ЗАТВЕРДЖЕНО

Науково-методичною радоюДержавного університету «Житомирська політехніка»

протокол від 24 травня 2023 р. № 8

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ
для виконання лабораторних робіт з навчальної дисципліни «Штучний інтелект в задачах комп'ютерної інженерії»

для здобувачів вищої освіти освітнього ступеня «магістр»
спеціальності код спеціальності «123 – Комп'ютерна інженерія»
освітньо-професійна програма «Комп'ютерна інженерія»
факультет інформаційно-комп'ютерних технологій
(назва факультету)
кафедра комп'ютерної інженерії та кібербезпеки
(назва кафедри)

Рекомендовано на засіданні кафедри комп'ютерної інженерії та кібербезпеки
(назва кафедри)
_22_лютого___ 2023 р.,
протокол № _2_

Розробники:кандидат технічних наук, доцент І. В. Пулеко,
доктор технічних наук, доцент В. В. Воротніков
кандидат економічних наук, доцент О. М. Свінцицька

Житомир
2023

ЗМІСТ

ВСТУП

На сьогоднішній день методи штучного інтелекту широко застосовуються для рішення цілого ряду задач комп'ютерної інженерії, тому вивчення їх дозволить майбутнім фахівцям оволодіти перспективними технологіями в обраній спеціалізації.

Метою навчальної дисципліни «Штучний інтелект в задачах комп'ютерної інженерії» є набуття студентами знань, умінь і здатностей (компетенцій) щодо розробки та застосування методів штучного інтелекту в задачах комп'ютерної інженерії для ефективного вирішення завдань професійної діяльності.

Розроблені методичні рекомендації призначені для студентів освітнього рівня «магістр», які навчаються за спеціальністю 123 – Комп'ютерна інженерія кафедри комп'ютерної інженерії та кібербезпеки факультету інформаційно-комп'ютерних технологій.

У частині 1 викладено теоретичний матеріал для підготовки, завдання та методичні рекомендації до виконання лабораторних робіт № 1-4 що присвячені темі «Машинне навчання». Лабораторні роботи розраховані на 4 год. і направлені на розвиток фахових компетентностей (відповідно до Стандарту вищої освіти України: другий (магістерський) рівень, галузь знань 12 Інформаційні технології, спеціальність 123 Комп'ютерна інженерія, який Затверджено і введено в дію наказом Міністерства освіти і науки України від 18.03.2021 р. № 330) :

Інтегральна компетентність: Здатність розв'язувати складні задачі і проблеми в галузі комп'ютерної інженерії або у процесі навчання, що передбачає проведення досліджень та/або здійснення інновацій та характеризується невизначеністю умов і вимог.

Загальні компетентності

ЗК2. Здатність до абстрактного мислення, аналізу і синтезу.

Спеціальні (фахові, предметні) компетентності

СК2. Здатність розробляти алгоритмічне та програмне забезпечення, компоненти комп'ютерних систем та мереж, Інтернет додатків, кіберфізичних систем з використанням сучасних методів і мов програмування, а також засобів і систем автоматизації проектування.

СК6. Здатність використовувати та впроваджувати нові технології, включаючи технології розумних, мобільних, зелених і безпечних обчислень, брати участь в модернізації та реконструкції комп'ютерних систем та мереж, різноманітних вбудованих і розподілених додатків, зокрема з метою підвищення їх ефективності.

Після вивчення дисципліни студенти повинні отримати такі *результати навчання:*

РН4. Застосовувати спеціалізовані концептуальні знання, що включають сучасні наукові здобутки у сфері комп'ютерної інженерії, необхідні для

професійної діяльності, оригінального мислення та проведення досліджень, критичного осмислення проблем інформаційних технологій та на межі галузей знань.

РН9. Розробляти програмне забезпечення для вбудованих і розподілених застосувань, мобільних і гібридних систем.

Підгунтям для виконання лабораторних робіт є базові знання освітнього рівня «бакалавр» з основ програмування мовою Pythoni дискретної математики та оптимізації.

ПІДГОТОВКА ДО ЛАБОРАТОРНИХ РОБІТ

До початку проведення лабораторних робітвиконайте підготовчі роботи

1. Встановіть актуальну версію Python

Використовуємо командний рядок Windows.
Поміняйте джерело pip на домашнє дзеркало та оновіть pip

```
Microsoft Windows [Version 10.0.19042.1348]
(c) Корпорація Майкрософт. Усі права захищені.

C:\Users\I-11> python -m pip install -upgrade pip
```

```
Microsoft Windows [Version 10.0.19042.1348]
(c) Корпорація Майкрософт. Усі права захищені.

C:\Users\I-11> python -m pip install -upgrade pip

Usage:
  C:\Python39\python.exe -m pip install [options] <requirement specifier> [package-index-options] ...
  C:\Python39\python.exe -m pip install [options] -r <requirements file> [package-index-options] ...
  C:\Python39\python.exe -m pip install [options] [-e] <vcs project url> ...
  C:\Python39\python.exe -m pip install [options] [-e] <local project path> ...
  C:\Python39\python.exe -m pip install [options] <archive url/path> ...

no such option: -u
```

Потім використовуйте команду cd, щоб перейти в папку Scripts місця установки Python ython (знайдіть каталог установки Python). В цьому каталозі виконуютьсянаступні установки бібліотек функцій python.

2. Встановіть необхідні бібліотеки, що будуть використовуватись

Для цього скористайтеся командою:

pip install numpy scipy matplotlib ipython scikit-learn pandas

```
C:\Users\I-11>cd C:\Python39\Scripts

C:\Python39\Scripts>pip install numpy scipy matplotlib ipython scikit-learn pandas
Collecting numpy
  Downloading numpy-1.21.4-cp39-cp39-win_amd64.whl (14.0 MB)
     |████████████████████████████████| 14.0 MB 1.6 MB/s
Collecting scipy
  Downloading scipy-1.7.2-cp39-cp39-win_amd64.whl (34.3 MB)
     |████████████████████████████████| 34.3 MB 288 kB/s
Collecting matplotlib
  Downloading matplotlib-3.5.0-cp39-cp39-win_amd64.whl (7.2 MB)
     |████████████████████████████████| 7.2 MB 3.3 MB/s
Collecting ipython
  Downloading ipython-7.29.0-py3-none-any.whl (790 kB)
     |████████████████████████████████| 790 kB 2.2 MB/s
Collecting scikit-learn
  Downloading scikit_learn-1.0.1-cp39-cp39-win_amd64.whl (7.2 MB)
     |████████████████████████████████| 7.2 MB 1.3 MB/s
```

```
Collecting pandas
  Downloading pandas-1.3.4-cp39-cp39-win_amd64.whl (10.2 MB)
     |████████████████████████████████| 10.2 MB 683 kB/s
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.3.2-cp39-cp39-win_amd64.whl (52 kB)
     |████████████████████████████████| 52 kB 271 kB/s
Collecting setuptools-scm>=4
  Downloading setuptools_scm-6.3.2-py3-none-any.whl (33 kB)
Collecting fonttools>=4.22.0
  Downloading fonttools-4.28.1-py3-none-any.whl (873 kB)
     |████████████████████████████████| 873 kB 1.6 MB/s
Collecting cycler>=0.10
  Downloading cycler-0.11.0-py3-none-any.whl (6.4 kB)
Collecting cycler>=0.10
  Downloading cycler-0.11.0-py3-none-any.whl (6.4 kB)
Collecting packaging>=20.0
  Downloading packaging-21.3-py3-none-any.whl (40 kB)
     |████████████████████████████████| 40 kB 1.3 MB/s
Collecting pyparsing>=2.2.1
  Downloading pyparsing-3.0.6-py3-none-any.whl (97 kB)
     |████████████████████████████████| 97 kB 1.7 MB/s
Collecting python-dateutil>=2.7
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
     |████████████████████████████████| 247 kB 2.2 MB/s
Collecting pillow>=6.2.0
  Downloading Pillow-8.4.0-cp39-cp39-win_amd64.whl (3.2 MB)
     |████████████████████████████████| 3.2 MB 2.2 MB/s
Collecting backcall
  Downloading backcall-0.2.0-py2.py3-none-any.whl (11 kB)
Collecting decorator
  Downloading decorator-5.1.0-py3-none-any.whl (9.1 kB)
Collecting prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0
  Downloading prompt_toolkit-3.0.22-py3-none-any.whl (374 kB)
     |████████████████████████████████| 374 kB 2.2 MB/s
Collecting jedi>=0.16
  Downloading jedi-0.18.1-py2.py3-none-any.whl (1.6 MB)
     |████████████████████████████████| 1.6 MB 1.3 MB/s
Collecting colorama
  Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting traitlets>=4.2
  Downloading traitlets-5.1.1-py3-none-any.whl (102 kB)
     |████████████████████████████████| 102 kB 2.2 MB/s
Collecting pickleshare
  Downloading pickleshare-0.7.5-py2.py3-none-any.whl (6.9 kB)
Requirement already satisfied: setuptools>=18.5 in c:\python39\lib\site-packages (from ipython) (57.4.0)
Collecting matplotlib-inline
  Downloading matplotlib_inline-0.1.3-py3-none-any.whl (8.2 kB)
Collecting pygments
  Downloading Pygments-2.10.0-py3-none-any.whl (1.0 MB)
     |████████████████████████████████| 1.0 MB 2.2 MB/s
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-3.0.0-py3-none-any.whl (14 kB)
Collecting joblib>=0.11
  Downloading joblib-1.1.0-py2.py3-none-any.whl (306 kB)
     |████████████████████████████████| 306 kB 2.2 MB/s
Collecting pytz>=2017.3
  Downloading pytz-2021.3-py2.py3-none-any.whl (503 kB)
     |████████████████████████████████| 503 kB 2.2 MB/s
Collecting parso<0.9.0,>=0.8.0
  Downloading parso-0.8.2-py2.py3-none-any.whl (94 kB)
     |████████████████████████████████| 94 kB 865 kB/s
Collecting wcwidth
  Downloading wcwidth-0.2.5-py2.py3-none-any.whl (30 kB)
Collecting six>=1.5
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting tomli>=1.0.0
  Downloading tomli-1.2.2-py3-none-any.whl (12 kB)
Installing collected packages: pyparsing, wcwidth, traitlets, tomli, six, parso, packaging, numpy, threadpoolctl, setupt
ools-scm, scipy, pytz, python-dateutil, pygments, prompt-toolkit, pillow, pickleshare, matplotlib-inline, kiwisolver, jo
blib, jedi, fonttools, decorator, cycler, colorama, backcall, scikit-learn, pandas, matplotlib, ipython
Successfully installed backcall-0.2.0 colorama-0.4.4 cycler-0.11.0 decorator-5.1.0 fonttools-4.28.1 ipython-7.29.0 jedi-
0.18.1 joblib-1.1.0 kiwisolver-1.3.2 matplotlib-3.5.0 matplotlib-inline-0.1.3 numpy-1.21.4 packaging-21.3 pandas-1.3.4 p
arso-0.8.2 pickleshare-0.7.5 pillow-8.4.0 prompt-toolkit-3.0.22 pygments-2.10.0 pyparsing-3.0.6 python-dateutil-2.8.2 py
tz-2021.3 scikit-learn-1.0.1 scipy-1.7.2 setuptools-scm-6.3.2 six-1.16.0 threadpoolctl-3.0.0 tomli-1.2.2 traitlets-5.1.1
 wcwidth-0.2.5
WARNING: You are using pip version 21.2.3; however, version 21.3.1 is available.
You should consider upgrading via the 'C:\Python39\python.exe -m pip install --upgrade pip' command.

C:\Python39\Scripts>
```

Також у деяких випадках ми будемо використовувати додаткову бібліотеку для побудови графіків «mglearn» та «seaborn».
Встановіть її командою

pip install mglearn

```
c:\Python39\Scripts>pip install mglearn
Collecting mglearn
  Downloading mglearn-0.1.9.tar.gz (540 kB)
     |████████████████████████████████| 540 kB 1.7 MB/s
Requirement already satisfied: numpy in c:\python39\lib\site-packages (from mglearn) (1.21.4)
Requirement already satisfied: matplotlib in c:\python39\lib\site-packages (from mglearn) (3.5.0)
Requirement already satisfied: scikit-learn in c:\python39\lib\site-packages (from mglearn) (1.0.1)
Requirement already satisfied: pandas in c:\python39\lib\site-packages (from mglearn) (1.3.4)
Requirement already satisfied: pillow in c:\python39\lib\site-packages (from mglearn) (8.4.0)
Requirement already satisfied: cycler in c:\python39\lib\site-packages (from mglearn) (0.11.0)
Collecting imageio
  Downloading imageio-2.13.2-py3-none-any.whl (3.3 MB)
     |████████████████████████████████| 3.3 MB 6.4 MB/s
Requirement already satisfied: joblib in c:\python39\lib\site-packages (from mglearn) (1.1.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\python39\lib\site-packages (from matplotlib->mglearn) (4.28.1)
Requirement already satisfied: packaging>=20.0 in c:\python39\lib\site-packages (from matplotlib->mglearn) (21.3)
Requirement already satisfied: setuptools-scm>=4 in c:\python39\lib\site-packages (from matplotlib->mglearn) (6.3.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\python39\lib\site-packages (from matplotlib->mglearn) (1.3.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\python39\lib\site-packages (from matplotlib->mglearn) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\python39\lib\site-packages (from matplotlib->mglearn) (3.0.6)
Requirement already satisfied: six>=1.5 in c:\python39\lib\site-packages (from python-dateutil>=2.7->matplotlib->mglear
) (1.16.0)
Requirement already satisfied: tomli>=1.0.0 in c:\python39\lib\site-packages (from setuptools-scm>=4->matplotlib->mglea
n) (1.2.2)
Requirement already satisfied: setuptools in c:\python39\lib\site-packages (from setuptools-scm>=4->matplotlib->mglearn
 (57.4.0)
Requirement already satisfied: pytz>=2017.3 in c:\python39\lib\site-packages (from pandas->mglearn) (2021.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\python39\lib\site-packages (from scikit-learn->mglearn) (3.0.
```

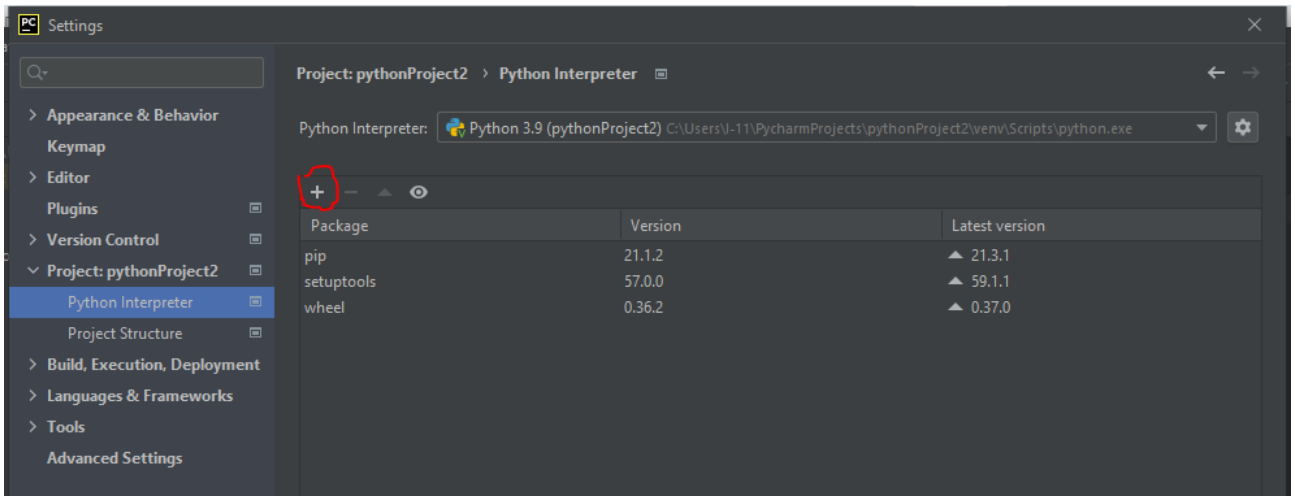
c:\Python39\Scripts> pip install seaborn

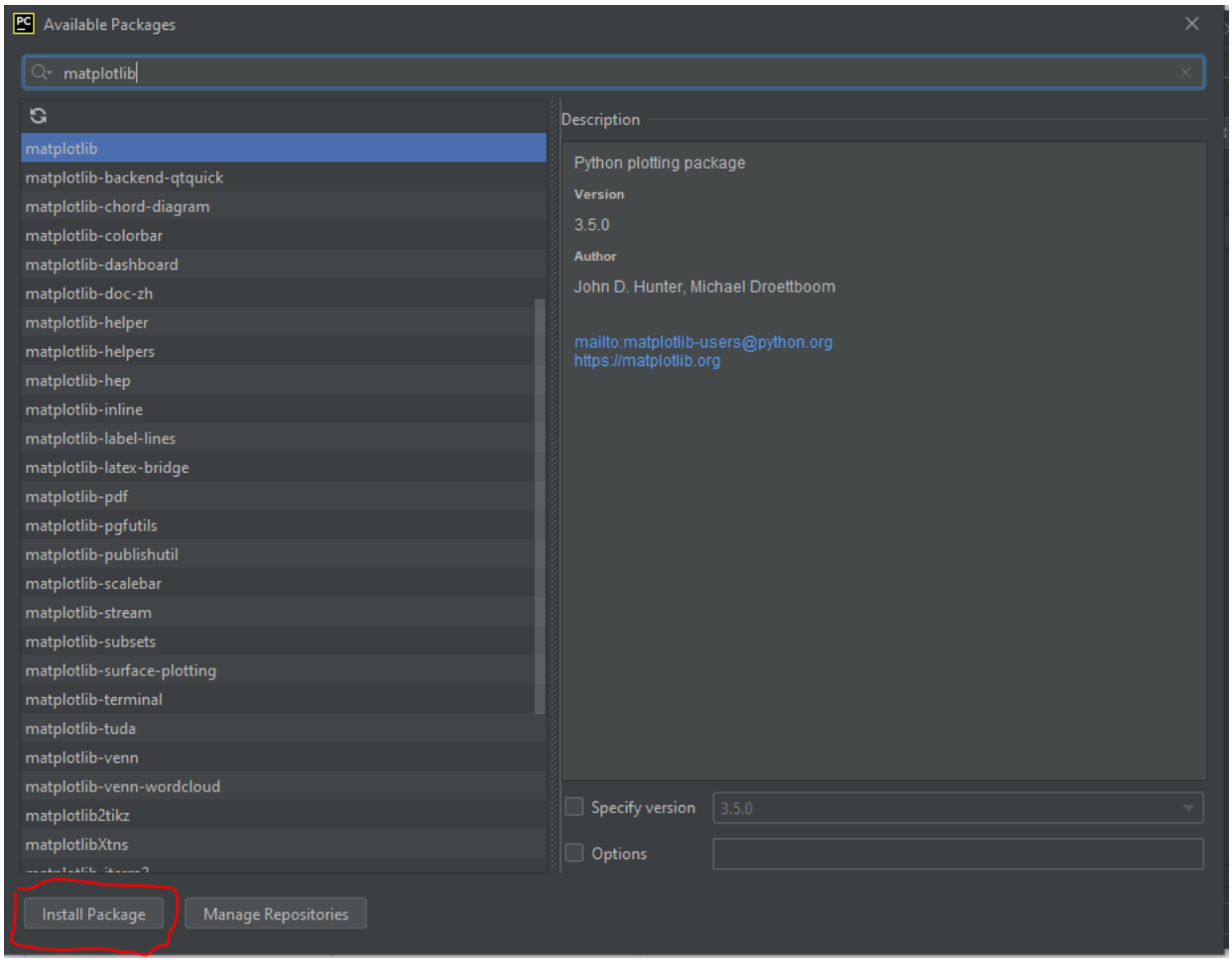
3. Встановіть PyCharm

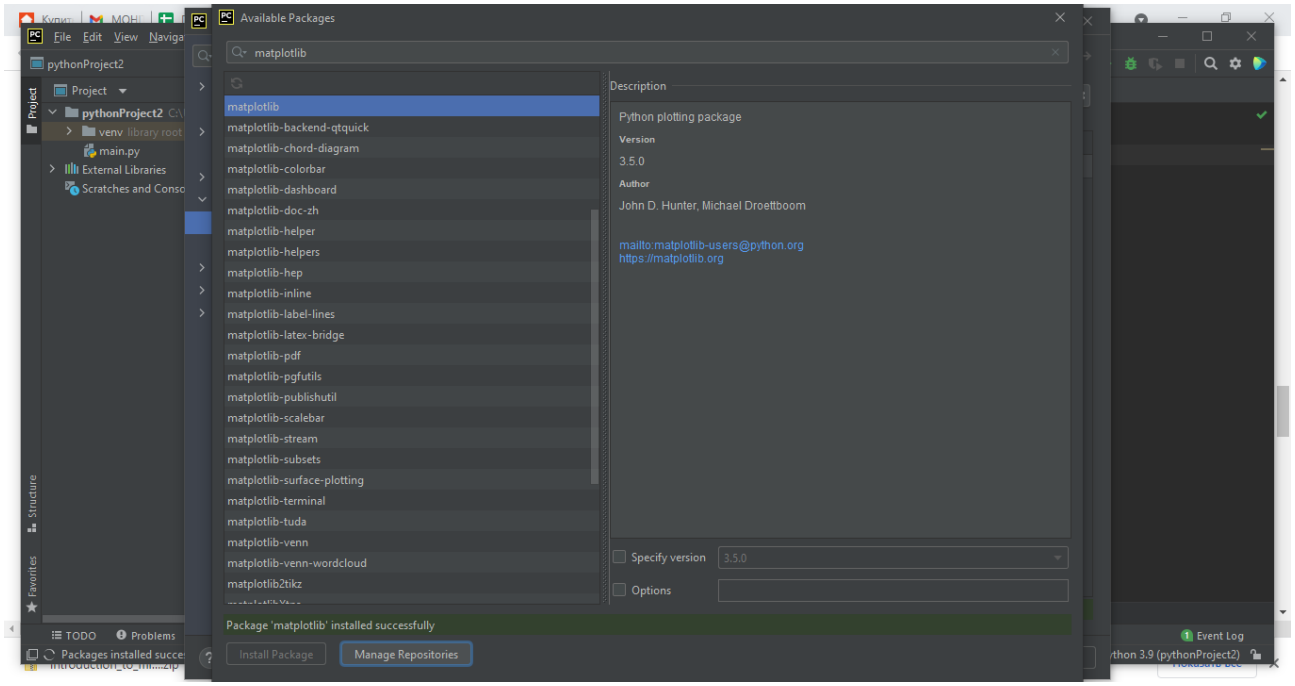
Встановіть PyCharm відповідно до інструкції з установки.

