0.42105263 -0.84210526  0.21052632 -0.42105263  0.84210526]]
[[8. 0. 3. 4. 9. 7. 1. 6. 3. 9.]]
```

21 КОНВЕЄР

В прикладі pipeline.py показано використання конвеєра (`Pipeline`) для спрощення побудови і використання поліноміальної моделі. **Конвеєр** дозволяє об'єднати разом кілька операції обробки даних в єдину модель scikit-learn, яка має звичні атрибути `fit`, `predict`, `score`. До етапів конвеєра можна

звернутись за допомогою атрибуту `steps`. Створити конвеєр можна за допомогою функції `make_pipeline`. Програма створює конвеєр з операціями `PolynomialFeatures(degree=2)` та `LinearRegression()` і виводить прогноз для x=5, оцінку моделі, імена поліноміальних ознак та прогноз для x=5, який з метою перевірки обчислений за явно введеним поліномом.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])

model = make_pipeline(PolynomialFeatures(degree=2),
LinearRegression())
model.fit(x,y)
print model.predict([[5]])
print model.score(x,y)
step0=model.steps[0][-1]
print step0.get_feature_names()
step1=model.steps[1][-1]
a,b = step1.coef_, step1.intercept_
# перевірка
print a[0]+a[1]*5+a[2]*5**2+b
```

Результати:

```
[6.48914858]
0.9233988148307714
['1', 'x0', 'x0^2']
6.489148580968279
```

22 КОНВЕЄР ДВОВИМІРНОЇ МОДЕЛІ

Приклад pipeline2D.py є аналогічним до попереднього, але будує двовимірну поліноміальну модель.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

X=np.mgrid[0:10:1.0, 0:10:1.0]
x=X.reshape((2, X.size//2))
y = x[0]**2+x[1]**2+10*np.random.normal(size=x[0].shape)
x=x.T
model = make_pipeline(PolynomialFeatures(2),
LinearRegression())
model.fit(x, y)

s0,s1=model.steps[0][-1],model.steps[1][-1]
print s0.get_feature_names()
a,b=s1.coef_, s1.intercept_
# перевірка
print model.predict([[5.,5.]])
#print a,b
print a[0]+a[1]*5+a[2]*5+a[3]*25+a[4]*25+a[5]*25+b
```

Результати:

```
['1', 'x0', 'x1', 'x0^2', 'x0 x1', 'x1^2']
[49.72650669]
49.72650668982178
```

23 ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ

Основною задачею МН є пошук найкращої моделі з множини можливих. Якщо ця множина містить однотипні моделі, що відрізняються значеннями гіперпараметрів, то цю задачу можна розв'язати шляхом пошуку найкращих (**оптимальних**) значень цих гіперпараметрів. Найкраща модель володіє найбільшим значенням правильності (`score`) на тестових даних (рис. 20). Приклад paramOptimization.py показує два методи оптимізації гіперпараметра `alpha` моделі `Ridge` – сітковий метод оптимізації (оптимальне значення 32,7) та рандомізований (18,7).

Після оптимізації параметрів моделі потрібно її перевірити на **екзаменаційних** даних. Для цього на початку дані треба ділити на дві частини. Перша буде використовуватись для оптимізації, а друга – для перевірки найкращої моделі.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score

# емпіричні дані
x = np.array([0,1,2,3,4,5,6,7,8,9,10])[:, None]
y = np.array([0,1,2,3,4,5,6,7,8,9,30])

# пошук найкращого параметра alpha
alphas = np.logspace(-1, 2, 100) # масив параметрів
регуляризації
scores = []
for alpha in alphas: # для кожного alpha
    # середня правильність моделі шляхом перехресної
перевірки
    s=cross_val_score(Ridge(alpha), x, y, cv=3).mean()
    scores.append([s,alpha]) # додати у список
```

```
maxscore,alpha=max(scores, key=lambda s:s[0])
print maxscore, alpha # найбільша правильність і
відповідне alpha

import matplotlib.pyplot as plt
plt.plot(alphas, [s[0] for s in scores]) # залежність
score від alpha
plt.xlabel('alpha'),plt.ylabel('score')
plt.show()

# або
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(Ridge(), dict(alpha=alphas), cv=3)
# виконує перехрестну перевірку (CV) для кожного елементу
alphas
model.fit(x, y)
print model.best_score_ # середня правильність CV моделі,
яка побудована на навчальних даних CV
print model.best_params_
model.best_estimator_ # найкраща модель
print model.score(x,y) # правильність найкращої моделі
model.cv_results_ # усі результати пошуку

# або
from sklearn.linear_model import RidgeCV
model = RidgeCV(alphas=alphas, cv=3)
model.fit(x, y)
print model.alpha_

# або рандомізований пошук найкращих параметрів
from sklearn.model_selection import RandomizedSearchCV
model = RandomizedSearchCV(Ridge(), dict(alpha=alphas),
cv=3, n_iter=10)
model.fit(x, y)
print model.best_score_
```

```
print model.best_params_

# вкладена перехресна перевірка [Мюллер с.292]
scores = cross_val_score(RidgeCV(alphas=alphas, cv=3), x,
y, cv=3)
print scores.mean()
```