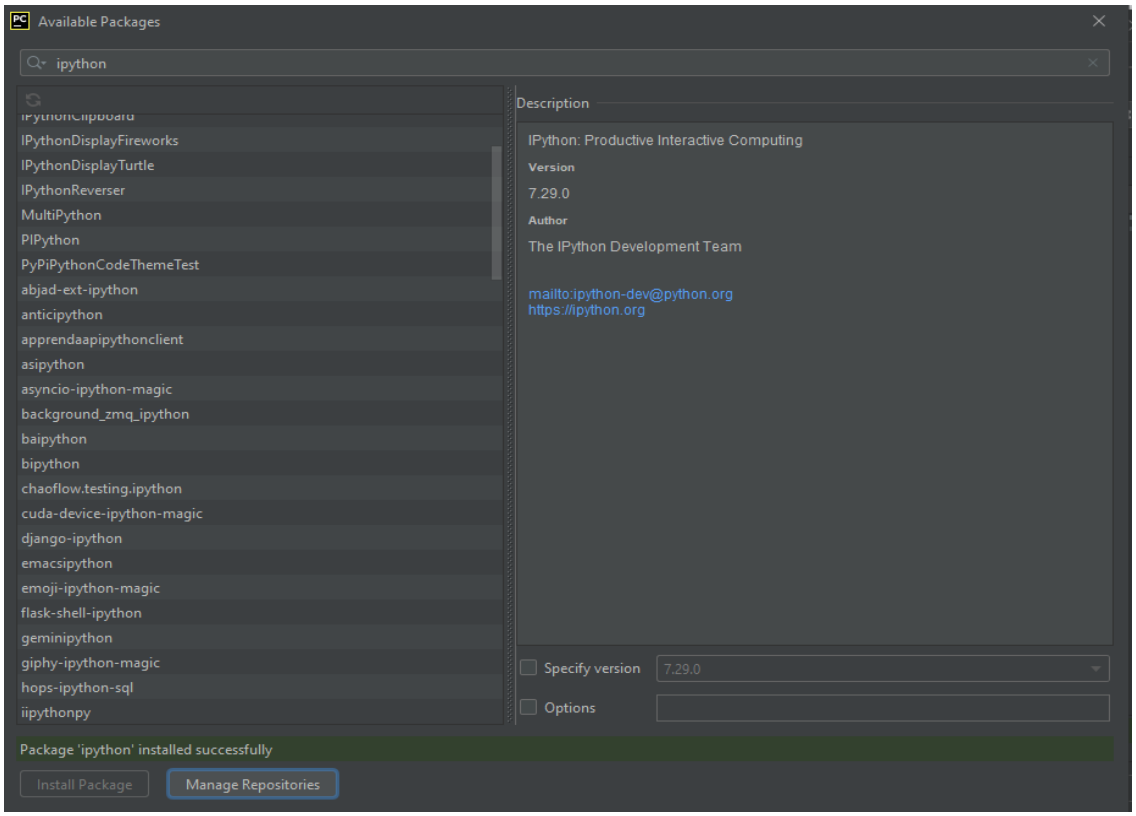
4. ПідключітьбібліотекивPyCharmдопроекту

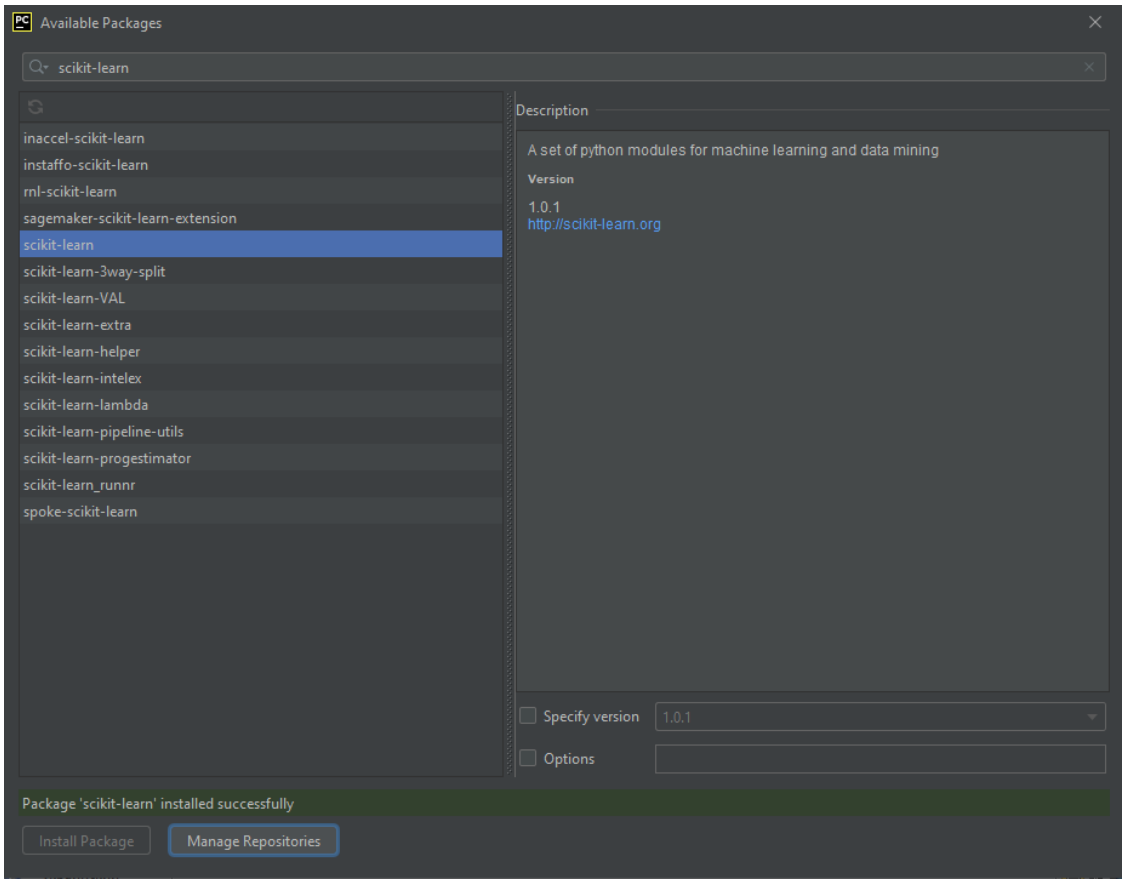
ВідкрийтеPyCharm, потімвідкрийтеFile->Settingsінайдітьпроект, якпоказанонарисунку. Простослідуйтеінструкціямнарисунках.

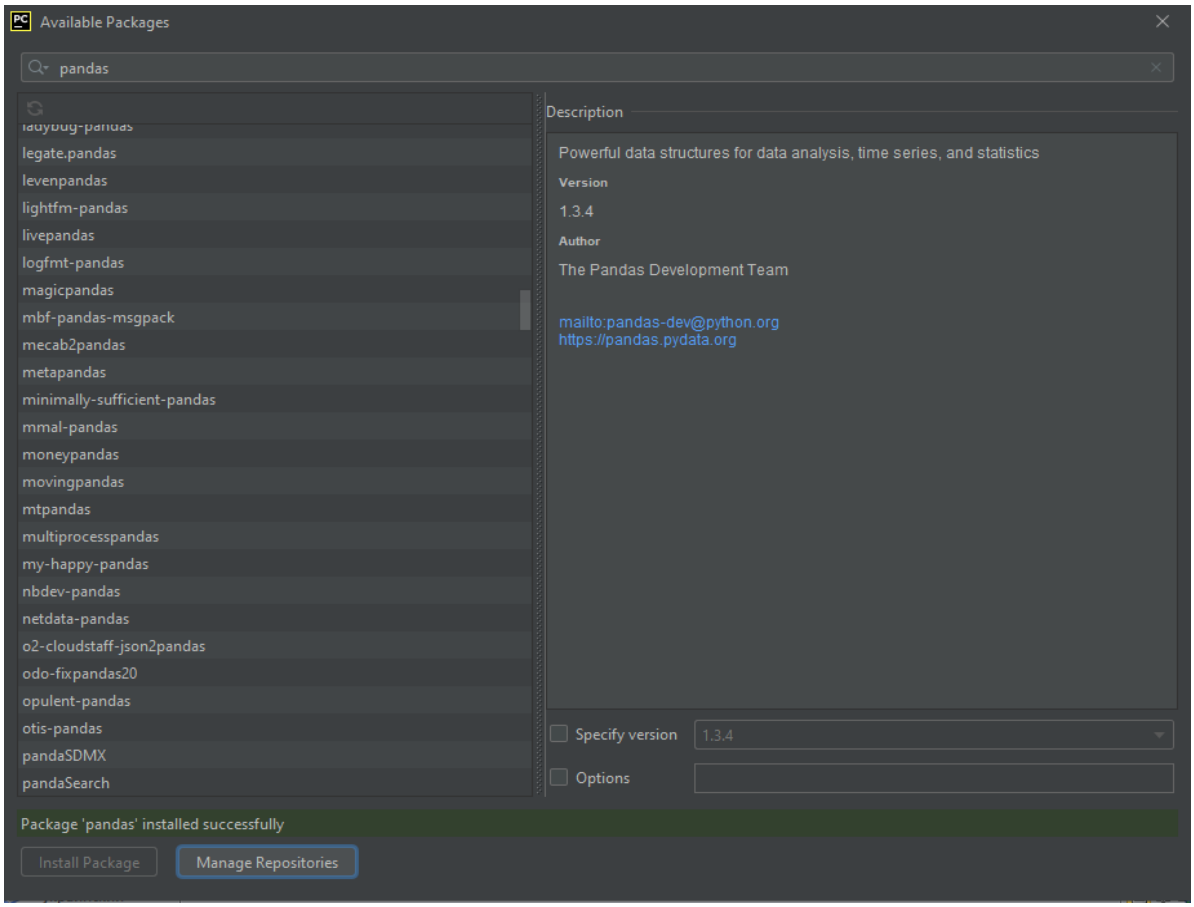



Available Packages
matplotlib
matplotlib
matplotlib-backend-qtquick
matplotlib-chord-diagram
matplotlib-colorbar
matplotlib-dashboard
matplotlib-doc-zh
matplotlib-helper
matplotlib-helpers
matplotlib-hep
matplotlib-inline
matplotlib-label-lines
matplotlib-latex-bridge
matplotlib-pdf
matplotlib-pgfutils
matplotlib-publishutil
matplotlib-scalebar
matplotlib-stream
matplotlib-subsets
matplotlib-surface-plotting
matplotlib-terminal
matplotlib-tuda
matplotlib-venn
matplotlib-venn-wordcloud
matplotlib2tikz
matplotlibXtns
Description
Python plotting package
Version
3.5.0
Author
John D. Hunter, Michael Droettboom
mailto:matplotlib-users@python.org
https://matplotlib.org
Specify version
3.5.0
Options
Install Package
Manage Repositories


Available Packages
matplotlib
Description
Python plotting package
Version
3.5.0
Author
John D. Hunter, Michael Droettboom
mailto:matplotlib-users@python.org
https://matplotlib.org
Specify version
3.5.0
Options
Package 'matplotlib' installed successfully
Install Package
Manage Repositories


Available Packages
ipython
IPythonDisplayFireworks
IPythonDisplayTurtle
IPythonReverser
MultiPython
PIPython
PyPiPythonCodeThemeTest
abjad-ext-ipython
anticipython
apprendaapipythonclient
asipython
asyncio-ipython-magic
background_zmq_ipython
baipython
bipython
chaoflow.testing.ipython
cuda-device-ipython-magic
django-ipython
emacsipython
emoji-ipython-magic
flask-shell-ipython
geminipython
giphy-ipython-magic
hops-ipython-sql
iipythonpy
Description
IPython: Productive Interactive Computing
Version
7.29.0
Author
The IPython Development Team
mailto:ipython-dev@python.org
https://ipython.org
Specify version
7.29.0
Options
Package 'ipython' installed successfully
Install Package
Manage Repositories


Available Packages
scikit-learn
inaccel-scikit-learn
instaffo-scikit-learn
rnl-scikit-learn
sagemaker-scikit-learn-extension
scikit-learn
scikit-learn-3way-split
scikit-learn-VAL
scikit-learn-extra
scikit-learn-helper
scikit-learn-intelex
scikit-learn-lambda
scikit-learn-pipeline-utils
scikit-learn-progestimator
scikit-learn_runnr
spoke-scikit-learn
Description
A set of python modules for machine learning and data mining
Version
1.0.1
http://scikit-learn.org
Specify version
1.0.1
Options
Package 'scikit-learn' installed successfully
Install Package
Manage Repositories



Аналогічним чином додайте бібліотеку для побудови графіків «mglearn».

5. Перевірка працездатності встановлених бібліотек

Перевірте чи працює numpy

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6]])
print("x:\n{}".format(x))
```

Результат, що повинен бути:

```
x:
[[1 2 3]
 [4 5 6]]
```

ЛАБОРАТОРНА РОБОТА № 1
ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

Мета роботи: *використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та методи класифікації даних*.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії попередньої обробки та матеріал поданий у цьому підрозділі. Також доцільно вивчити матеріал поданий в літературі:

Alberto Artasanchez, Prateek Joshi. Artifcial Intelligence with Python. Second Edition. BIRMINGHAM – MUMBAI:Packt Publishing 2020. – 592 p. ISBN 978-1-83921-953-5.

При виконанні роботи можна використовувати Google Colab або Jupiter Notebook.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 1 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 1.1. Попередня обробка даних

Як правило, при обробці ми маємо справу з великими обсягами необроблених вихідних даних. Алгоритми машинного навчання розраховані на те, що, перш ніж вони зможуть розпочати процес тренування, отримані дані будуть відформатовані певним чином [1]. Щоб привести дані до форми, що прийнятна для алгоритмів машинного навчання, ми повинні попередньо підготувати їх і перетворити на потрібний формат. Покажемо, як це робиться.

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
from sklearn import preprocessing
```

Визначимо деяку вибірку даних.

```
input_data = np.array([[5.1, -2.9, 3.3],
                       [-1.2, 7.8, -6.1],
                       [3.9, 0.4, 2.1],
                       [7.3, -9.9, -4.5]])
```

Розглянемо декілька різних методів попередньої обробки даних

1.1.1. Бінарізація

Цей процес застосовується в тих випадках, коли ми хочемо перетворити наші числові значення на булеві значення (0, 1). Скористаємося вбудованим методом для бінаризації вхідних даних, встановивши значення 2,1 як порогове.

Додамо наступні рядки до того ж файлу Python.

```
# Бінаризація даних
data_binarized =
preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\n Binarized data:\n", data_binarized)
```

Виконавши код, отримаємо результат

```
Binarized data:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

Як бачите, всі значення понад 2,1 примусово встановлюються рівними 1. Інші значення стають рівними 0.

1.1.2. Виключення середнього

Виключення середнього - методика попередньої обробки даних, що зазвичай використовується в машинному навчанні. Як правило, із векторів ознак (feature vectors) доцільно виключати середні значення, щоб кожна ознака (feature) центрувалася на нулі. Це робиться з метою, виключити з розгляду зміщення значень у векторах ознак.

Додамо наступні рядки до того ж файлу Python, який використовувався в попередньому завданні 2.1.1.

```
# Виведення середнього значення та стандартного відхилення

print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
```

Ці рядки коду відображають середнє значення і середньоквадратичне відхилення для вхідних даних.

Тепер виключимо середнє значення

```
# Исключение среднего
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
```

```
print("Std deviation =", data_scaled.std(axis=0))
```

Після виконання коду отримаємо:

```
BEFORE:
Mean = [ 3.775 -1.15  -1.3  ]
Std deviation = [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean = [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [1. 1. 1.]
```

Неважко помітити, що середнє значення практично рівне 0, а стандартне відхилення - 1.

1.1.3. Масштабування

У нашому векторі ознак кожне значення може змінюватись у деяких випадкових межах. Тому дуже важливо масштабувати ознаки, щоб вони були рівним ігровим полем для тренування алгоритму машинного навчання. Ми не хочемо, щоб будь-яка з ознак могла набувати штучно великого або малого значення лише через природу вимірів.

Додамо до того ж файлу Python наступні рядки.

```
# Масштабування MinMax
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0,
1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)
```

Після виконання коду отримаємо такий результат:

```
Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]
```

Кожен рядок відмасштабований таким чином, щоб максимальним значенням була б 1, а всі решта значень визначалися відносно неї.

1.1.4. Нормалізація

Процес нормалізації полягає у зміні значень у векторі ознак таким чином, щоб для їх вимірювання можна було використовувати одну загальну шкалу. У машинному навчанні використовують різні форми нормалізації. У найбільш поширених з них, значення змінюються так, щоб їх сума дорівнювала 1. **L1-нормалізація** використовує метод найменших абсолютних відхилень (Least

Absolute Deviations), що забезпечує рівність 1 суми абсолютних значень в кожному ряду. **L2-нормалізація** використовує метод найменших квадратів, що забезпечує рівність 1 суми квадратів значень. Взагалі, техніка *L1-нормалізації* вважається більш надійною по порівняно з *L2-нормалізацією*, оскільки вона менш чутлива до викидів [1, 3].

Дуже часто дані містять викиди, і з цим нічого не вдієш. Ми хочемо використовувати безпечні методики, що дозволяють ігнорувати викиди у процесі обчислень. Якби ми вирішували завдання, в якому викиди грають важливу роль, то, ймовірно, найкращим вибором була б *L2-нормалізація*.

Додамо наступні рядки в той же файл Python.