Результати:

```
-2.643160396372387 32.74549162877728
-2.8177579339917127
{'alpha': 32.74549162877728}
0.5638233693613304
32.74549162877728
-4.457309216790754
{'alpha': 18.73817422860385}
-7.333380237138638
```

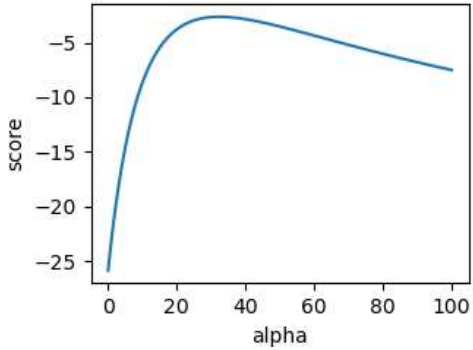


Рисунок 20 – Залежність `score` від `alpha`

24 ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ З КОНВЕЄРОМ

Приклад paramOptimizationPipeline.py показує задачу оптимізації гіперпараметрів моделі, яка побудована за допомогою конвеєра. Конвеєр об'єднує операції `StandardScaler`, `PolynomialFeatures` та `Ridge`. Модель має два гіперпараметра – степінь полінома `polynomialfeatures__degree` та параметр

регуляризації ridge__alpha. Для оптимізації використовується сітковий метод – розглядаються усі можливі комбінації значень параметрів polynomialfeatures__degree (1, 2, 3) та ridge__alpha (0,001, 0,01, 0,1). Результати показують, що найкращою моделлю є модель зі значеннями: ridge__alpha=0,1, polynomialfeatures__degree=2. Її правильність 0,67.

Також не забувайте, що після оптимізації параметрів моделі потрібно її перевірити на екзаменаційних даних.

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV

x = np.array([8,0,3,4,9,7,1,6,3,9])[:, None]
y = np.array([9,0,2,6,9,8,2,9,4,9])
# розділити дані випадково
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2,random_state=1)

pipe =
make_pipeline(StandardScaler(),PolynomialFeatures(),
Ridge())
param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'ridge__alpha': [0.001, 0.01, 0.1]}
grid = GridSearchCV(pipe, param_grid=param_grid, cv=3,
n_jobs=1)
grid.fit(x_train, y_train)# Увага!!! Усі можливі
комбінації параметрів
print grid.score(x_test, y_test) # або cross_val_score
#grid.fit(x, y) # або використовувати усі дані
```

```
print grid.best_params_
```

Результати:

```
0.6732729931663997
{'ridge__alpha': 0.1, 'polynomialfeatures__degree': 2}
```

25 ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ ДВОВИМІРНОЇ МОДЕЛІ З КОНВЕЄРОМ

Приклад paramOptimizationPipeline2D.py подібний на попередній, але будується двовимірна модель (рис. 21) – з двома ознаками. Додатково, після оптимізації гіперпараметрів, модель навчається на усіх даних. Програма виводить: правильність моделі на тестових даних, оптимальні параметри, імена поліноміальних ознак, відповідні коефіцієнти моделі та прогноз в точці (5, 5).

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV

X=np.mgrid[0:10:1.0, 0:10:1.0]
x=X.reshape((2, X.size//2))
y = x[0]**2+x[1]**2+10*np.random.normal(size=x[0].shape)
x=x.T
#
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2)
```

```
pipe =
make_pipeline(StandardScaler(),PolynomialFeatures(),
Ridge())
param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'ridge__alpha': [0.001, 0.01, 0.1]}
grid = GridSearchCV(pipe, param_grid=param_grid, cv=3,
n_jobs=1)
grid.fit(x_train, y_train)

print grid.score(x_test, y_test)
print grid.best_params_
model=grid.best_estimator_

model.fit(x,y) # fit на усіх даних - інша модель!!
s0,s1,s2=model.steps[0][-1],model.steps[1][-
1],model.steps[2][-1]
print s1.get_feature_names()
a, b = s2.coef_, s2.intercept_
print a,b

from mpl_toolkits.mplot3d import axes3d
fig = plt.figure() # створити фігуру
ax = fig.add_subplot(111, projection='3d') # додати графік
3D
Y=model.predict(x)
Y=Y.reshape(X[0].shape)
ax.scatter(x[:,0], x[:,1], y) # показати емпіричні точки
ax.plot_wireframe(X[0], X[1], Y, rstride=1, cstride=1) #
показати теоретичну поверхню
ax.set_xlabel('X0'),ax.set_ylabel('X1'),ax.set_zlabel('Y')
plt.show()

# перевірка в точці [5.,5.]
# Увага! Модель стандартизована
print model.predict([[5.,5.]])
```

```
x=s0.transform([[5.,5.]]) # стандартизувати
# або x=([[5.,5.]]-s0.mean_)/s0.scale_
x=x.ravel()
print
a[0]+a[1]*x[0]+a[2]*x[1]+a[3]*x[0]**2+a[4]*x[0]*x[1]+a[5]*
x[1]**2+b
```

Результати:

```
0.9002114602859088
{'ridge__alpha': 0.01, 'polynomialfeatures__degree': 2}
['1',  'x0',      'x1',    'x0^2',    'x0 x1',    'x1^2'  ]
[ 0. 25.10557193 24.91765 8.95162019 -0.96012827 7.09519125]
40.10193135141922
[49.26702879]
49.26702878691941
```

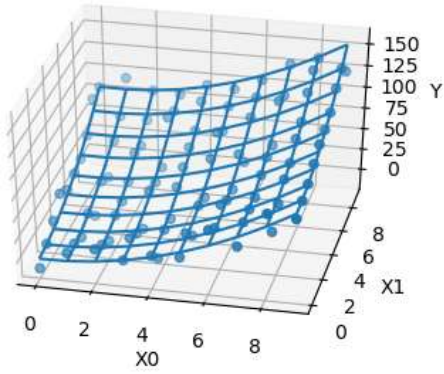


Рисунок 21 – Поліноміальна модель з оптимальними гіперпараметрами

ЗАДАЧІ

1. Створити одновимірну лінійну регресійну модель даних, які генеруються за допомогою коду:

```
x=np.arange(0, 20)
y=a*x+b+np.random.normal(0, 1+c, 20)
```

Значення a, b, c наведено в таблиці 3 відповідно до варіанта завдання. Вивести коефіцієнти моделі, коефіцієнт детермінації, виконати прогноз в довільній точці та побудувати на графіку точки даних і модель.

Таблиця 3 – Значення a, b, c, d для різних варіантів *n*

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
a	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3
b	1	1	2	2	3	3	0	0	2	2	3	3	0	0	1	1	3	3	0	0	1	1	2	2
c	2	3	1	3	1	2	2	3	0	3	0	2	1	3	0	3	0	1	1	2	0	2	0	1
d	3	2	3	1	2	1	3	2	3	0	2	0	3	1	3	0	1	0	2	1	2	0	1	0

2. Створити двовимірну лінійну регресійну модель даних, які генеруються за допомогою коду:

```
x =
np.array([[0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4],
          [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4]])
y = a*x[0] + b*x[1] + c + np.random.normal(0, 1+d, 25)
```

Значення a, b, c, d наведено в таблиці 3 відповідно до варіанта завдання. Вивести коефіцієнти моделі, коефіцієнт детермінації, виконати прогноз в довільній точці та побудувати на графіку точки даних і модель.

3. Шляхом перехресної перевірки знайти найкращу одновимірну поліноміальну регресійну модель даних, які генеруються за допомогою коду:

```
x=np.arange(0, 20)
y=a*x**3+b*x**2+c*x+d+np.random.normal(0, 10**3, 20)
```

Значення a, b, c, d наведено в таблиці 3 відповідно до варіанта завдання. Вивести коефіцієнти моделі, коефіцієнт детермінації,

виконати прогноз в довільній точці та побудувати на графіку точки даних і модель.