```
# Нормалізація даних
data_normalized_l1 = preprocessing.normalize(input_data,
norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data,
norm='l2')
print("\nl1 normalized data:\n", data_normalized_l1)
print("\nl2 normalized data:\n", data_normalized_l2)
```

Виконавши код отримаємо наступні результати:

```
l1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

l2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

Копії екрану з кодом програми та результатами занесіть у звіт (рис.1 звіту). Зробіть висновок чим відрізняються L1-нормалізація від L2-нормалізації

1.1.5. Кодування міток

Як правило, в процесі класифікації даних ми маємо справу з множиною міток (labels). Ними можуть бути слова, числа або інші об'єкти. Функції машинного навчання, що входять до бібліотеки sklearn, очікують, що мітки є числами. Тому, якщо мітки – це вже числа, ми можемо використовувати їх безпосередньо для того, щоб почати тренування. Однак, зазвичай, це не так. На практиці мітками служать слова, оскільки в такому вигляді вони краще всього сприймаються людиною. Ми позначаємо тренувальні дані словами, щоб

полегшити відстеження відповідностей. Для перетворення слів у числа необхідно використовувати *кодування*. Під *кодуванням міток* (label encoding) мається на увазі процес перетворення словесних міток на числову форму. Завдяки цьому алгоритми можуть оперувати нашими даними.

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
frorn sklearn import preprocessing
```

Визначимо мітки.

```
# Надання позначок вхідних даних
Input_labels = ['red', 'Ыack', 'red', 'green', 'Ьlack', 'yellow',
'white']
```

Створимо об'єкт кодування міток та навчимо його.

```
# Створення кодувальника та встановлення відповідності
# між мітками та числами
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
```

Виведемо відображення слів на числа.

```
# Виведення відображення
print("\nLabel mapping:")
for i, item in enumerate(encoder.classes ) :
print(item, '-->', i)
```

Перетворимо набір випадково впорядкованих міток, щоб перевірити роботу кодувальника.

```
# перетворення міток за допомогою кодувальника
test_labels = ['green', 'red', 'Ыack']
encoded_values = encoder.transform(test_labels )
print("\nLabels =", test labels )
print("Encoded values =", list (encoded_values ) )
```

Декодуємо випадковий набір чисел.

```
# Декодування набору чисел за допомогою декодера
encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list (decoded_list ) )
```

Після виконання цього коду у вікні терміналу повинна відобразитися інформація. Проаналізуйте цю інформацію.

Копії екрану з кодом програми та результатами занесіть у звіт (рис. 2 звіту)

Програмний код збережіть під назвою LR_1_task_1.py

Завдання 1.2. Попередня обробка нових даних

У коді програми попереднього завдання поміняйте дані по рядках (значення змінної input_data) на значення відповідно варіанту таблиці 1 та виконайте операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації.

Варіант обирається відповідно номера за списком групи відповідно до таблиці 1.

Копії екрану з кодом програми та результатами занесіть у звіт (рис. 3 звіту)

Програмний код збережіть під назвою LR_1_task_2.py

Завдання 1.3. Класифікація логістичною регресією або логістичний класифікатор

Логістична регресія (logistic regression) - це методика, що використовується для пояснення відносин між вхідними та вихідними змінними. Вхідні змінні вважаються незалежними, вихідні – залежними [2]. Залежна змінна може мати лише фіксований набір значень. Ці значення відповідають класам завдання класифікації. Метою є ідентифікація відносин між незалежними та залежними змінними за допомогою оцінки ймовірностей того, що та або інша залежна змінна відноситься до того чи іншого класу. За своєю природою логістична функція є сигмоїдою, що використовується для створення функцій з різними параметрами. Вона тісно пов’язана з аналізом даних на основі узагальненої лінійної моделі, у відповідності до якої робиться спроба підігнати пряму лінію до групи точок таким чином, щоб мінімізувати помилку. Замість лінійної регресії ми застосовуємо логістичну регресію. В дійсності сама по собі логістична регресія призначена не для класифікації даних, проте вона дозволяє спростити вирішення цього завдання. Зважаючи на її простість, логістичну регресію часто задіюють у машинному навчанні. Розглянемо приклад застосування логістичної регресії до створення класифікатора. Перш ніж продовжити, переконайтеся, що у вашій системі встановлено пакет Tkinter. У разі потреби ви зможете знайти його за адресою https://docs.python.org/2/library/tkinter.html.

Створіть новий файл Python та імпортуйте наведені нижче пакети.

Ми імпортуватимемо одну з функцій, що містяться у файлі utilities.py. Ці функції не є обов’язковими для проведення класифікації, але вони дозволяють наглядно показати, що ж відбувається при класифікації (візуалізувати функції та результати). Нижче ми познайомимось із цією функцією більш детально, а поки що просто імпортуйте її. (перепишіть файл у директорію вашого проекту).

Таблиця 1

№ варіанту	Значення змінної input_data												Поріг бінаризації
1.	4.3	-9.9	-3.5	-2.9	4.1	3.3	-2.2	8.8	-6.1	3.9	1.4	2.2	2.2
2.	4.1	-5.9	-3.5	-1.9	4.6	3.9	-4.2	6.8	6.3	3.9	3.4	1.2	3.2
3.	1.3	-3.9	6.5	-4.9	-2.2	1.3	2.2	6.5	-6.1	-5.4	-1.4	2.2	1.1
4.	-5.3	-8.9	3.0	2.9	5.1	-3.3	3.1	-2.8	-3.2	2.2	-1.4	5.1	3.0
5.	-1.3	3.9	4.5	-5.3	-4.2	-1.3	5.2	-6.5	-1.1	-5.2	2.6	-2.2	3.0
6.	2.3	-1.6	6.1	-2.4	-1.2	4.3	3.2	5.5	-6.1	-4.4	1.4	-1.2	2.1
7.	1.3	3.9	6.2	4.9	2.2	-4.3	-2.2	6.5	4.1	-5.2	-3.4	-5.2	2.0
8.	4.6	9.9	-3.5	-2.9	4.1	3.3	-2.2	8.8	-6.1	3.9	1.4	2.2	2.2
9.	4.1	-5.9	3.3	6.9	4.6	3.9	-4.2	3.8	2.3	3.9	3.4	1.2	3.2
10.	1.3	-3.9	6.5	-4.9	-2.2	1.3	2.2	6.5	-6.1	3.4	-3.4	-2.2	1.2
11.	-5.3	-8.9	3.0	2.9	5.1	-3.3	3.1	-2.8	-3.2	2.2	-1.4	5.1	2.0
12.	-1.3	3.9	4.5	-5.3	-4.2	3.3	-5.2	-6.5	-1.1	-5.2	2.6	-2.2	1.8
13.	-2.3	-1.6	-6.1	-2.4	-1.2	4.3	3.2	3.1	6.1	-4.4	1.4	-1.2	2.1
14.	-1.3	3.9	6.2	-4.9	2.2	-4.3	-2.2	6.5	4.1	-5.2	-3.4	-5.2	2.2
15.	-2.3	3.9	-4.5	-5.3	-4.2	-1.3	5.2	-6.5	-1.1	-5.2	2.6	-2.2	3.0
16.	-3.3	-1.6	6.1	2.4	-1.2	4.3	-3.2	5.5	-6.1	-4.4	1.4	-1.2	2.1
17.	1.3	3.9	6.2	4.9	2.2	-4.3	-2.6	6.5	4.1	-5.2	-3.4	-5.2	2.0
18.	4.6	3.9	-3.5	-2.9	4.1	3.3	2.2	8.8	-6.1	3.9	1.4	2.2	2.2
19.	-4.1	-5.5	3.3	6.9	4.6	3.9	-4.2	3.8	2.3	3.9	3.4	-1.2	3.2
20.	6.3	-3.9	6.5	-4.9	-2.2	1.3	2.2	6.5	-6.1	-3.4	5.2	-1.2	1.2
21.	-5.3	-8.9	3.0	2.9	5.1	-3.3	3.1	-2.8	-3.2	4.2	-1.4	6.1	2.0
22.	-1.6	3.9	4.5	-4.3	4.2	3.3	-5.2	-6.5	5.1	-5.2	2.6	-2.2	3.8
23.	2.5	-1.6	-6.1	-2.4	-1.2	4.3	3.2	3.1	6.1	-4.4	1.4	-1.2	2.5
24.	-4.3	3.3	-6.2	4.9	5.2	-5.3	-4.2	6.5	4.4	-3.2	-3.4	6.1	2.2
25.	-3.3	-1.6	6.1	2.4	-1.2	4.3	-3.2	5.5	-5.3	-4.4	1.4	-1.2	2.1
26.	1.3	3.9	6.2	4.9	2.2	-4.3	-2.6	6.5	4.6	-5.2	-3.4	-5.2	2.7
27.	4.6	3.9	-3.5	-2.9	4.1	3.3	2.2	8.8	-4.1	3.9	2.4	4.2	2.2
28.	-4.1	-5.5	3.3	6.9	4.6	3.9	-4.2	3.8	2.3	3.9	3.4	-1.2	3.0
29.	1.6	3.1	6.5	-4.9	-2.2	1.3	2.2	6.5	-6.1	-3.4	5.2	-3.2	3.2
30.	-5.3	-8.9	4.2	2.9	-5.0	-3.3	3.1	-2.8	-3.2	4.2	-1.4	6.1	1.5

```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier
```

Визначимо зразок вхідних даних за допомогою двовимірних векторів і відповідних міток.

```
# Визначення зразка вхідних даних
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5],
              [6, 5], [5.6, 5], [3.3, 0.4],
              [3.9, 0.9], [2.8, 1],
```

```
                [0.5, 3.4], [1, 4], [0.6, 4.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])
```

Ми тренуватимемо класифікатор, використовуючи ці позначені дані.

Створимо об'єкт логістичного класифікатора.

```
# Створення логістичного класифікатора
classifier =
linear_model.LogisticRegression(solver='liblinear',C=1)
```

Навчимо класифікатор, використовуючи певні дані.

```
# Тренування класифікатора
classifier.fit(X, y)
```

Візуалізуємо результати роботи класифікатора, відстеживши межі

```
visualize_classifier(classifier, X, y)
```

Копії екрану з кодом програми та результатами занесіть у звіт (рис. 4 звіту)

Програмний код збережіть під назвою LR_1_task_3.py

Завдання 1.4. Класифікація наївним байєсовським класифікатором

Наївний байєсовський :класифікатор (Naive Bayes classifier) - це простий класифікатор, заснований на використанні теореми Байєса, яка описує ймовірність події з урахуванням пов'язаних з нею умов [7]. Такий класифікатор створюється за допомогою привласнення позначок класів екземплярам завдання. Останні представляються як векторів значень ознак. При цьому передбачається, що значення будь-якої заданої ознаки не залежить від значень інших ознак. Його припущення про незалежністьознак і становить наївну частину байєсовського класифікатора. Ми можемо оцінювати вплив будь-якої ознаки змінної класу незалежно від впливу інших ознак. Наприклад, ми можемо вважати тварину гепардом, якщо воно має плямисту шкіру, чотири лапи та хвіст і розвиває швидкість, рівну приблизно 70 миль на годину. У разі використання наївного байєсівського класифікатора вважається, що кожна з ознак робить незалежний внесок у кінцевий результат, що оцінює ймовірністьте, що тварина є гепардом. Ми не обтяжуватимемо себе розглядом кореляції між малюнком шкіри, кількістю лап, наявністю хвоста та швидкістю переміщення.

Займемося створенням наївного байєсівського класифікатора. Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.model_selection import train_test_split

from utilities import visualize_classifier
```

Як джерело даних ми будемо використовувати файл data_multivar_nb.txt, кожен рядок якого містить значення розділені комою.

```
# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'
```

Завантажимо дані із цього файлу.

```
# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Створимо екземпляр наївного байєсовського класифікатора. У даному випадку ми будемо використовувати гаусівський наївний байєсівський класифікатор, в якому передбачається, що значення, які асоціюються з кожним класом, дотримуються закону розподілу Гауса.

```
# Створення наївного байєсовського класифікатора
classifier = GaussianNB()
```

Навчимо класифікатор, використовуючи тренувальні дані.

```
# Тренування класифікатора
classifier.fit(X, y)
```

Запустимо класифікатор на тренувальних даних та спрогнозуємо результати.

```
# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)
```

Обчислимо якість (accuracy)1 класифікатора, порівнявши передбачені значення з істинними мітками, а потім візуалізуємо результат.

```
# Обчислення якості класифікатора
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2),
"%")

# Візуалізація результатів роботи класифікатора
visualize_classifier(classifier, X, y)
```

Копії екрану з кодом програми та результатами занесіть у звіт (рис. 5 звіту)

Попередній метод обчислення якості класифікатора не є надійним. Нам потрібно виконати перехресну перевірку, щоб не використовувати ті самі тренувальні дані при тестуванні.

Розіб'ємо дані на навчальний та тестовий набори. Відповідно до значення параметра test_size, зазначеного в рядку коду нижче, ми віднесемо 80% даних до тренування, а 20% - до тестування. Потім ми виконаємотренування наївним байєсовським класифікатором на цих даних.

```
# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
train_test_split.train_test_split(X, y, test_size=0.2,
random_state=3)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)
```

Обчислимо якість класифікатора і візуалізуємо результати.

```
# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

# Візуалізація роботи класифікатора
visualize_classifier(classifier_new, X_test, y_test)
```

Скористаємося вбудованими функціями для обчислення якості (accuracy),точності (precision) 2 та повноти (recall) 3 класифікатора на підставіпотрійний перехрестної перевірки.

```
num_folds = 3
accuracy_values = train_test_split.cross_val_score(classifier,
                          X, y, scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) +
"%")

precision_values = train_test_split.cross_val_score(classifier,
               X, y, scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2))
+ "%")

recall_values = train_test_split.cross_val_score(classifier,
           X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) +
"%")

f1_values = train_test_split.cross_val_score(classifier,
                     X, y, scoring='f1_weighted', cv=num_folds)
```

```
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```

Виконавши цей код, ви отримаєте для першого тренувального прогону зображення. На ньому показані межі, отримані за допомогою класифікатора.

Копії екрану з кодом програми та результатами занесіть у звіт (рис. 6 звіту)

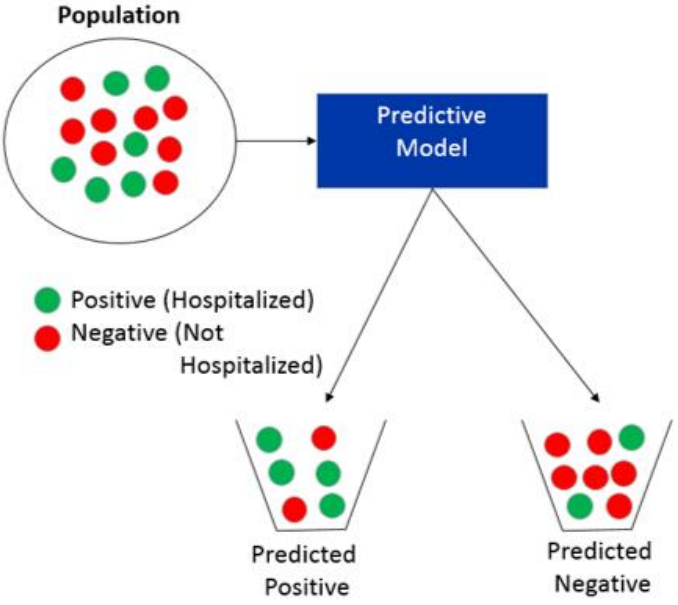
Зробіть ще один прогін та зображення результатів класифікації занесіть у звіт. (рис. 7 звіту)

Порівняйте між собою результати висновок запишіть у звіт

Програмний код збережіть під назвою LR_1_task_4.py

Завдання 2.5. Вивчити метрики якості класифікації

Класифікація полягає у спробі передбачити, з якого класу надходить конкретна вибірка із популяції. Наприклад, якщо ми намагаємося передбачити, чи буде конкретного пацієнта повторно госпіталізовано, то можливі два класи: госпіталізований (позитивний) і не госпіталізований (негативний) [1, 5]. Потім модель класифікації намагається передбачити, чи кожен пацієнт буде госпіталізований чи не госпіталізований. Іншими словами, класифікація просто намагається передбачити, який сегмент (прогнозований позитивний або пргнозований негативний) має бути розміщений конкретною вибіркою із сукупності, як показано нижче.



Коли ви тренуєте свою прогностичну класифікаційну модель, ви захочете оцінити, наскільки вона хороша. Цікаво, що є безліч різних способів оцінки продуктивності. Більшість дослідників даних, які використовують Python для прогнозного моделювання використовують пакет scikit-learn. Scikit-learn

Python. Він містить багато вбудованих функцій для аналізу продуктивності моделей. У цьому завданні ми розглянемо деякі з цих метрик і напишемо власні функції з нуля, щоб зрозуміти математику, що лежить в основі деяких з них.

Запрограмуємо наступні показники зі sklearn.metrics:

confusion_matrix – матриця помилок (або матриця неточностейчи плутанини);

accuracy_score – акуратність (з англ. може перекласти як точність, але не плутайте бо то інший показник)

recall_score – повнота

precision_score – точність

f1_score – F-міра

roc_curve – ROC-крива, крива робочих характеристик (англ. Receiver Operating Characteristics curve).

roc_auc_score – вимір площі під ROC-кривою (англ. Area Under the Curve - AUC). (ROC-AUC)

Ми напишемо наші власні функції з нуля, припускаючи двокласову класифікацію.

Зверніть увагу, що вам потрібно буде заповнити частини, позначені як # your code here!

Завантажте зразок набору даних, який має фактичні мітки (actual_label) та ймовірності прогнозування для двох моделей (model_RF та model_LR). Тут ймовірність - це можливість бути 1-м класом.

```
import pandas as pd
df = pd.read_csv('data_metrics.csv')
df.head()
```

Вони матимуть такий вигляд

	actual_label	model_RF	model_LR
0	1	0.639816	0.531904
1	0	0.490993	0.414496
2	1	0.623815	0.569883
3	1	0.506616	0.443674
4	0	0.418302	0.369532

У більшості проектів ви визначатимете граничне значення, щоб визначити, які ймовірності прогнозування позначені як прогнозований позитивний або передбачений негативний результат. А поки що давайте припустимо, що поріг

дорівнює 0,5. Давайте додамо два додаткові стовпці, які перетворять ймовірності на передбачені мітки.

```
thresh = 0.5
df['predicted_RF'] = (df.model_RF >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
df.head()
```

	actual_label	model_RF	model_LR	predicted_RF	predicted_LR
0	1	0.639816	0.531904	1	1
1	0	0.490993	0.414496	0	0
2	1	0.623815	0.569883	1	1
3	1	0.506616	0.443674	1	0
4	0	0.418302	0.369532	0	0

Матриця помилок (confusion_matrix)

Матриця неточностей (confusion matrix) – це таблиця, що використовується для опису ефективності класифікатора. Зазвичай вона витягується з тестового набору даних, для котрого відомі базові справжні значення.

Ми аналізуємо результати віднесення до кожного класу та визначаємо частку невірно віднесених класів. У процесі конструювання вищезгаданої таблиці ми маємо справу з кількома ключовими метриками, що грають дуже важливу роль у машинному навчанні.

Для нашої здачі, враховуючи фактичну мітку та прогнозовану мітку, перше, що ми можемо зробити, це розділити наші вибірки на 4 сегменти:

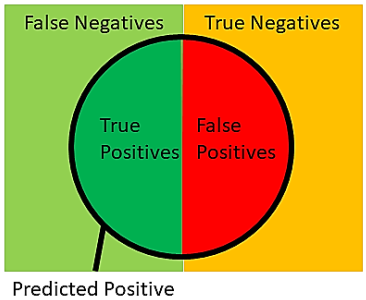
Істинно позитивний – фактичний = 1, прогнозований = 1

Хибний позитивний – фактичний = 0, прогнозований = 1

Невірно негативний - фактичний = 1, прогнозований = 0

Істинно негативний - фактичний = 0, прогнозований = 0

Ці сегменти можуть бути представлені наступним зображенням (джерело https://en.wikipedia.org/wiki/Precision_and_recall#/media/File:Precisionrecall.svg) і ми будемо посилатися на це зображення в багатьох розрахунках нижче.



Ці сегменти можна подати і у вигляді матриці чи таблиці

Confusion Matrix		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Ми можемо отримати матрицю помилок (у вигляді масиву 2x2) з scikit-learn, яка приймає як вхідні дані фактичні мітки та передбачені мітки

```
from sklearn.metrics import
confusion_matrixconfusion_matrix(df.actual_label.values,
df.predicted_RF.values)
```

```
array([[5519, 2360],
       [2832, 5047]], dtype=int64)
```

де було 5047 істинних позитивних результатів, 2360 помилкових позитивних результатів, 2832 помилкових негативних та 5519 істинних негативних.

Визначте ваші власні функції для перевірки confusion_matrix. Зверніть увагу, що тут заповнено перший елемент, а вам потрібно заповнити решту 3.

УВАГА! При написанні ваших власних функцій в коді ТУТ І ДАЛІ замість my в ваших функціях my_confusion_matrix повинно стояти ваше прізвище англ. мовою! Наприклад:

def ivanov_confusion_matrix(y_true, y_pred):

```
def find_TP(y_true, y_pred):
    # counts the number of true positives (y_true = 1, y_pred = 1)
    return sum((y_true == 1) & (y_pred == 1))
def find_FN(y_true, y_pred):
    # counts the number of false negatives (y_true = 1, y_pred =
0)
    return # your code here
def find_FP(y_true, y_pred):
    # counts the number of false positives (y_true = 0, y_pred =
1)
    return # your code here
def find_TN(y_true, y_pred):
    # counts the number of true negatives (y_true = 0, y_pred = 0)
    return # your code here
```

Перевірте свої результати на відповідність

```
print('TP:',find_TP(df.actual_label.values,
df.predicted_RF.values))
print('FN:',find_FN(df.actual_label.values,
df.predicted_RF.values))
print('FP:',find_FP(df.actual_label.values,
df.predicted_RF.values))
print('TN:',find_TN(df.actual_label.values,
df.predicted_RF.values))
```

Давайте напишемо функцію, яка обчислить всі чотири сегменти для нас, та іншу функцію для дублювання confusion_matrix.

```
import numpy as np
def find_conf_matrix_values(y_true,y_pred):
    # calculate TP, FN, FP, TN
    TP = find_TP(y_true,y_pred)
    FN = find_FN(y_true,y_pred)
    FP = find_FP(y_true,y_pred)
    TN = find_TN(y_true,y_pred)
    return TP,FN,FP,TN
def my_confusion_matrix(y_true, y_pred):
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return np.array([[TN,FP],[FN,TP]])
```

Перевірте відповідність результатів з

```
my_confusion_matrix(df.actual_label.values,
df.predicted_RF.values)
```

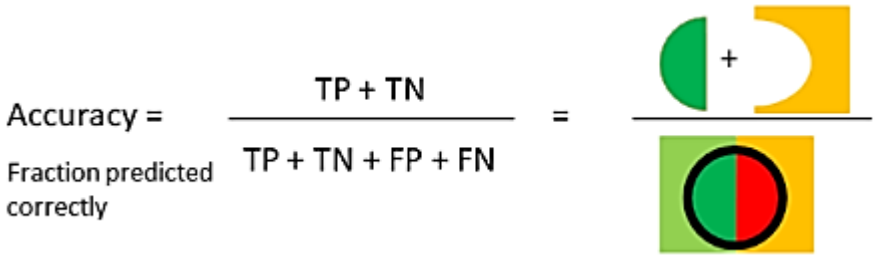
Замість того, щоб порівнювати вручну, давайте перевіримо, що наші функції працюють, використовуючи вбудовані Pythonassert і Numpy'sarray_equal функції

```
assert  np.array_equal(my_confusion_matrix(df.actual_label.values,
df.predicted_RF.values), confusion_matrix(df.actual_label.values,
df.predicted_RF.values) ), 'my_confusion_matrix() is not correct
for RF'assert
np.array_equal(my_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),confusion_matrix(df.actual_label.values,
df.predicted_LR.values) ), 'my_confusion_matrix() is not correct
for LR'
```

Зважаючи на ці чотири сегменти (TP, FP, FN, TN), ми можемо обчислити багато інших показників продуктивності.

accuracy_score

Найбільш поширеним показником для класифікації є акуратність, тобто частка вибірок, що правильно спрогнозовані, як показано нижче:



Ми можемо отримати оцінку точності з scikit-learn, яка приймає як вхідні дані фактичні мітки та прогнозовані мітки.

```
from sklearn.metrics import
accuracy_scoreaccuracy_score(df.actual_label.values,
df.predicted_RF.values)
```

Ваша відповідь повинна бути 0.6705165630156111

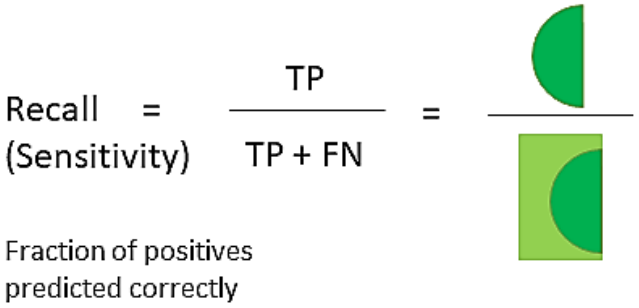
Визначте свою власну функцію, яка дублює accuracy_score, використовуючи формулу вище.

```
def my_accuracy_score(y_true, y_pred):
    # calculates the fraction of samples
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return # your code hereassert
my_accuracy_score(df.actual_label.values, df.predicted_RF.values)
== accuracy_score(df.actual_label.values, df.predicted_RF.values),
'my_accuracy_score failed on
assert my_accuracy_score(df.actual_label.values,
df.predicted_LR.values) == accuracy_score(df.actual_label.values,
df.predicted_LR.values), 'my_accuracy_score failed on LR'
print('Accuracy RF:
%.3f'%(my_accuracy_score(df.actual_label.values,
df.predicted_RF.values))))
print('Accuracy LR:
%.3f'%(my_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))
```

Використовуючи акуратність як показник продуктивності, можна зробити висновок, що модель RF є більш точною (0,67), ніж модель LR (0,62). Так що ми маємо зупинитися тут і сказати, що модель RF – найкраща модель? Ні! Акуратність не завжди є найкращою метрикою для оцінки моделей класифікації. Наприклад, припустимо, що ми намагаємося передбачити те, що

відбувається лише 1 раз із 100 разів. Ми могли б побудувати модель, яка матиме точність 99%, сказавши, що подія ніколи не відбувалася. Тим не менш, ми ловимо 0% подій, які нас цікавлять. Показник 0% тут є ще одним показником продуктивності, відомий як **recall_score**:

recall_score (також відомий як чутливість) є частка позитивних подій, які ви правильно передбачили, як показано нижче:



Ми можемо отримати оцінку точності з scikit-learn, яка приймає як вхідні дані фактичні мітки та прогнозовані мітки.

```
from sklearn.metrics import
recall_scorerecall_score(df.actual_label.values,
df.predicted_RF.values)
```

Визначте власну функцію, яка дублює recall_score, використовуючи формулу вище.

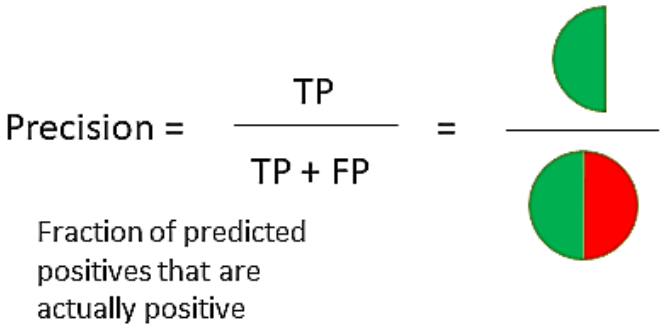
```
def my_recall_score(y_true, y_pred):
    # calculates the fraction of positive samples predicted
correctly
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return # your code hereassert
my_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values, df.predicted_RF.values),
'my_accuracy_score failed on RF'
assert my_recall_score(df.actual_label.values,
df.predicted_LR.values) == recall_score(df.actual_label.values,
df.predicted_LR.values), 'my_accuracy_score failed on LR'
print('Recall RF: %.3f'%(my_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f'%(my_recall_score(df.actual_label.values,
df.predicted_LR.values)))
```

Один із способів підвищити повноту – збільшити кількість вибірок, які ви визначаєте як прогнозовані позитивні, шляхом зниження порога для прогнозованих позитивних результатів. На жаль, це також збільшить кількість

хибних спрацьовувань. Інший показник продуктивності, названий точністю, враховує це.

precision_score

Точність - це частка очікуваних позитивних подій, які є позитивними, як показано нижче:



Ми можемо отримати оцінку точності з scikit-learn, яка приймає як вхідні дані фактичні мітки та прогнозовані мітки.

```
from sklearn.metrics import
precision_scoreprecision_score(df.actual_label.values,
df.predicted_RF.values)
```

Визначте власну функцію, яка дублює precision_score, використовуючи формулу вище.

```
def my_precision_score(y_true, y_pred):
    # calculates the fraction of predicted positives samples that
are actually positive
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return # your code hereassert
my_precision_score(df.actual_label.values, df.predicted_RF.values)
== precision_score(df.actual_label.values,
df.predicted_RF.values), 'my_accuracy_score failed on RF'
assert my_precision_score(df.actual_label.values,
df.predicted_LR.values) == precision_score(df.actual_label.values,
df.predicted_LR.values), 'my_accuracy_score failed on LR'
print('Precision RF:
%.3f'%(my_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR:
%.3f'%(my_precision_score(df.actual_label.values,
df.predicted_LR.values)))
```

В цьому випадку, схоже, що RF модель краще як за повнотою, так і по точності. Але що б ви зробили, якби одна модель краща за повнотою, а інша

була точнішою. Один метод, який використовують у цьому випадку, називають рахунком F1.

f1_score. Оцінка f1 є гармонійним середнім значенням повноти та точності, з більш високою оцінкою як краща модель. Оцінка f1 розраховується за такою формулою:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 * (precision * recall)}{precision + recall}$$

Ми можемо отримати оцінку f1 з scikit-learn, яка приймає як вхідні дані фактичні мітки та прогнозовані мітки

```
from sklearn.metrics import
f1_scoref1_score(df.actual_label.values, df.predicted_RF.values)
```

Визначте власну функцію, яка дублює f1_score, використовуючи формулу вище.

```
def my_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = my_recall_score(y_true,y_pred)
    precision = my_precision_score(y_true,y_pred)
    return # your code hereassert
my_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values, df.predicted_RF.values),
'my_accuracy_score failed on RF'
assert my_f1_score(df.actual_label.values, df.predicted_LR.values)
== f1_score(df.actual_label.values, df.predicted_LR.values),
'my_accuracy_score failed on LR'
print('F1 RF: %.3f'%(my_f1_score(df.actual_label.values,
df.predicted_RF.values))))
print('F1 LR: %.3f'%(my_f1_score(df.actual_label.values,
df.predicted_LR.values))))
```

До цих пір ми припускали, що ми визначили поріг 0,5 для вибору зразків, які прогнозуються як позитивні. Якщо ми змінимо цей поріг, показники продуктивності зміняться. Як показано нижче:

```
print('scores with threshold = 0.5')
print('Accuracy RF:
%.3f'%(my_accuracy_score(df.actual_label.values,
df.predicted_RF.values))))
print('Recall RF: %.3f'%(my_recall_score(df.actual_label.values,
df.predicted_RF.values))))
```

```
print('Precision RF:
%.3f'%(my_precision_score(df.actual_label.values,
df.predicted_RF.values))))
print('F1 RF: %.3f'%(my_f1_score(df.actual_label.values,
df.predicted_RF.values))))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF:
%.3f'%(my_accuracy_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values))))
print('Recall RF: %.3f'%(my_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values))))
print('Precision RF:
%.3f'%(my_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values))))
print('F1 RF: %.3f'%(my_f1_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values))))
```

Порівняйте результати для різних порогів та зробіть висновки.

Якже оцінювати модель, якщо ми не вибрали поріг? Одним із найпоширеніших методів є використання кривої робочих характеристик приймача (ROC).

roc_curve та roc_auc_score

Криві ROC дуже допомагають зрозуміти баланс між істинно позитивними показниками та хибнопозитивними показниками. Scikit_learn має вбудовані функції для кривих ROC та їх аналізу. Входи в ці функції (roc_curvea також roc_auc_score) фактичні мітки та прогнозовані ймовірності (не прогнозовані мітки). І та й інша roc_curve, а також roc_auc_score обидві складні функції, тому ми не будемо писати ці функції з нуля. Натомість ми покажемо, як використовувати функції Scikit_learn і пояснимо ключові моменти. Давайте почнемо з використання roc_curve.

```
from sklearn.metrics import roc_curvefpr_RF, tpr_RF, thresholds_RF
= roc_curve(df.actual_label.values, df.model_RF.values)
fpr_LR, tpr_LR, thresholds_LR = roc_curve(df.actual_label.values,
df.model_LR.values)
```

Функція roc_curve повертає три списки:

thresholds= всі унікальні ймовірності передбачення в порядку спадання

fpr = хибнопозитивний показник (FP/(FP+TN)) для кожного порогу

tpr = істинно позитивна швидкість (TP/(TP+FN)) для кожного порогу.

```
thresholds_RF
```

```
array([0.93052053, 0.82363091, 0.82354671, ..., 0.25654616, 0.25587275,
       0.17142947])
```

```
fpr_RF
```

```
array([0.       , 0.       , 0.       , ..., 0.9941617, 0.9941617,
       1.       ])
```

```
tpr_RF
```

```
array([1.26919660e-04, 5.33062571e-03, 5.58446503e-03, ...,
       9.99873080e-01, 1.00000000e+00, 1.00000000e+00])
```

Побудуйте криву ROC для кожної моделі, як показано нижче.

```
importmatplotlib.pyplotaspltplt.plot(fpr_RF, tpr_RF,'r-',label =
'RF')
plt.plot(fpr_LR,tpr_LR,'b-', label= 'LR')
plt.plot([0,1],[0,1],'k-',label='random')
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')
plt.legend()
plt.xlabel('FalsePositiveRate')
plt.ylabel('TruePositiveRate')
plt.show()
```

Рисунок занесіть у звіт

Є кілька речей, які ми можемо спостерігати із цієї фігури:

- модель, яка випадково вгадує мітку, призведе до чорної лінії, і нам необхідно мати модель із кривою над цією чорною лінією.
- ROC- крива, яка знаходиться далі від чорної лінії, краще, тому RF (червона) виглядає краще, ніж LR (синя)
- Хоча це не видно безпосередньо, високий поріг призводить до точки у лівому нижньому кутку, а низький поріг призводить до точки у верхньому правому куті. Це означає, що, зменшуючи поріг, ви отримуєте вищий TPR за рахунок вищого FPR.

Для аналізу продуктивності ми використовуватимемо метрику площі під кривою.

```
from sklearn.metrics import roc_auc_scoreauc_RF =
roc_auc_score(df.actual_label.values, df.model_RF.values)
auc_LR = roc_auc_score(df.actual_label.values,
df.model_LR.values)print('AUC RF:%.3f'% auc_RF)
print('AUC LR:%.3f'% auc_LR)
```

Як ви зможете побачити, площа під кривою для моделі RF (AUC = 0,738) краще, ніж LR (AUC = 0,666). Коли будуєте криву ROC, доцільно додавати AUC до легенди, як показано нижче.

```
import matplotlib.pyplot as plt
plt.plot(fpr_RF, tpr_RF,'r-',label = 'RF AUC: %.3f'%auc_RF)
plt.plot(fpr_LR,tpr_LR,'b-', label= 'LR AUC: %.3f'%auc_LR)
plt.plot([0,1],[0,1],'k-',label='random')
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

Рисунок занесіть у звіт. Зробіть висновки яка з двох моделей (RF та LR***) краща і чому. Висновки занесіть у звіт.***

У прогнозуючій аналітиці при виборі між двома чи декількома моделями важливо вибрати одну метрику продуктивності. Ви можете вибирати з множини (акуратність, повнота, точність, f1-оцінка, AUC тощо). Зрештою, ви повинні використовувати метрику продуктивності, яка найбільше підходить для аналізованого бізнес-завдання. Багато дослідників даних вважають за краще використовувати AUC для аналізу продуктивності кожної моделі, оскільки він не вимагає вибору порога і допомагає збалансувати істинний позитивний показник і хибнопозитивний показник.

Програмний код збережіть під назвою LR_1_task_5.py

Завдання 2.6. Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVM). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.

Код програми та результати занесіть у звіт.
Програмний код збережіть під назвою LR_1_task_6.py

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 1

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КІМ-ЛР-1-NNN-XXXXX.doc

де NNN – номер групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КІМ-ЛР-1-NNN-XXXXX.pdf
Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Яка основна мета машинного навчання?
2. Які основні складові необхідні для машинного навчання?
3. Дати класифікацію методів класичного навчання.
4. Що таке класифікація даних?
5. Які показники якості класифікації ви знаєте?
6. Що означає ROC-крива?
7. Що описує оцінка f1_score?
8. Що описує матриця помилок (confusion_matrix)?
9. Що описує показник accuracy_score?
10. Що описує показник recall_score(повнота)?
11. Що описує показник precision_score(точність)?
12. Що описує показник roc_auc_score?

ЛАБОРАТОРНА РОБОТА № 2
ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

Мета роботи: *використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні методи класифікації даних та навчитися їх порівнювати.*

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії попередньої обробки та матеріал поданий у цьому підрозділі. Також доцільно вивчити матеріал поданий в літературі:

Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). Mathematics for machine learning. Cambridge University Press. Available: https://mml-book.github.io/book/mml-book.pdf.

При виконанні роботи можна використовувати Google Colab або Jupiter Notebook.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 2 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 2.1. Класифікація за допомогою машин опорних векторів (SVM)

Створіть класифікатор у вигляді машини опорних векторів, призначений для прогнозування меж доходу заданої фізичної особи на основі 14 ознак (атрибутів). Метою є з'ясування умов, за яких щорічний прибуток людини перевищує $50000 або менше цієї величини за допомогою бінарної класифікації. Набір даних знаходяться за посиланням

https://archive.ics.uci.edu/ml/datasets/census+income

Слід зазначити одну особливість цього набору, яка полягає в тому, що кожна точка даних є поєднанням тексту і чисел. Ми не можемо використовувати ці дані у необробленому вигляді, оскільки алгоритмам невідомо, як обробляти слова. ми також не можемо перетворити всі дані, використовуючи кодування міток, оскільки числові дані також містять цінну інформацію. Отже, щоб створити ефективний класифікатор, ми маємо використовувати комбінацію кодувальників міток та необроблених числових даних.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Ознайомтесь з набором даних.

Випишіть у звіт всі 14 ознак з набору даних – їх назви та що вони позначають та вид (числові чи категоріальні).

Створіть новий файл Python та імпортуйте туди пакети

УВАГА! В коді є помилки які ви повинні виправити!

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split
```

Для завантаження даних використовуємо файл income_data.txt, що містить докладну інформацію про доходи населення.

```
# Вхідний файл, який містить дані
input_file = 'income data.txt'
```

Завантажені з файлу дані потрібно підготувати до класифікації, піддавши їх попередньої обробки. Для кожного класу ми будемо використовувати трохи більше 25000 точок даних.

```
# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000
```

Відкриємо файл і прочитаємо рядки.

```
with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >=
max_datapoints:
            break

        if '?' in line:
            continue
```

Кожен рядок даних відокремлюється від наступного за допомогою коми, що потребує відповідного розбиття рядків. Останнім елементом кожного рядка є позначка. Залежно від цієї мітки ми відноситимемо дані до того чи іншого класу.

```
data = line[:-1].split(', ')

if data[-1] == '<=50K' and count_class1 < max_datapoints:
    X.append(data)
    count_class1 += 1

if data[-1] == '>50K' and count_class2 < max_datapoints:
    X.append(data)
    count_class2 += 1
```

Перетворимо список на масив array, щоб його можна було використовувати як вхідні дані функції sklearn.

```
# Перетворення на масив numpy
X = np.array(X)
```

Якщо атрибут - рядок, то він потребує кодування. Якщо атрибут - число, ми можемо залишити його у тому вигляді, як він є. Зауважте, що зрештою ми отримаємо кілька кодувальників міток, які нам потрібно буде відстежувати.

```
# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(X.shape)
for i,item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
```

Створіть SVM-класифікатор із лінійним ядром.

```
# Створення SVM-класифікатора
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
```

Навчіть класифікатор.

```
# Навчання класифікатора
classifier.fit(X, Y)
```

Виконайте перехресну перевірку, розбивши дані на навчальний та тестовий набори у пропорції 80/20, а потім спрогнозуйте результат для тренувальних даних.

```
X_train, X_test, y_train, y_test =
cross_validation.train_test_split(X, y, test_size=0.2,
random_state=5)
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)
```

Обчисліть для класифікатора F-міру.

```
# Обчислення F-міри для SVM-класифікатора
f1 = train_test_split.cross_val_score(classifier, X, y,
scoring='f1_weighted', cv=3)
print("F1 score: " + str(round(100*f1.mean(), 2)) + "%")
```

Тепер, маючи підготовлений класифікатор, подивимося, що станеться, якщо ми виберемо деяку випадкову точку даних і передбачимо результат. Визначимо одну таку точку.

```
# Передбачення результату для тестової точки даних
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-
married', 'Handlers-cleaners', 'Not-in-family', 'White', 'Male',
'0', '0', '40', 'United-States']
```

Перш ніж приступити до прогнозування, ми повинні присвоїти цій точці даних код, використовуючи створений раніше кодувальник ознак.

```
# Кодування тестової точки даних
input_data_encoded = [-1] * len(input_data)
count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(input_data[i])
    else:
        input_data_encoded[i] =
int(label_encoder[count].transform(input_data[i]))
        count += 1

input_data_encoded = np.array(input_data_encoded)
```

Спрогнозуйте результат за допомогою класифікатора.

```
# Використання класифікатора для кодованої точки даних
# та виведення результату
predicted_class = classifier.predict(input_data_encoded)
print(label_encoder[-1].inverse_transform(predicted_class)[0])
```

Тренування класифікатора за допомогою цього коду займе кілька секунд. Після виконання коду ви отримаєте F-міру та результат класифікації

Обчисліть значення інших показників якості класифікації (акуратність, повнота, точність) та разом з F1 занесіть їх у звіт. (Див. ЛР-1).