4. Шляхом перехресної перевірки знайти найкращу двовимірну поліноміальну регресійну модель даних, які генеруються за допомогою коду:

```
x =
np.array([[0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4],
          [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4]])
y = a*x[0]**2 + b*x[1]**2 + c*x[0]*x[1] + d*x[0] + a*x[1] + b +
np.random.normal(0, 1+d, 25)
```

Значення a, b, c, d наведено в таблиці 3 відповідно до варіанта завдання. Вивести коефіцієнти моделі, коефіцієнт детермінації, виконати прогноз в довільній точці та побудувати на графіку точки даних і модель.

5. Створити двовимірну лінійну модель класифікації даних, які генеруються за допомогою коду:

```
x =
np.array([[0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4],
          [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4]])
y = a*x[0] + b*x[1] + c + np.random.normal(0, 1+d, 25)
y = (y>y.mean())*1
```

Значення a, b, c, d наведено в таблиці 3 відповідно до варіанта завдання. Вивести коефіцієнти моделі, метрики класифікації, виконати прогноз в довільній точці та побудувати на графіку точки даних і границю прийняття рішень.

6. Шляхом перехресної перевірки знайти найкращу двовимірну модель класифікації даних, які генеруються за допомогою коду:

```
x =
np.array([[0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4],
          [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4]])
```

```
y = a*x[0]**2 + b*x[1]**2 + c*x[0]*x[1] + d*x[0] + a*x[1] + b +
np.random.normal(0, 1+d, 25)
y = (y>y.mean())*1
```

Значення a, b, c, d наведено в таблиці 3 відповідно до варіанта завдання. Вивести коефіцієнти моделі, метрики класифікації, виконати прогноз в довільній точці та побудувати на графіку точки даних і границю прийняття рішень.

7. Розв`язати задачу зменшення розмірності даних, які генеруються за допомогою коду:

```
x0 = np.linspace(0,10,100)
x1 = a*x0 + b + np.random.normal(size=x0.size)
x = np.vstack([x0, x1])
```

Побудувати графіки з точками початкових і перетворених даних.

8. Розв`язати задачу кластеризації даних, які генеруються за допомогою коду:

```
s=0.5
x0=np.hstack([np.random.normal(a,s,10), np.random.normal(b,s,10)])
x1=np.hstack([np.random.normal(c,s,10), np.random.normal(d,s,10)])
x=np.vstack([x0, x1])
```

Побудувати графік з точками даних і мітками кластерів. Виконати прогноз в довільній точці. Обчислити оцінку якості кластеризації.

9. Розв`язати задачі 3 і 4 з використанням конвеєра. Побудувати криві перевірки і навчання. Застосувати вбудовані засоби оптимізації гіперпараметрів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методи та системи штучного інтелекту: Навчальний посібник для студентів напряму підготовки 6.050101 «Комп'ютерні науки» / Уклад. : А.С. Савченко, О. О. Синельніков. Київ : НАУ, 2017. 190 с.
2. Харченко В. О. Основи машинного навчання : навч. посіб. Суми : Сумський державний університет, 2023. 264 с.
3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка». Харків: ХНУ імені В. Н. Каразіна, 2020. 301 с.
4. Копей В. Б. Мова програмування Python для інженерів і науковців : Навчальний посібник. Івано-Франківськ : ІФНТУНГ, 2019. 274 с.
5. Костюченко А. О. Основи програмування мовою Python: навчальний посібник. Чернігів : ФОП Баликіна С.М., 2020. 180 с.
6. Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий; за ред. А. В. Анісімова. Київ : Видавничо-поліграфічний центр "Київський університет", 2014. 640 с.
7. scikit-learn: machine learning in Python. URL: https://scikit-learn.org/0.19/
8. Müller A., Guido S. Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, 2016. 392 p.
9. Harrison M. Machine Learning Pocket Reference: Working with Structured Data in Python. O'Reilly Media, 2019. 320 p.
10. Bruce P., Bruce A., Gedeck P. Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python. O'Reilly Media, 2020. 368 p.
11. Aurélien Géron. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2017. 564 p.
12. Raschka S., Mirjalili V. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. 3 ed. Packt Publishing, 2019. 770 p.

13. Avila J. scikit-learn Cookbook: Over 80 recipes for machine learning in Python with scikit-learn. 2 ed. Packt Publishing, 2017. 374 p.
14. Chollet F. Deep Learning with Python. 2 ed. Manning Publications, 2021. 504 p.