Збережіть код робочої програми під назвою LR_2_task_1.py

Код програми занесіть у звіт.

Зробіть висновок до якого класу належить тестова точка

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

У попередньому завданні ми побачили, як простий алгоритм SVM LinearSVC може бути використаний для знаходження межі рішення для лінійних даних [1,6]. Однак у разі нелінійно розділених даних, пряма лінія не може бути використана як межа прийняття рішення. Натомість використовується модифікована версія SVM, що зветься Kernel SVM.

В основному, ядро SVM проектує дані нижніх вимірювань, що нелінійно розділяються, на такі, що лінійно розділяються більш високих вимірювань таким чином, що точки даних, що належать до різних класів, розподіляються за різними вимірами. В цьому є закладена складна математика, але вам не потрібно турбуватися про це, щоб використовувати SVM. Ми можемо просто використовувати бібліотеку Scikit-Learn Python для реалізації та використання SVM ядра.

Реалізація SVM ядра за допомогою Scikit-Learn аналогічна до простого SVM.

Використовуючи набір даних та код з попереднього завдання створіть та дослідіть нелінійні класифікатори SVM.

з поліноміальним ядром;

з гаусовим ядром;

з сигмоїдальним ядром.

Для кожного виду класифікатора отримайте та запишіть у звіт показники якості алгоритму класифікації.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Для навчання ядра SVM ми використовуємо той же клас SVC бібліотеки Scikit-Learn svm . Різниця полягає у значенні параметра ядра класу SVC. У разі простого SVM ми використовували "лінійний" (LinearSVC) як значення параметра ядра (kernelSVM). Однак для ядра SVM ви можете використовувати

гаусове, поліноміальне, сигмоїдне або сумісне ядро. Реалізуйте поліноміальні, гаусівські та сигмоїдні ядра, щоб побачити, яке з них найкраще підходить для нашого завдання.

1. Поліноміальне ядро. У разі поліноміального ядра ви також повинні передати значення для параметра degree класу SVC. Це переважно ступінь многочлена. Фактично у попередньому коді вам необхідно замінити лінійний параметр на: KernelSVC(kernel='poly', degree=8):

Не забудьте імпортувати відповідну функцію з бібліотеки.

Вся решта коду повинна працювати.

2. Гаусове ядро. Ми можемо використовувати гаусове ядро для реалізації kernel SVM: KernelSVC(kernel='rbf')

Щоб використовувати ядро Гауса, ви повинні вказати 'rbf' як значення параметра ядра класу SVC.

3. Сигмоїдальне ядро. Щоб використовувати сигмоїдальне ядро, ви повинні вказати 'sigmoid' як значення для параметра kernel класу SVC .

KernelSVC(kernel='sigmoid')

Збережіть код робочих програм під назвами: LR_2_task_2_1.py; LR_2_task_2_2.py, LR_2_task_2_3.py.

Коди та результати занесіть у звіт.

У висновках опишіть який з видів SVM найкраще виконує завдання класифікації за результатами тренування.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Необхідно класифікувати сорти ірисів за деякими їх характеристиками: довжина та ширина пелюсток, а також довжина та ширина чашолистків (див. рис. 1).

Рис. 1. Структура квітки та види ірису

Також, в наявності є вимірювання цих же характеристик ірисів, які раніше дозволили досвідченому експерту віднести їх до сортів: setosa, versicolor і virginica.

Використовувати класичний набір даних у машинному навчанні та статистиці - Iris. Він включений у модуль datasets бібліотеки scikit-learn.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

КРОК 1. ЗАВАНТАЖЕННЯ ТА ВИВЧЕННЯ ДАНИХ

Наша мета полягає в побудові моделі машинного навчання, яка зможе навчитися на основі характеристик ірисів, вже класифікованих за сортами, а потім передбачить сорт для нової квітки ірису.

Оскільки у нас є приклади, за якими ми вже знаємо правильні сорти ірису, завдання, що вирішується, є завданням навчання з учителем. У цьому завданні нам потрібно прогнозувати один із сортів ірису. Це приклад завдання класифікації (classification). Можливі відповіді (різні сорти ірису) називають класами (classes). Кожен ірис в наборі даних належить до одного з трьох класів, таким чином завдання, що вирішується, є завданням трикласової класифікації.

Відповіддю для окремої точки даних (ірису) є той чи інший сорт цієї квітки. Сорт, до якого належить квітка (конкретна точка даних), називається міткою (label).

Завантажте дані, викликавши функцію load_iris:

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

Об'єкт iris, що повертається load_iris, є об'єктом Bunch, який дуже схожий на словник. Він містить ключі та значення:

```
print("Ключі iris_dataset: \n{}".format(iris_dataset.keys()))
```

Результат:

```
Ключі iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR',
'feature_names', 'filename', 'data_module'])
```

УВАЖНО РОЗБЕРІТЬСЯ З ОЗНАКАМИ ТА ДАНИМИ!

Значення ключа DESCR – це короткий опис набору даних [9]. Тут ми покажемо початок опису (частину опису, що залишилася, необхідно вивести та ознайомитись самостійно):

```
print(iris_dataset['DESCR'][:193] + "\n...")
```

Значення ключа target_names – це масив рядків, що містить сорти квітів, які ми хочемо передбачити:

```
print("Назвивідповідей: {}".format(iris_dataset['target_names']))
```

Значення feature_names – це список рядків із описом кожної ознаки:

```
print("Назваознак: \n{}".format(iris_dataset['feature_names']))
```

Самі дані записані в масивах target та data. data – масив NumPy, який містить кількісні вимірювання довжини чашолистків, ширини чашолистків, довжини пелюсток та ширини пелюсток:

```
print("Тип масиву data: {}".format(type(iris_dataset['data'])))
```

Рядки в масиві data відповідають квіткам ірису, а стовпці є чотири ознаки, які були виміряні для кожної квітки:

```
print("Формамасиву data: {}".format(iris_dataset['data'].shape))
```

Визначте, що масив містить виміри для 150 різних квітів за 4 ознаками. Пригадаємо, що у машинному навчанні окремі елементи називаються прикладами (samples), які властивості – характеристиками чи ознаками (feature). Форма (shape) масиву даних визначається кількістю прикладів, помноженим на кількість ознак. Це загальноприйнята угода в scikit-learn, і ваші дані завжди будуть представлені в цій формі.

Виведіть значення ознак для перших п'яти прикладів

Поглянувши на ці дані, ви побачите, що всі п'ять квіток мають ширину пелюстки 0.2 см і перша квітка має найбільшу довжину чашолистки, 5.1 см.

Масив target містить сорти вже виміряних квіток, також записані у вигляді масиву NumPy:

```
print("Типмасиву target: {}".format(type(iris_dataset['target'])))
```

target є одномірним масивом, по одному елементу для кожної квітки:

Сорти кодуються як цілі числа від 0 до 2:

```
print("Відповіді:\n{}".format(iris_dataset['target']))
```

Значення чисел задаються масивом iris['target_names']: 0 - setosa, 1 - versicolor, а 2 - virginica.

Код для ознайомлення зі структурою даних та результати його виконання занесіть у звіт

Є ще кілька варіантів завантаження наборів даних: з таблиці або з URL-адреси зберігання.

Створіть новий файл Python та імпортуйте такі пакети. Тут записані всі необхідні бібліотеки. Їх можна імпортувати або одразу всі, або по мірі використання.

```
# Завантаження бібліотек

from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Все має завантажуватись без помилок. Якщо у вас є помилка, зупиніться. Перед продовженням потрібне робоче середовище SciPy. Подивіться пораду вище про налаштування вашого середовища.

Ми можемо завантажити дані безпосередньо із репозиторію машинного навчання UCI.

Ми використовуємо модуль pandas для завантаження даних. Ми також будемо використовувати pandas для дослідження даних як цілей описової статистики, так і для візуалізації даних.

Зверніть увагу, що під час завантаження даних ми вказуємо імена кожного стовпця. Це допоможе пізніше, коли ми будемо досліджувати дані.

```
# Завантаження датасету
```

```
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.
csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-
width', 'class']
dataset = read_csv(url, names=names)
```

Датасет повинен завантажитись без подій.

Якщо у вас є проблеми з мережею, ви можете завантажити файл iris.csv в робочу директорію і завантажити його за допомогою того ж методу, змінивши URL-адресу на локальне ім'я файлу.

Ми можемо отримати швидке уявлення про те, скільки екземплярів (рядків) та скільки атрибутів (стовпців) міститься в датасеті за допомогою методу shape.

```
# shape
print(dataset.shape)
```

Ви повинні побачити 150 екземплярів та 5 атрибутів:

(150, 5)

При дослідженні даних, варто відразу в них зазирнути, для цього є метод head()

```
# Зріз даних head
print(dataset.head(20))
```

Це має вивести перші 20 рядків датасету.

Поглянемо тепер на статистичне резюме кожного атрибута. Статистичне зведення включає кількість екземплярів, їх середнє, мін. і макс. значення, а також деякі відсотки.

```
# Стастичні зведення методом describe
print(dataset.describe())
```

Ви побачите, що всі чисельні значення мають однакову шкалу (сантиметри) та аналогічні діапазони від 0 до 8 сантиметрів. (Тобто нормувати дані в цьому наборі не потрібно, вони вже відномовані і приведені до однієї шкали).

Перевірте кількість екземплярів (рядків), що належать до кожного класу. Ми можемо розглядати це як абсолютний відлік.

```
# Розподіл за атрибутом class
print(dataset.groupby('class').size())
```

Ви побачите, що кожен клас має однакову кількість екземплярів (50 або 33% від датасету). Це фактично ідеальний варіант.

КРОК 2. ВІЗУАЛІЗАЦІЯ ДАНИХ

Перед тим як будувати модель машинного навчання, непогано було б досліджувати дані, щоб зрозуміти, чи можна легко вирішити поставлене завдання без машинного навчання, чи міститься потрібна інформація в даних.

Крім того, дослідження даних – це добрий спосіб виявити аномалії та особливості. Наприклад, цілком можливо, що деякі ваші іриси виміряні в дюймах, а не в сантиметрах. **У реальному світі нестиковки в даних та несподіванки дуже поширені!**

Один із найкращих способів досліджувати дані – візуалізувати їх.

Ми розглянемо два типи графіків:

Одновимірні (Univariate) графіки, щоб краще зрозуміти кожен атрибут.

Багатомірні (Multivariate) графіки, щоб краще зрозуміти взаємозв'язок між атрибутами.

Одновимірні графіки

Почнемо з деяких одномірних графіків, тобто графіки кожної окремої змінної. Враховуючи, що вхідні змінні є числовими, ми можемо створювати діаграму розмаху (або "скриню з вусами", по-англійському "box and whiskers diagram") кожного з них.

```
# Діаграма розмаху
dataset.plot(kind='box', subplots=True, layout=(2,2),
sharex=False, sharey=False)
pyplot.show()
```

Це дає більш чітке уявлення про розподіл атрибутів на вході.

Графіки функції занесіть у звіт!

Діаграма розмаху атрибутів вхідних даних

Ми також можемо створити гістограму вхідних даних кожної змінної, щоб отримати уявлення про розподіл.

```
# Гістограма розподілу атрибутів датасета
dataset.hist()
```

```
pyplot.show()
```

Графіки функції занесіть у звіт!

З графіків видно, що вхідні змінні мають близький до гаусівського (нормального) розподіл. Це корисно відзначити, оскільки ми можемо використовувати алгоритми, які можуть використовувати цю властивість.

Багатовимірні графіки

Тепер ми можемо переглянути взаємодії між змінними.

Це можна зробити за допомогою діаграми розсіювання (scatter plot). У діаграмі розсіювання одна ознака відкладається по осі x, а інша ознака – по осі y, кожному спостереженню відповідає точка. На жаль, екран комп'ютера мають лише два виміри, що дозволяє розмістити на графіку лише два (або, можливо, три) ознаки одночасно. Таким чином, важко розмістити на графіку набори даних з більш ніж трьома ознаками. *Один із способів вирішення цієї проблеми - побудувати матрицю діаграм розсіювання (scatterplot matrix) або парні діаграми розсіювання* (pair plots), на яких будуть зображені всі можливі пари ознак. Якщо у вас є невелика кількість ознак, наприклад чотири, як тут, то використання матриці діаграм розсіювання буде цілком розумним. Однак, ви повинні пам'ятати, що матриця діаграм розсіювання не показує взаємодію між усіма ознаками відразу, тому деякі цікаві аспекти даних не будуть виявлені за допомогою цих графіків.

Щоб побудувати діаграми, ми спочатку перетворюємо масив NumPy на DataFrame (основний тип даних у бібліотеці pandas). Pandas має функцію для створення парних діаграм розсіювання під назвою scatter_matrix. По діагоналі цієї матриці розташовуються гістограми кожної ознаки:

```
#Матриця діаграм розсіювання
scatter_matrix(dataset)
pyplot.show()
```

Матриця діаграм розсіювання для набору даних Iris, колір точок даних визначається мітками класів

Поглянувши на графік, ми можемо побачити, що, схоже, вимірювання чашолистків та пелюсток дозволяють відносно добре розділити три класи. Це означає, що модель машинного навчання, можливо, зможе навчитися розділяти їх.

Код для візуалізації та отримані графіки занесіть у звіт

КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ

На основі даних (dataset) нам потрібно побудувати модель машинного навчання, яка передбачить сорти ірису для нового набору вимірювань. Але перш, ніж ми застосувати нашу модель до нового набору, ми повинні переконатися, що модель насправді працює і її прогнозам можна довіряти.

На жаль, для оцінки якості моделі ми не можемо використовувати дані, які ми взяли для побудови моделі. Це зумовлено тим, що наша модель просто запам'ятає весь навчальний набір і тому вона завжди буде передбачати правильну мітку для будь-якої точки даних у навчальному наборі. Це «запам'ятовування» нічого не говорить нам про узагальнюючу здатність моделі (іншими словами, ми не знаємо, чи ця модель так само добре працюватиме на нових даних).

Для оцінки ефективності моделі ми пред'являємо їй нові розмічені дані (розмічені дані, які вона раніше не бачила). Зазвичай це робиться шляхом розбиття зібраних даних (у даному випадку 150 квіток) на дві частини. Одна частина даних використовується для побудови нашої моделі машинного навчання і називається навчальними даними (training data) або навчальним набором (training set). Інші дані будуть використані для оцінки якості моделі, їх називають тестовими даними (test data), тестовим набором (test set) або контрольним набором (hold-out set).

У бібліотеці scikit-learn є функція train_test_split, яка перемішує набір даних та розбиває його на дві частини. Ця функція відбирає в навчальний набір 80% рядків даних із відповідними мітками. 20% даних, що залишилися, з мітками оголошуються тестовим набором. Питання, скільки даних відбирати в навчальний і тестовий набори, є дискусійним, проте використання тестового набору, що містить 20% даних, є хорошим правилом.

У scikit-learn дані, як правило, позначаються великою **X**, тоді як мітки позначаються рядковою ***y***. Це навіяно стандартною математичною формулою ***f(x)=y***, де ***x*** є аргументом функції, а ***y*** – результатом. Відповідно з деякими математичними угодами ми використовуємо велику **X**, тому що дані являють собою двовимірний масив (матрицю) і малим ***y***, тому що цільова змінна - це одновимірний масив (вектор). Викличте функцію train_test_split для наших даних і задайте навчальні дані, навчальні мітки, тестові дані, тестові мітки, використовуючи вищезгадані літери:

```
# Розділення датасету на навчальну та контрольну вибірки
array = dataset.values

# Вибір перших 4-х стовпців
X = array[:,0:4]

# Вибір 5-го стовпця
```

```
y = array[:,4]

# Разделение X и y на обучающую и контрольную выборки
X_train, X_validation, Y_train, Y_validation = train_test_split(X,
Y, test_size=0.20, random_state=1)
```

Перед розбиттям функція train_test_split перемішує набір даних за допомогою генератора псевдовипадкових чисел. Якщо ми просто візьмемо останні 20% спостережень як тестовий набор, всі точки даних будуть мати мітку 2, оскільки всі точки даних відсортовані за мітками (див. вивід для iris['target'], показаний раніше). Використовуючи тестовий набір, що містить лише один із трьох класів, ви не зможете об'єктивно судити про узагальнюючу здатність моделі, таким чином ми перемішуємо наші дані, щоб тестові дані містили всі три класи.

Щоб точно повторно відтворити отриманий результат, ми скористаємося генератором псевдовипадкових чисел з фіксованим стартовим значенням, яке задається за допомогою параметра random_state. Це дозволить зробити результат відтворюваним, тому наведений вище програмний код буде генерувати один і той же результат.

Тепер у вас є дані в X_train і Y_train для підготовки моделей і контрольна вибірка X_validation і Y_validation, які ми можемо використовувати пізніше.

Зверніть увагу, що ми використовували зріз Python для вибору стовпців в масиві NumPy.

Для покращення точності моделі використовуватимемо ***стратифіковану 10-кратну крос-валідацію. Це додаткова процедура і в принципі її можна не використовувати коли об'єм вхідних даних великий. Але наш датасет на 150 рядків (по 50 кожного виду) є невеликим. Тому вона потрібна для підвищення точності моделі.***

Це розділить наш датасет на 10 частин, Для навчання на 9 частинах та тестовій – 1 для перевірки. Навчання повторюватиметься на всіх комбінаціях з вибірок train-test.

Стратифікована означає, що кожен прогін за вибіркою даних буде прагнути мати такий самий розподіл прикладу за класом, як це існує у всьому наборі даних, що навчаються.

Ми встановлюємо випадковий початок (затравка) через random_state аргумент на фіксоване число, щоб гарантувати, що кожен алгоритм оцінюється на тих же вибірках даних, що навчаються. Конкретні випадкові затравки не мають значення.

Використаємо поки що лише одну метрику acccuracy для оцінки якості роботи моделей.

Це співвідношення кількості правильно передбачених прикладів, розділених на загальну кількість прикладів у наборі даних, помноженому на 100, щоб дати відсоток (наприклад, точність 95%).

КРОК 4. КЛАСИФІКАЦІЯ (ПОБУДОВА МОДЕЛІ)

Тепер ми можемо розпочати будувати реальну модель машинного навчання. У бібліотеці scikit-learn є багато алгоритмів класифікації, які ми могли б використовувати для побудови моделі. Ми не знаємо, які алгоритми будуть хороші для цієї задачі або які конфігурації їх використовувати.

Протестуємо 6 різних алгоритмів:

Логістична регресія або логіт-модель (LR)
Лінійний дискримінантний аналіз (LDA)
Метод k-найближчих сусідів (KNN)
Класифікація та регресія за допомогою дерев (CART)
Наївний баєсовський класифікатор (NB)
Метод опорних векторів (SVM)

Тут використовується суміш простих лінійних (LR і LDA) та нелінійних (KNN, CART, NB і SVM) алгоритмів.

У scikit-learn всі моделі машинного навчання реалізовані у власних класах, які називають класами Estimator.

Будуємо і оцінюємо моделі:

```
# Завантажуємо алгоритми моделі
models = []
models.append(('LR', LogisticRegression(solver='liblinear',
multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# оцінюємо модель на кожній ітерації
results = []
names = []

for name, model in models:
     kfold = StratifiedKFold(n_splits=10, random_state=1,
shuffle=True)
     cv_results = cross_val_score(model, X_train, Y_train,
cv=kfold, scoring='accuracy')
     results.append(cv_results)
```

```
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(),
cv_results.std()))
```

Ми також можемо створити графік результатів оцінки моделі та порівняти розбіжність середньої точності кожної моделі. Існує розбір показників точності для кожного алгоритму, тому що кожен алгоритм будемо оцінювати 10 разів (в рамках 10-кратної крос-валідації).

Хороший спосіб порівняти результати для кожного алгоритму полягає у створенні діаграмі розмаху атрибутів вихідних даних та їх вусів для кожного розподілу та порівняння розподілів.

```
# Порівняння алгоритмів
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```

Отримайте та порівняйте результати.

Отримані графіки та результати занесіть у звіт. Виберіть та напишіть чому обраний вами метод класифікації ви вважаєте найкращим.

КРОК 5. ОПТИМІЗАЦІЯ ПАРАМЕТРІВ МОДЕЛІ

Більшість моделей у scikit-learn мають масу параметрів, але більшість їх пов'язані з оптимізацією швидкості обчислень чи призначені для особливих випадків використання. Поки що не потрібно турбуватися про інші параметри моделей. У цьому завданні ми не будемо оптимізувати параметри.. Виведення моделі в scikit-learn може бути дуже довгим, але не треба його лякатися. Трохи досвіду і нам буде легше виконувати цей крок. Частково ми опробовували цей крок у попередньому завданні, коли спробували міняти ядро алгоритму SVM.

КРОК 6. ОТРИМАННЯ ПРОГНОЗУ (ПЕРЕДБАЧЕННЯ НА ТРЕНУВАЛЬНОМУ НАБОРІ)

Щоб зробити прогноз, ми викликаємо метод predict

Ми можемо протестувати модель на всій вибірці даних, що навчаються, і зробити прогноз та перевірку на контрольній вибірці.

```
# Створюємо прогноз на контрольній вибірці
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
```

КРОК 7. ОЦІНКА ЯКОСТІ МОДЕЛІ

Ми можемо оцінити прогноз, порівнявши його з очікуваним результатом контрольної вибірки, а потім обчислити точність класифікації, а також матрицю помилок та звіт про класифікацію.

```
# Оцінюємо прогноз
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

Оцініть точність на контрольній вибірці.

Матриця помилок дає уявлення про одну допущені помилки (сума недіагональних значень).

Звіт про класифікацію передбачає розбивку кожного класу за точністю (precision), повнотою (recall), f1-оцінкою.

КРОК 8. ОТРИМАННЯ ПРОГНОЗУ (ЗАСТОСУВАННЯ МОДЕЛІ ДЛЯ ПЕРЕДБАЧЕННЯ)

Тепер ми можемо отримати прогнози, застосувавши цю модель до нових даних, за якими ми не знаємо правильні мітки. Уявіть, що ми знайшли в дикій природі ірис з довжиною чашолистки 5 см, шириною чашолистки 2.9 см, довжиною пелюстки 1 см і шириною пелюстки 0.2 см. До якого сорту ірису потрібно віднести цю квітку? Ми можемо помістити ці дані в масив NumPy Обчислимо форму масиву, тобто. кількість прикладів (1), помножимо на кількість ознак (4).

Напишіть свій код.Тут код наведено для прикладу він працювати не буде:

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("форма массива X_new: {}".format(X_new.shape))
```

Зверніть увагу, що ми записали вимірювання по одній квітці у двовимірний масив NumPy, оскільки scikit-learn працює з двовимірними масивами даних.

Щоб зробити прогноз, ми викликаємо метод predict об'єкта knn:

```
prediction = knn.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозированная метка: {}".format(
iris_dataset['target_names'][prediction]))
```

Напишіть свій код.*Тут код наведено для прикладу він працювати не буде*.

Оцініть який клас передбачила ваша модель.

Але як дізнатися, чи ми можемо довіряти нашій моделі? Правильний сорт ірису для цього прикладу нам невідомий, адже саме отримання правильних прогнозів і є головним завданням побудови моделі!

Збережіть код робочих програм під назвами: LR_2_task_3.py.
Коди та результати занесіть у звіт.
У висновках опишіть яку якість класифікації за результатами тренування вдалося досягти та до якого класу належить квітка з кроку 8.

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

По аналогії із завданням 2.3 створіть код для порівняння якості класифікації набору даних income_data.txt (із завдання 2.1) різними алгоритмами.

Використати такі алгоритми класифікації:
Логістична регресія або логіт-модель (LR)
Лінійний дискримінантний аналіз (LDA)
Метод k-найближчих сусідів (KNN)
Класифікація та регресія за допомогою дерев (CART)
Наївний баєсовський класифікатор (NB)
Метод опорних векторів (SVM)

Розрахуйте показники якості класифікації для кожного алгоритму
Порівняйте їх між собою. Оберіть найкращий для рішення задачі.
Поясніть чому ви так вирішили у висновках до завдання.

Збережіть код робочих програм під назвами: LR_2_task_4.py.
Коди та результати занесіть у звіт.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Наступний код Python використовує лінійний класифікатор Ridge за допомогою API бібліотеки scikit-learn. Набір даних Iris класифікується за допомогою лінійного класифікатора Ridge. Розраховуються показники якості. Також надано звіт про класифікацію та матрицю плутанини.

Код містить помилки та потребує бібліотеки Seaborn!

```
# ================================================
# Приклад класифікатора Ridge
# ================================================
import numpy as np
```

```
from sklearn.datasets import load_iris
from sklearn.linear_model import RidgeClassifier
iris = load_iris()
X, y = iris.data, iris.target
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size =
0.3, random_state = 0)
clf = RidgeClassifier(tol = 1e-2, solver = "sag")
clf.fit(Xtrain,ytrain)
ypred = clf.predict(X_test)
from sklearn import metrics
print('Accuracy:',
np.round(metrics.accuracy_score(ytest,ypred),4))
print('Precision:',
np.round(metrics.precision_score(ytest,ypred,average =
'weighted'),4))
print('Recall:', np.round(metrics.recall_score(ytest,ypred,average
= 'weighted'),4))
print('F1 Score:', np.round(metrics.f1_score(ytest,ypred,average =
'weighted'),4))
print('Cohen Kappa Score:',
np.round(metrics.cohen_kappa_score(ytest,ypred),4))
print('Matthews Corrcoef:',
np.round(metrics.matthews_corrcoef(ytest,ypred),4))
print('\t\tClassification Report:\n',
metrics.classification_report(ypred,ytest))
from sklearn.metrics import confusion_matrix
from io import BytesIO #neded for plot
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
mat = confusion_matrix(ytest, ypred)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar =
False)
plt.xlabel('true label')
plt.ylabel('predicted label');
plt.savefig("Confusion.jpg")
# Save SVG in a fake file object.
f = BytesIO()
plt.savefig(f, format = "svg")
```

Seaborn – це бібліотека для створення статистичної інфографіки на Python. Він побудований поверх matplotlib, а також підтримує структуру даних numpy і pandas. Він також підтримує статистичні одиниці з SciPy

Виправте код та виконайте класифікацію.

Опишіть які налаштування класифікатора Ridge тут використані та що вони позначають.

Опишіть які показники якості використовуються та їх отримані результати. Вставте у звіт та поясніть зображення Confusion.jpg

Опишіть, що таке коефіцієнт Коена Каппа та коефіцієнт кореляції Метьюза. Що вони тут розраховують та що показують.

Збережіть код робочих програм під назвами: LR_2_task_5.py.
Коди та результати занесіть у звіт.

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 2

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КІМ-ЛР-2-NNN-XXXXX.doc
де NNN – номер групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КІМ-ЛР-2-NNN-XXXXX.pdf

Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Як працює логістична регресія або логіт-модель (LR)?
2. Як працює лінійний дискримінантний аналіз (LDA)?
3. Як працює метод k-найближчих сусідів (KNN)?
4. Як працює класифікація та регресія за допомогою дерев (CART)?
5. Як працює наївний баєсовський класифікатор (NB)?
6. Як працює метод опорних векторів (SVM)?

ЛАБОРАТОРНА РОБОТА № 3
ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ

Мета роботи: *використовуючи спеціалізовані бібліотеки і мову програмування Python дослідити методи регресії та неконтрольованої класифікації даних у машинному навчанні.*

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії попередньої обробки та матеріал поданий у цьому підрозділі. Також доцільно вивчити матеріал поданий в літературі:

Alberto Artasanchez, Prateek Joshi. Artifcial Intelligence with Python. Second Edition. BIRMINGHAM – MUMBAI:Packt Publishing 2020. – 592 p. ISBN 978-1-83921-953-5.

При виконанні роботи можна використовувати Google Colab або Jupiter Notebook.

Регресія - це процес оцінки того, як співвідносяться між собою вхідні та вихідні змінні [1, 10]. Слід зазначити, що вихідні змінні можуть мати значення з безперервного ряду дійсних чисел. Отже, існує безліч результуючих можливостей. Це різко контрастує з процесом класифікації, у якому кількість вихідних класів фіксовано.

У регресії передбачається, що вихідні змінні залежить від вхідних, і завдання полягає у з'ясуванні співвідношення між ними. Звідси вхідні змінні називаються незалежними змінними (або предикторами), а вихідні – залежними (або критеріальними змінними). При цьому не потрібно, щоб вхідні змінні були незалежними один від одного. Існує безліч ситуацій, коли між вхідними змінними існує кореляція.

Регресійний аналіз дозволяє з'ясувати, як змінюється значення вихідний змінної, коли змінюємо лише частина вхідних змінних, залишаючи інші вхідні змінні фіксованими. У разі лінійної регресії передбачається, що вхідні та вихідні змінні пов'язані між собою лінійною залежністю. Це накладає обмеження на нашу процедуру моделювання, але прискорює її та робить більш ефективною.

Іноді лінійної регресії виявляється недостатньо для пояснення співвідношень між вхідними та вихідними змінними. У подібних випадках ми використовуємо поліноміальну регресію, в якій вхідні та вихідні змінні пов'язані між собою поліноміальною залежністю.

З обчислювальної точки зору такий підхід складніший, але забезпечує більш високу точність. Вибір виду регресії для виявлення зазначених

відношень визначається видом конкретної задачі. Регресію часто використовують для прогнозування цін, економічних показників та інше.

Термін *навчання без вчителя (unsupervised learning)* відноситься до процесу побудови моделі машинного навчання, що не вимагає залучення розмічених тренувальних даних [2, 9]. Машинне навчання без вчителя знаходить застосування у багатьох галузях, включаючи сегментування ринку, торгівля акціями, обробка природної мови, машинний зір та ін.

У попередніх лабораторних роботах ми мали справу з даними, з якими асоціювалися позначки (маркери). У разі помічених навчальних даних алгоритми вчаться класифікувати по цих мітках.

Алгоритми навчання без вчителя намагаються будувати моделі, які здатні знаходити підгрупи в заданому наборі даних, використовуючи різні метрики подібності.

Розглянемо, як формулюється завдання навчання, якщо воно проводиться без учителя. Коли у нас є набір даних, які не асоціюються з будь-якими мітками, ми припускаємо, що ці дані генеруються під впливом прихованих змінних, що управляють їх розподілом. У такому разі процес навчання може наслідувати якусь ієрархічну схему, використовуючи на початковому етапі індивідуальні точки даних. Далі можна створювати більш глибокі рівні представлення даних.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 3 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 3.1. Створення регресора однієї змінної

Побудувати регресійну модель на основі однієї змінної. Використовувати файл вхідних даних: data_singlevar_regr.txt.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import pickle
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt
```

Завантажуємо вхідні дані.

```
# Вхідний файл, який містить дані
input_file = 'data_singlevar_regr.txt'
```

У цьому текстовому файлі використовується кома, тому для завантаження даних можна скористатися наступним викликом функції.

```
# Завантаження даних
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбивка даних на навчальний та тестовий набори
num_training = int(0.8 * len(X))
num_test = len(X) - num_training

# Тренувальні дані
X_train, y_train = X[:num_training], y[:num_training]

# Тестові дані
X_test, y_test = X[num_training:], y[num_training:]
```

Створимо об'єкт лінійного регресора та навчимо його, використовуючи тренувальні дані.

```
# Створення об'єкта лінійного регресора
regressor = linear_model.LinearRegression()

regressor.fit(X_train, y_train)
```

Спрогнозуємо результат для тестового набору даних, використовуючи модель, що навчається.

```
# Прогнозування результату
y_test_pred = regressor.predict(X_test)
```

Побудуємо вихідний графік.

```
# Побудова графіка
plt.scatter(X_test, y_test, color='green')
plt.plot(X_test, y_test_pred, color='black', linewidth=4)
plt.xticks(())
plt.yticks(())
plt.show()
```

Обчислимо метричні параметри регресора, порівнюючи справжні значення з передбаченими.

```
print("Linear regressor performance:")
print("Mean absolute error =",
round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test,
y_test_pred), 2))
print("Median absolute error =",
round(sm.median_absolute_error(y_test, y_test_pred), 2))
print("Explain variance score =",
round(sm.explained_variance_score(y_test, y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
```

Створивши модель, ми можемо зберегти її у файлі для подальшого використання. Python надає відмінний модуль, який дозволяє легко це зробити.

```
# Файл для збереження моделі
output_model_file = 'model.pkl'

# Збереження моделі
with open(output_model_file, 'wb') as f:
    pickle.dump(regressor, f)
```

Завантажимо модель з файлу на диску та побудуємо прогноз.

```
# Завантаження моделі
y_test_pred_new = regressor_model.predict(X_test)
print("\nNew mean absolute error =",
round(sm.mean_absolute_error(y_test, y_test_pred_new), 2))
```

Збережіть код робочої програми під назвою LR_3_task_1.py

Код програми, графік функції та результати оцінки якості занесіть у звіт.

Зробіть висновок

Завдання 3.2. Передбачення за допомогою регресії однієї змінної

Побудувати регресійну модель на основі однієї змінної. Використовувати вхідні дані відповідно свого варіанту, що визначається за списком групи у журналі (таблиця 3.1).

Таблиця 3.1

№ за списком	1	2	3	4	5	6	7	8	9	10
№ варіанту	1	2	3	4	5	1	2	3	4	5

№ за списком	11	12	13	14	15	16	17	18	19	20
№ варіанту	1	2	3	4	5	1	2	3	4	5

№ за списком	21	22	23	24	25	26	27	28	29	30
№ варіанту	1	2	3	4	5	1	2	3	4	5

Варіант 1 файл: data_regr_1.txt
Варіант 2 файл: data_regr_2.txt
Варіант 3 файл: data_regr_3.txt
Варіант 4 файл: data_regr_4.txt
Варіант 5 файл: data_regr_5.txt

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Зробити по аналогії з пунктом 3.1.

Збережіть код робочої програми під назвою LR_3_task_2.py
Код програми, графік функції та результати оцінки якості занесіть у звіт.
Зробіть висновок

Завдання 3.3. Створення багатовимірного регресора

Використовувати файл вхідних даних: data_multivar_regr.txt, побудувати регресійну модель на основі багатьох змінних.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures
```

Відкрийте файл, , який містить дані: data_multivar_regr.txt.
У цьому текстовому файлі в якості роздільника використовується кома.
Завантажте данні.
Розбийте дані на навчальний та тестовий набори (як в завданні 3.1).
Створіть та навчіть модель **лінійного регресора** (як в завданні 3.1).
Спрогнозуйте результат для тестового набору даних.

Виведіть на екран метрики якості лінійної регресії.

Linear Regressor performance: Mean absolute error, Mean squared error, Median absolute error, Explained variance score, R2 score.

Створіть **поліноміальний регресор** ступеня 10 та навчіть його на тренувальних даних.

```
# Поліноміальна регресія
polynomial = PolynomialFeatures(degree=10)
X_train_transformed = polynomial.fit_transform(X_train)
```

Візьміть деяку вибіркову точку даних і спрогнозуйте для неї результат. Перший крок полягає в тому, щоб перетворити її на поліном.

```
datapoint = [[7.75, 6.35, 5.56]]
poly_datapoint = polynomial.fit_transform(datapoint)
```

Неважко зазначити, що ця точка дуже близька до точки даних [7.66, 6.29, 5.66], зазначеної в рядку 11 нашого файлу даних. Тому вдало створений регресор повинен передбачити результат, близький до 41.35. Створіть об'єкт лінійного регресора і виконайте підгонку до полінома. Побудуйте прогноз з використанням як лінійного, так і поліноміального регресора, щоб побачити різницю.

```
poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(X_train_transformed, y_train)

print("\nLinear regression:\n",
linear_regressor.predict(datapoint))
print("\nPolynomial regression:\n",
poly_linear_model.predict(poly_datapoint))
```

Оцініть та порівняйте отримані характеристики.

Зверніть увагу, що порівняно з лінійним регресором поліноміальний регресор забезпечує отримання результату, ближчого до значення 41.35. Тобто дає кращі результати.

Збережіть код робочої програми під назвою LR_3_task_3.py
Код програми та результати оцінки якості занесіть у звіт.
Зробіть висновок

Завдання 3.4. Регресія багатьох змінних

Розробіть лінійний регресор, використовуючи набір даних по діабету, який існує в sklearn.datasets.

Набір даних містить 10 вихідних змінних — вік, стать, індекс маси тіла, середній артеріальний тиск і шість вимірювань сироватки крові, отриманих у 442 пацієнтів із цукровим діабетом, а також реакцію, що цікавить, — кількісний показник прогресування захворювання через 1 рік після вихідного рівня. Отже, існує 442 екземпляри з 10 атрибутами. Колонка 11 є кількісною мірою прогресування захворювання через 1 рік після вихідного рівня. Кожен з цих 10 атрибутів був відцентрований по середньому та масштабований за часом стандартного відхилення n_samples (тобто сума квадратів кожного стовпця складає 1). Оригінальні дані можна завантажити з: https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html'.

Використайте всі функції набору даних про діабет, щоб побудувати двовимірний графік лінійної регресії. Побудуйте графік залежності між спостережуваними відповідями в наборі даних і відповідями, передбаченими лінійним наближенням (крапками) та пряму лінію, по цьому графіку, що покаже, як лінійна регресія намагається провести пряму лінію, яка мінімізує залишкову суму квадратів між спостережуваними відповідями в наборі даних і відповідями, передбаченими лінійним наближенням. Також розрахуйте коефіцієнт кореляції R2, середню абсолютну помилку (MAE) і середньоквадратичну помилку (MSE).

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Вам знадобляться:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
```

Завантажте дані

```
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

Поділіть їх на навчальну та тестову вибірки з параметрами test_size = 0.5, random_state = 0

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size =
0.5, random_state = 0)
```

Створіть модель лінійної регресії та натренуйте її.

```
regr = linear_model.LinearRegression()

regr.fit(Xtrain, ytrain)
```

Зробіть прогноз по тестовій вибірці

```
ypred = regr.predict(Xtest)
```

Розрахуйте, підпишіть та виведіть на екран коефіцієнти регресії та показники

regr.coef_
regr.intercept_
r2_score
mean_absolute_error
mean_squared_error

Побудуйте графіки

```
fig, ax = plt.subplots()
ax.scatter(ytest, ypred, edgecolors = (0, 0, 0))
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw = 4)
ax.set_xlabel('Виміряно')
ax.set_ylabel('Передбачено')
plt.show()
```

Збережіть код робочої програми під назвою LR_3_task_4.py
Код програми, графіки та результати оцінки якості занесіть у звіт. Зробіть висновок.

Завдання 3.5. Самостійна побудова регресії

Згенеруйте свої випадкові дані обравши за списком відповідно свій варіант (згідно табл. 3.2) та виведіть їх на графік. Побудуйте по них модель лінійної регресії, виведіть на графік. Побудуйте по них модель поліноміальної регресії, виведіть на графік. Оцініть її якість.

Таблиця 3.2

№ за списком	1	2	3	4	5	6	7	8	9	10
№ варіанту	1	2	3	4	5	6	7	8	9	10

№ за списком	11	12	13	14	15	16	17	18	19	20
№ варіанту	1	2	3	4	5	6	7	8	9	10

№ за списком	21	22	23	24	25	26	27	28	29	30
№ варіанту	1	2	3	4	5	6	7	8	9	10

Варіант 1

```
m = 100
X = 6 * np.random.rand(m, 1) - 5
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

Варіант 2

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.6 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

Варіант 3

```
m = 100
X = 6 * np.random.rand(m, 1) - 4
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

Варіант 4

```
m = 100
X = 6 * np.random.rand(m, 1) - 5
y = 0.7 * X ** 2 + X + 3 + np.random.randn(m, 1)
```

Варіант 5

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.4 * X ** 2 + X + 4 + np.random.randn(m, 1)
```

Варіант 6

```
m = 100
X = np.linspace(-3, 3, m)
y = 2 * np.sin(X) + np.random.uniform(-0.6, 0.6, m)
```

Варіант 7

```
m = 100
X = np.linspace(-3, 3, m)
y = np.sin(X) + np.random.uniform(-0.5, 0.5, m)
```

Варіант 8

```
m = 100
X = np.linspace(-3, 3, m)
y = 2 * np.sin(X) + np.random.uniform(-0.5, 0.5, m)
```

Варіант 9

```
m = 100
X = np.linspace(-3, 3, m)
y = 3 + np.sin(X) + np.random.uniform(-0.5, 0.5, m)
```

Варіант 10

```
m = 100
X = np.linspace(-3, 3, m)
y = 4 + np.sin(X) + np.random.uniform(-0.6, 0.6, m)
```

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Побудуйте вашу модель, та виведіть ваші дані на графік залежності *y=f(X)*.

Що, якщо ваші дані насправді складніші за звичайну пряму лінію? Дивно, але ви насправді можете застосовувати лінійну модель для припасування до нелінійних даних. Простий спосіб передбачає додавання ступенів кожної ознаки у вигляді нових ознак і наступне навчання лінійної моделі на такому розширеному наборі ознак. Цей прийом називається *поліноміальною регресією (polynomial regression)*.

Якщо вхідні данні розподілені нелінійно, то, безумовно, пряму лінію ніколи не буде підігнано під такі дані належним чином. Тому скористайтеся класом PolynomialFeatures з Scikit-Learn, щоб перетворити наші навчальні дані, додавши як нові ознаки квадрат (поліном 2-го ступеня) кожної ознаки (у нашому випадку є тільки одна ознака):

```
PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

Виведіть значення коефіцієнтів полінома X[0] та X_poly на екран.

Тепер X_poly містить первинну ознаку X плюс її квадрат.

Далі підгоніть модель LinearRegresion до таких розширених навчальних даних.

```
lin_reg = LinearRegression( )
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_
```

Зверніть увагу, що за наявності множини ознак поліноміальна регресія здатна знайти зв'язок між ознаками (те, що проста лінійна регресійна модель робити неспроможна). Це стає можливим завдяки тому факту, що клас PolynornialFeatures також додає всі комбінації ознак до заданого ступеня. Наприклад, якщо є дві ознаки a і b, тоді PolynornialFeatures з degree=3 додав би як ознаки a2, a3, b2 і b3, а й комбінації ab, a2b і ab2.

Виведіть графік вашої моделі крапками, а регресію лінією. Наприклад, рис.3.1.

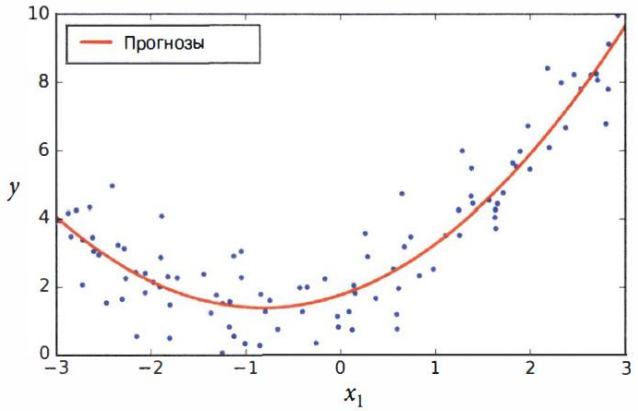


Рис. 3.1

Ваш графік занесіть у звіт.

Запишіть модель вашого варіанта у вигляді математичного рівняння (наприклад $y = 0.5x_1^2 + 1.0x_1 + 2.0 + гауссов\ шум$***) та запишіть отриману вами модель регресії з передбаченими коефіцієнтами (наприклад*** $y = 0.56x_1^2 + 0.93x_1 + 1.78$***).***

Отримані вами коефіцієнти повинні бути близьким до модельних. І це буде означати що модель навчена правильно.

Збережіть код робочої програми під назвою LR_3_task_5.py
Код програми та результати регресії занесіть у звіт.
Зробіть висновок.

Завдання 3.6. Побудова кривих навчання

Побудуйте криві навчання для ваших даних у попередньому завданні.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

У разі виконання поліноміальної регресії високого ступеня, ймовірно, ви краще підганяєте навчальні дані, ніж за допомогою лінійної регресії. Наприклад, на рис. 3.2 демонструється застосування поліноміальної моделі 300-го ступеня до попередніх навчальних даних, а результат порівнюється з чистою лінійною моделлю та квадратичною моделлю (поліноміальною 2-го ступеня).

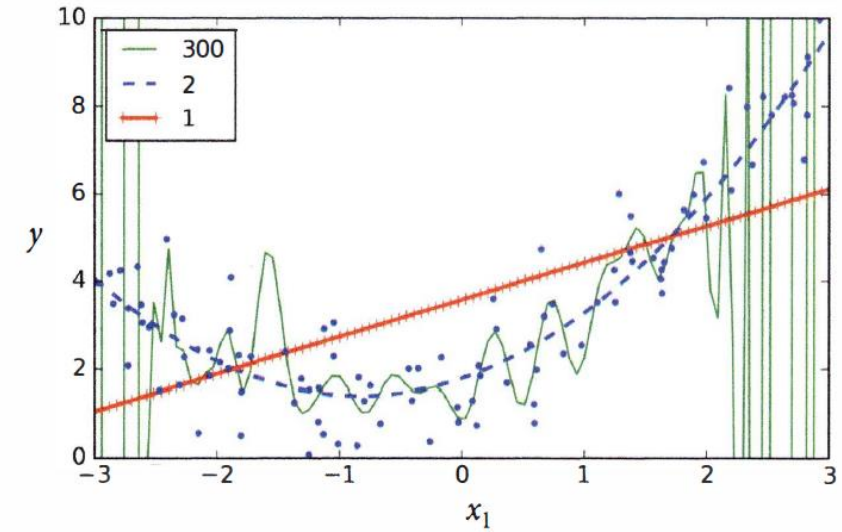


Рис. 3.2

Зверніть увагу, як поліноміальна модель 300-го ступеня коливається, щоб якомога більше наблизитися до навчальних зразків. Зрозуміло, така поліноміальна регресійна модель високого ступеня викликає сильне перенавчання навчальними даними, тоді як лінійна модель - недонавчання ними. У даному випадку добре узагальнюватиметься квадратична модель. Це має сенс, оскільки дані генерувалися з використанням квадратного рівняння, але з урахуванням того, що зазвичай ви *не знатимете функцію*, що використовується для генерації даних, як приймати рішення про те, наскільки складною має бути модель? Як з'ясувати, що модель перенавчається або недонавчається на даних?

У попередніх лабораторних роботах за допомогою перехресної перевірки оцінювалася продуктивність узагальнення моделі. Якщо згідно з метриками перехресної перевірки модель добре виконується на навчальних даних, але погано узагальнюється, то модель перенавчена. Якщо модель погано виконується в обох випадках, тоді вона недонавчена. Так виглядає один із способів сказати, що модель надто проста чи надмірно складна.

Інший спосіб передбачає перегляд кривих навчання (learning curve): вони є графіки продуктивності моделі на навчальному наборі і перевірочному наборі як функції від розміру навчального набору (або ітерації навчання). Щоб отримати такі графіки, потрібно просто навчити модель кілька разів на підмножині різних розмірів, взятих з навчального набору.

Визначте функцію, яка будує криві навчання моделі для встановлених навчальних даних:

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

```
def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val =
          train_test_split(X, y, test_size=0.2)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train_predict,
                                               y_train[:m]))
        val_errors.append(mean_squared_error(y_val_predict, y_val))
    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
```

Побудуйте криві навчання для звичайної лінійної регресійної моделі із попереднього завдання (ваш варіант):

```
lin_reg = LinearRegression()
plot_learning_curves(lin_reg, X, y)
```

Ви повинні отримати щось подібне до рис.3.3.

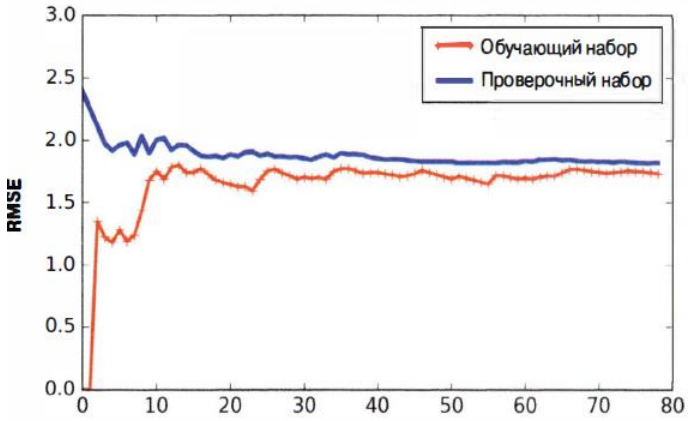


Рис. 3.3.Криві навчання для лінійної моделі

Ваш графік занесіть у звіт.

Тут потрібні деякі пояснення. Насамперед зверніть увагу на продуктивність моделі у разі використання навчальних даних: коли в навчальному наборі є тільки один або два зразки, модель може бути повною мірою підігнана до них, що пояснює початок кривої з нульової помилки. Але в міру додавання зразків у навчальний набір ідеальна підгонка моделі до навчальних даних стає неможливою, як через те, що дані зашумлені, так і тому, що вони зовсім відрізняються від лінійних. Отже, помилка на навчальних даних

рухається вгору, поки не стабілізується, коли додавання нових зразків у навчальний набір не робить середню помилку набагато краще чи гірше. Тепер перейдемо до продуктивності моделі на перевірочних даних. Коли модель навчалася на незначній кількості зразків, вона нездатна узагальнюватися належним чином, а тому помилка перевірки спочатку досить велика. Потім у міру того, як модель бачить все більше навчальних зразків вона навчається, а помилка перевірки відповідно повільно знижується. Однак пряма лінія знову не в змозі добре змоделювати дані, тому помилка стабілізується поблизу іншої кривої.

Такі криві навчання типові для недонавченої моделі. Обидві криві стабілізуються; вони розташовані близько одна до одної і знаходяться досить високо.

Якщо ваша модель недонавчена на навчальних даних, тоді додавання додаткових навчальних зразків не допоможе. Вам потрібно вибрати складнішу модель або знайти найкращі ознаки.

Тепер побудуйте криві навчання поліноміальної моделі 10-го ступеня на тих самих ваших даних:

```
from sklearn.pipeline import Pipeline

polynomial_regression = Pipeline([
        ("poly_features",
         PolynomialFeatures(degree=10, include_bias=False)),
        ("lin_reg", LinearRegression()),
    ])

plot_learning_curves(polynomial_regression, X, y)
```

Ви повинні отримати щось подібне до рис. 3.4.

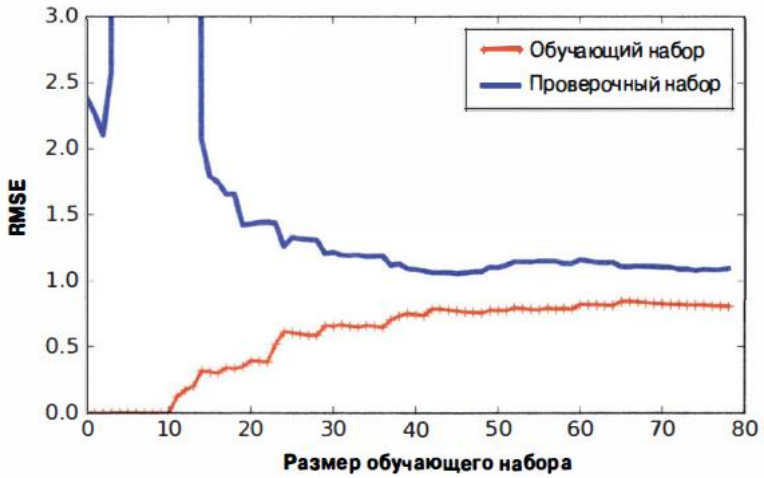


Рис. 3.4. Криві навчання для поліноміальної моделі

Ваш графік занесіть у звіт.

Криві навчання виглядають трохи краще за попередні, але є дві дуже важливі відмінності.

- Помилка на навчальних даних набагато нижча, ніж у випадку лінійної регресійної моделі.

- Між кривими є проміжок. Це означає, що модель виконується значно краще на навчальних даних, ніж на перевірочних даних, демонструючи ознаку перенавчання. Тим не менш, якщо ви застосуєте набагато більший навчальний набір, дві криві продовжать зближення.

Один із способів поліпшення перенавченої моделі полягає у наданні їй додаткових навчальних даних доти, доки помилка перевірки не досягне помилки навчання.

Компроміс між зміщенням та дисперсією

Важливим теоретичним результатом статистики та машинного навчання є той факт, що помилка узагальнення моделі може бути виражена у вигляді суми трьох різних помилок.

Зміщення. Ця частина помилки узагальнення пов'язана з невірними припущеннями, такими як припущення того, що дані є лінійними, коли вони насправді квадратичні. Модель з високим зсувом, швидше за все, недонавчиться на навчальних даних.

Дисперсія. Ця частина пояснюється надмірною чутливістю моделі до невеликих змін у навчальних даних. Модель з багатьма ступенями свободи (така як поліноміальна модель високого ступеня), ймовірно, матиме високу дисперсію і тому перевчитися навчальними даними.

Непереборна похибка. Ця частина з'являється внаслідок шуму самих даних. Єдиний спосіб скоротити непереборну похибку в помилці передбачає очищення даних (наприклад, упорядкування джерел даних, таких як несправні датчики, або виявлення та усунення викидів).

Зростання складності моделі зазвичай збільшує її дисперсію та зменшує зміщення. І навпаки, скорочення складності моделі збільшує її зміщення та зменшує дисперсію. Ось чому це називається компромісом.

Тепер побудуйте криві навчання поліноміальної моделі 2-го ступеня на тих самих ваших даних.

Ваш отриманий графік занесіть у звіт.
Код програми та результати занесіть у звіт.
Програмний код збережіть під назвою LR_3_task_6.py

Завдання 3.7. Кластеризація даних за допомогою методу k-середніх

Провести кластеризацію даних методом k-середніх. Використовувати файл вхідних даних: data_clustering.txt.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Кластеризація - один із найпопулярніших методів навчання без вчителя. Ця методика застосовується для аналізу даних та виділення кластерів серед них. Для знаходження кластерів застосовують різні метрики подібності, такі як евклідова відстань, що дозволяють виділяти підгрупи даних. Використовуючи міру подібності, можна оцінити складність кластера. Таким чином, кластеризація - це процес організації даних у підгрупи, елементи яких подібні між собою відповідно до деяких критеріїв.

Наше завдання полягає в тому, щоб ідентифікувати приховані властивості точок даних, що визначають їхню приналежність до однієї і тієї ж підгрупи. Універсальних метричних параметрів подібності, які б у всіх випадках працювали, не існує. Все визначається конкретикою завдання. Наприклад, нас може цікавити знаходження представницької точки даних для кожної підгрупи або викидів. Залежно від ситуації ми вибираємо ту або іншу метрику, яка, на нашу думку, найбільш повно навчає специфіку завдання.

Метод k-середніх (k-means) - це добре відомий алгоритм кластеризації. Його використання передбачає, що кількість кластерів заздалегідь відома. Далі ми сегментуємо дані до підгруп, застосовуючи різні атрибути даних. Ми починаємо з того, що фіксуємо кількість кластерів та, виходячи з цього, класифікуємо дані. Основна ідея полягає в оновленні положень центроїдів (центрів тяжіння кластеру, або головні точки) на кожній ітерації. Ітеративний процес продовжується до тих пір, поки всі центроїди не займуть оптимального положення.

Як неважко здогадатися, у цьому алгоритмі вибір початкового розташування центроїдів відіграє дуже важливу роль, оскільки це безпосередньо впливає на кінцеві результати. Одна із стратегій полягає в тому, щоб центроїди розташовувалися на якомога більшій відстані один від одного. Базовому методу k-середніх відповідає випадкове розташування центроїдів, тоді як у вдосконаленому варіанті методу (k-means++) ці точки вибираються алгоритмічно з списку вхідних точок даних. На початку процесу робиться спроба розташувати центри кластерів на великих відстанях один від одного, щоб забезпечити швидку збіжність. Потім ми перебираємо дані навчального набору та покращуємо стартове розбиття на кластери за допомогою віднесення кожної точки до найближчого кластерного центру.

Завершення перебору всіх точок набору даних означає закінчення першої ітерації. У цьому етапі точки виявляються згрупованими виходячи з початкових положень центрів кластерів. Далі нам необхідно заново вирахувати положення центроїдів, відштовхуючись від новихкластерів, отриманих наприкінці першої ітерації. Отримавши новий набір до центрів, ми повторюємо весь процес, знову ітеруючи по набору даних і відносячи кожну точку до найближчого центроїду.

У процесі повторення описаних кроків центри кластерів поступово зміщуються до своїх стійких положень. Після виконання певної кількості ітерацій центри кластерів перестануть зміщуватися. Це свідчить про те, що ми досягли сталого розташування центрів кластерів. Отримані ***K*** центроїдів і являють собою остаточну модель k-середніх, які будуть використовуватися для виведення суджень (inference).

Щоб подивитися, як працює метод кластеризації k-середніх, застосуємо його до двовимірних даних. Будемо використовувати дані, що містяться у файлі data_clustering txt. У цьому файлі кожен рядок містить два числа, розділені комою.

У scikit-learnK-means реалізується як об’єкт кластера, який називається sklearn.cluster.KMeans, і використовується для пошуку кластерів.

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics
```

Завантажимо вхідні дані із файлу.

```
# Завантаження вхідних даних
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

Щоб застосувати k-середніх необхідно задати кількість кластерів

```
num_clusters = 5
```

Візуалізуйте вхідні дані, щоб побачити, як виглядає розподіл.

```
# Включення вхідних даних до графіка
```

```
plt.figure()
plt.scatter(X[:,0], X[:,1], marker='o', facecolors='none'
            edgecolors='black', s=80)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Входные данные')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
```

Графік занесіть у звіт.

Ми можемо отримати наочне підтвердження, що наші дані складаються з п'яти груп. Створимо об'єкт KMeans, використовуючи параметри ініціалізації.

Параметр init дозволяє встановити спосіб ініціалізації початкових центрів кластерів. Замість того, щоб вибирати їх випадковим чином, ми використовуємо для цього параметра значення k-means++, яке забезпечує покращений спосіб вибору положень центроїдів, що гарантує швидку збіжність алгоритму. Параметр n_clusters визначає кількість кластерів, тоді як параметр n_init дозволяє вказати, скільки разів повинен виконатися алгоритм, перш ніж буде прийнято рішення щодо найкращого результату.

```
# Створення об'єкту KMeans
kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=10)
```

Навчимо модель k-середніх на вхідних даних.

```
# Навчання моделі кластеризації KMeans
kmeans.fit(X)
```

Щоб візуалізувати межі, ми маємо створити сітку точок та обчислити модель на всіх вузлах сітки. Визначимо крок сітки.

```
# Визначення кроку сітки
step_size = 0.01
```

Далі визначимо саму сітку і переконаємось у тому, що вона охоплює всі вхідні значення.

```
#Відображення точок сітки
```

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_vals, y_vals = np.meshgrid(np.arange(x_min, x_max, step_size),
        np.arange(y_min, y_max, step_size))
```

Спрогнозуйте результати всіх точок сітки, використовуючи навчену модель k-середніх.

```
# Передбачення вихідних міток для всіх точок сітки
output = kmeans.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
```

Відобразіть на графіку вихідні значення та виділіть кожну область своїм кольором.

```
# Графічне відображення областей та виділення їх кольором
output = output.reshape(x_vals.shape)
plt.figure()
plt.clf()
plt.imshow(output, interpolation='nearest',
           extent=(x_vals.min(), x_vals.max(),
                   y_vals.min(), y_vals.max()),
           cmap=plt.cm.Paired,
           aspect='auto',
           origin='lower')
```

Відобразіть вхідні дані на виділених кольором областях.

```
# Відображення вхідних точок
plt.scatter(X[:,0], X[:,1], marker='o', facecolors='none',
           edgecolors='black', s=80)
```

Відобразіть на графіку центри кластерів, отримані з використанням методу k-середніх.

```
# Відображення центрів кластерів
cluster_centers = kmeans.cluster_centers_
plt.scatter(cluster_centers[:,0], cluster_centers[:,1],
           marker='o', s=210, linewidths=4, color='black',
           zorder=12, facecolors='black')
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y min, y max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```
plt.title('Границы кластеров')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

Ваш графік занесіть у звіт.

Збережіть код робочої програми під назвою LR_3_task_7.py

Код програми, графік функції та результати оцінки якості занесіть у звіт.

Зробіть висновок.

Завдання 3.8. Кластеризація K-середніх для набору даних Iris

Виконайте кластеризацію K-середніх для набору даних Iris, який включає три типи (класи) квітів ірису (Setosa, Versicolour і Virginica) з чотирма атрибутами: довжина чашолистка, ширина чашолистка, довжина пелюстки та ширина пелюстки. У цьому завданні використовуйте sklearn.cluster.KMeans для пошуку кластерів набору даних Iris.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Можна написати свій власний код програми з поясненнями по аналогії з попереднім завданням або скористатися підказками, але у цьому випадку ви повинні прокоментувати кожен рядок чи функцію коду де є позначка #.

Код підказки (містить помилки):

```
sklearn.svmimportSVC
from sklearn.metrics import pairwise_distances_argmin

import numpy as np
iris = load_iris()
X = iris['data']
y = iris['target']
#
sklearn.cluster.KMeans(n_clusters = 8, init = 'k-means + +',
n_init = 10, max_iter = 300, tol = 0.0001, precompute_distances =
'auto', verbose = 0, random_state = None, copy_x = True, n_jobs =
None, algorithm = 'auto')
#
kmeans = KMeans(n_clusters = 5)
#
```

```
kmeans.fit(X)
#
y_kmeans = kmeans.predict(X);
#
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 50, cmap =
'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'black', s = 200,
alpha = 0.5);

#
def find_clusters(X, n_clusters, rseed = 2):
#
rng = np.random.RandomState(rseed)
i = rng.permutation(X.shape[0])[:n_clusters]
centers = X[i]
while True:
#
labels = pairwise_distances_argmin(X, centers)
#
new_centers = np.array([X[labels == i].mean(0)
for i in range(n_clusters)])
#
if np.all(centers == new_centers):
break
centers = new_centers

return centers, labels
centers, labels = find_clusters(X, 3)
plt.scatter(X[:, 0], X[:, 1], c = labels,
s = 50, cmap = 'viridis');
#
centers, labels = find_clusters(X, 3, rseed = 0)
plt.scatter(X[:, 0], X[:, 1], c = labels,
s = 50, cmap = 'viridis');
#
labels = KMeans(3, random_state = 0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c = labels,
s = 50, cmap = 'viridis');
```

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_3_task_8.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок

Завдання 3.9. Оцінка кількості кластерів з використанням методу зсуву середнього

Відповідно до рекомендацій, напишіть програму та оцініть максимальну кількість кластерів у заданому наборі даних за допомогою алгоритму зсуву середньою. Для аналізу використовуйте дані, які містяться у файлі data_clustering.txt.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Метод зсуву середнього (Mean Shift) - потужний алгоритм, що використовується в навчанні без вчителя. Цей непараметричний алгоритм часто застосовується при вирішенні завдань кластеризації. Він називається не параметричним, оскільки в ньому не використовуються будь-які припущення щодо базового розподілу даних.

Цей метод контрастує з параметричними підходами, у яких передбачається, що базові дані підпорядковуються стандартному розподілу ймовірностей. Метод зсуву середнього знаходить безліч застосувань у таких областях, як відстеження об'єктів та аналіз даних у реальному часі.

У алгоритмі зсуву середнього весь простір ознак сприймається як функція розподілу ймовірності. Ми починаємо з тренувального набору даних і припускаємо, що ця вибірка відповідає функції розподілу ймовірності. В рамках такого підходу кластери відповідають максимуму базового розподілу. Якщо існують *K* кластерів, то в базовому розподілі існують *K* піків, і метод зсуву середнього ідентифікує ці вершини.

Метою методу зсуву середнього є ідентифікація позицій центрів кластерів. Для кожної точки навчального набору визначається оточуюче її вікно. Потім для цього вікна визначається центроїд і положення вікна оновлюється так, щоб воно відповідало стану нового центроїду. Далі процес повторюється для нового центроїду шляхом визначення вікна навколо нього. У міру продовження описаного процесу ми наближаємось до піку кластера. Кожна точка даних переміщатиметься у напрямку кластера, якому вона належить. Це переміщення здійснюється у напрямку області з більш високою щільністю ймовірності.

Ми продовжуємо процес зміщення центроїдів, що також звуться середніми, до піків кожного кластера. Оскільки середні при цьому зміщуються, метод і називається *зсув середнього*. Цей процес триває до того часу, поки алгоритм не зійдеться, тобто. поки центроїди не перестануть зміщуватися.

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
from itertools import cycle
```

Завантажимо вхідні дані.

```
# Завантаження
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

Зверніть увагу на ширину вікна вхідних даних. *Ширина вікна* (bandwidth) - це параметр базового процесу оцінки щільності розподілу ядра в алгоритмі зсуву середньою. Ширина вікна впливає на загальну швидкість збіжності алгоритму та результуючу кількість кластерів. Отже, цей параметр відіграє важливу роль. Вибір занадто малої ширини вікна може призвести до занадто великої кількості кластерів, тоді як завищені значення цього параметра призводять до злиття окремих кластерів.

Параметр quantile впливає на ширину вікна. Вищі значення цього параметра збільшують ширину вікна, тим самим зменшуючи кількість кластерів.

```
# Оцінка ширини вікна для X
bandwidth_X = estimate_bandwidth(X, quantile=0.1, n_samples=len(X))
```

Навчимо модель кластеризації на основі зсуву середнього, використовуючи отриману оцінку ширини вікна.

```
# Кластеризація даних методом зсуву середнього
meanshift_model = MeanShift(bandwidth=bandwidth_X, bin_seeding=True)
meanshift_model.fit(X)
```

Витягнемо центри всіх кластерів.

```
# Витягування центрів кластерів
cluster_centers = meanshift_model.cluster_centers_
print('\nCenters of clusters:\n', cluster_centers)
```

Витягнемо кількість кластерів.

```
# Оцінка кількості кластерів
labels = meanshift_model.labels_
num_clusters = len(np.unique(labels))
print("\nNumber of clusters in input data =", num_clusters)
```

Візуалізуємо точки даних.

```
# Відображення на графіку точок та центрів кластерів
plt.figure()
markers = 'o*xvs'
for i, marker in zip(range(num_clusters), markers):
    # Отображение на графике точек, принадлежащих
    # текущему кластеру
    plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker,
                color='black')
```

Відобразимо на графіку центр поточного кластера.

```
# Відображення на графіку центру кластера
cluster_center = cluster_centers[i]
plt.plot(cluster_center[0], cluster_center[1], marker='o',
         markerfacecolor='black', markeredgecolor='black',
         markersize=15)
plt.title('Кластеры')
plt.show()
```

Після виконання цього коду на екрані відобразиться графік. У вікні термінала відобразяться координати центрів кластерів.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_3_task_9.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок.

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 3

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КІМ-ЛР-3-NNN-XXXXX.doc
де NNN – номер групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КІМ-ЛР-3-NNN-XXXXX.pdf
Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Навіщо призначена регресія?
2. Чим лінійна регресія відрізняється від поліноміальної?
3. Чим логістична регресія відрізняється від звичайної?
4. Що таке навчання без вчителя?
5. До якого типу машинного навчання відноситься кластеризація?
6. Що такекластеризація?
7. Поясніть суть алгоритму k-середніх?

ЛАБОРАТОРНА РОБОТА № 4
ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Мета роботи: *використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи*.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії попередньої обробки та матеріал поданий у цьому підрозділі. Також доцільно вивчити матеріал поданий в літературі:

Russell, S., & Norvig, P. (3d or 4th Edition). Artificial intelligence: a modern approach. 5 Goodfellow I, Bengio Y, Courville A., Deep Learning // MIT, 2017 – 800 с.

При виконанні роботи можна використовувати Google Colab або Jupiter Notebook.

Термін ансамблеве навчання (ensable learning) відноситься до процесу побудови множини моделей та пошуку такої їх комбінації, яка дозволяє отримати кращі результати ніж кожна з моделей окремо. У якості індивідуальних моделей можуть виступати класифікатори, регресори і інші об’єкти, що дозволяють моделювати дані тим чи іншим способом [8].

Ансамблеве навчання застосовується в багатьох сферах, наприклад, прогностичній класифікації, виявлення аномалій та інше.

Чому застосовується ансамблеве навчання? Щоб це зрозуміти, звернемося до реального прикладу. Припустимо, ви хочете купити новий телевізор, але про останні моделі вам нічого не відомо.

Ваше завдання - купити найкращий телевізор із тих, які пропонуються за доступною для вас ціною, але ви недостатньо добре знаєте ринок, щоб зробити обґрунтований вибір. У подібних випадках ви цікавитеся думкою кількох експертів у цій галузі. Так вам легше прийняти найвірніше рішення. У більшості випадків ви не прив'язуватиметеся до думки якогось одного фахівця і приймете остаточне рішення на основі узагальнення оцінок, зроблених різними людьми. Ми робимо так, тому що прагнемо звести до мінімуму ймовірність прийняття невірних чи недостатньо оптимальних рішень.

При виборі моделі найчастіше виходять з того, щоб вона призводила до найменших помилок на тренувальному наборі даних [5]. Проблема полягає в тому, що такий підхід не завжди працює через можливий ефект перенавчання. Навіть якщо перехресна перевірка моделі підтверджує її адекватність вона може призводити до незадовільних результатів для невідомих даних.

Однією з основних причин ефективності ансамблевого навчання є те, що цей метод дозволяє знизити загальний ризик вибору невдалої моделі. Завдяки тому, що тренування здійснюється на широкій різноманітності навчальних наборів даних, ансамблевий підхід дозволяє отримувати непогані результати для невідомих даних. Якщо ми створюємо модель на основі ансамблевого навчання, то результати, отримані з використанням індивідуальних моделей, повинні виявляти певний розкид. Це дозволяє вловлювати всілякі нюанси, в результаті чого узагальнена модель виявляється більш точною. Зазначене розмаїття результатів досягається за рахунок використання різних навчальних параметрів для індивідуальних моделей, завдяки чому вони генерують різні межі рішень для тренувальних даних. Це означає, що кожна модель буде використовувати різні правила для логічного висновку, тим самим забезпечуючи більш ефективний спосіб валідації кінцевого результату. Якщо між моделями спостерігається узгодженість, то це означає що модель коректна.

Випадкові та гранично випадкові ліси

Випадковий ліс (random forest) - окремий випадок ансамблевого навчання у якому індивідуальні моделі конструюються з використанням дерев рішень [6]. Отриманий ансамбль використовується в подальшому для прогнозування результатів. При конструюванні окремих дерев використовують випадкові підмножини тренувальних даних. Це гарантує розкид даних між різними деревами рішень. Як зазначалося вище, в ансамблевому навчанні дуже важливо забезпечити різнорідність ансамблю індивідуальних моделей.

Однією з найбільших переваг випадкових лісів є те, що вони не перенавчаються. Як ви вже знаєте, у машинному навчанні ця проблема трапляється досить часто. Конструюючи неоднорідну множину дерев рішень за рахунок використання різних випадкових підмножин, ми гарантуємо відсутність перенавчання моделі на тренувальних даних.

У процесі конструювання дерева рішень його вузли послідовно розщеплюються, і їм вибираються найкращі порогові значення, що знижують ентропію кожному рівні. У процесі розщеплення вузлів враховуються в повному обсязі ознаки, що характеризують дані вхідного набору.

Натомість вибирається найкращий спосіб розщеплення вузлів, заснований на поточному випадковому піднаборі ознак, що розглядаються. Включення фактора випадковості збільшує зміщення випадкового лісу, проте дисперсія зменшується завдяки усередненню. Це обумовлює робастність (стійкість до відхилень) результуючої моделі.

Гранично випадкові ліси (extremly random forests) ще більше посилюють роль фактора випадковості. Поряд із випадковим вибором ознак випадково вибираються також граничні значення. Ці випадково генеровані значення стають правилами розбиття, що додатково зменшують варіативність моделі. Тому використання гранично випадкових лісів зазвичай призводить до більш гладких меж прийняття рішень у порівнянні з тими, які вдається отримати за

допомогою випадкових лісів.

Системи рекомендацій

Персоналізація користувацького досвіду була пріоритетом і стала новою мантрою в галузях, орієнтованих на споживача [1, 6]. Можливо, ви помітили, як компанії електронної комерції розміщують персоналізовану рекламу для вас, пропонуючи, що купувати, які новини читати, яке відео дивитися, де/що їсти та з ким вам може бути цікаво спілкуватися (друзі/професіонали) у соціальних мережах. медіа сайти. Системи рекомендацій — це основна система фільтрації інформації, розроблена для прогнозування переваг користувачів і допомоги рекомендувати правильні елементи для створення специфічного для користувача досвіду персоналізації.

Існує два типи рекомендаційних систем: 1) фільтрація на основі вмісту та 2) спільна фільтрація (рис. 4.1).

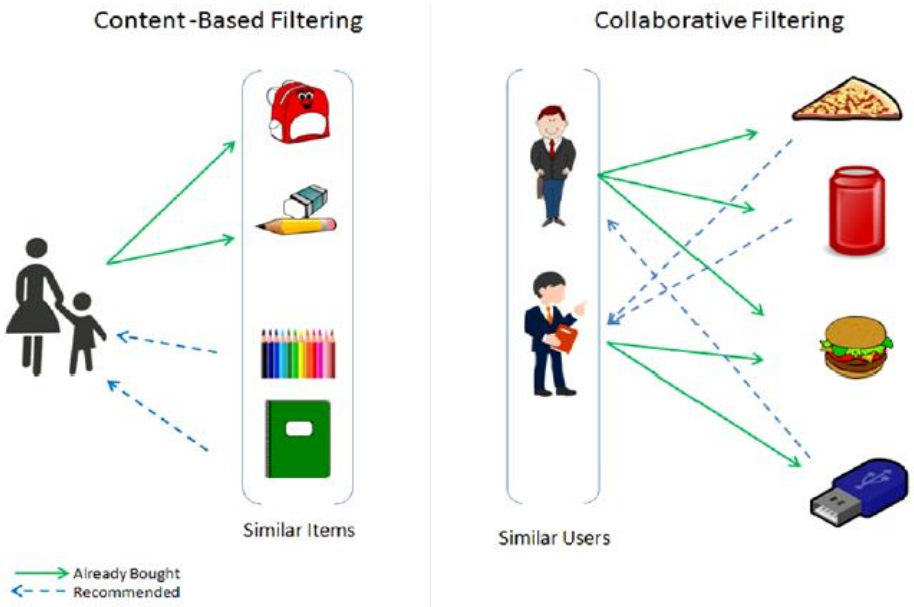


Рис. 4.1

Фільтрація на основі вмісту (Content-Based Filtering)

Цей тип системи зосереджується на атрибуті подібності елементів, щоб дати вам рекомендації. Найкраще це можна зрозуміти на прикладі: якщо користувач придбав товари певної категорії, інші подібні товари з тієї ж категорії рекомендовані користувачеві (див. рис. 4.1).

Алгоритм рекомендацій щодо схожості на основі елементів можна представити у вигляді:

$$\hat{x}_{k,m} = \frac{\sum_{i_b} sim_i(i_m, i_b)(x_{k,b})}{\sum_{i_b} |sim_i(i_m, i_b)|}$$

Спільна фільтрація (Collaborative Filtering (CF))

CF зосереджується на атрибуті подібності користувачів, тобто він знаходить людей зі схожими смаками на основі міри схожості з великої групи користувачів. На практиці існує два типи реалізації CF: на основі пам'яті та на основі моделі.

Тип на основі пам'яті в основному базується на алгоритмі подібності; Алгоритм розглядає елементи, які подобаються подібним людям, щоб створити ранжований список рекомендацій. Потім ви можете відсортувати рейтинговий список, щоб рекомендувати користувачеві перші n елементів.

Алгоритм рекомендацій щодо схожості на основі користувачів можна представити як:

$$pr_{x,k} = m_x + \frac{\sum_{u_y \in N_x} \left(r_{y,k} - m_y\right) \mathrm{sim}\left(\boldsymbol{u}_x, \boldsymbol{u}_y\right)}{\sum_{u_y \in N_x} \left|\mathrm{sim}\left(\boldsymbol{u}_x, \boldsymbol{u}_y\right)\right|}$$

У цій роботі розглянемо та дослідимо приклади рекомендаційних систем.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 4 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 4.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів

Використовувати файл вхідних даних: data_random_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Способи створення класифікаторів обох типів дуже схожі, тому для вказівки того, який класифікатор створюється, ми будемо використовувати вхідний прапор.

Створіть новий файл Python та імпортуйте такі пакети.

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation
from sklearn.ensemble import RandomForestClassifier,
```

```
ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report
from utilities import visualize_classifier
```

Визначимо синтаксичний аналізатор (парсер) аргументів для Python, щоб можна було приймати тип класифікатора як вхідний параметр. Задаючи відповідне значення цього параметра, ми зможемо вибирати тип класифікатора, що створюється.

```
# Парсер аргументів
def build_arg_parser():
      parser = argparse.ArgumentParser(description='Classify \
                  data using Ensemble Learning techniques')
      parser.add_argument('--classifier-type',
                  dest='classifier_type', required=True,
                  choices=['rf', 'erf'], help="Type of \
                  classifier to use; can be either 'rf' or \
                  'erf'")
      return parser
```

Визначимо основну функцію та вилучимо вхідні аргументи.

```
if __name__=='__main__':
# Вилучення вхідних аргументів
args = build_arg_parser().parse_args()
classifier_type = args.classifier_type
```

У файлі data_random_forests.txt кожен рядок містить значення розділені комою. Перші два значення відповідають вхідним даним, останнє – цільовій мітці. У цьому наборі даних містяться три різні класи. Завантажимо дані із цього файлу.

```
# Завантаження вхідних даних
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Розіб'ємо вхідні дані на три класи.

```
# Розбиття вхідних даних на три класи
```

```
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
class_2 = np.array(X[y==2])
```

Візуалізуємо вхідні дані.

```
# Візуалізація вхідних даних
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
      facecolors='white', edgecolors='black',
      linewidth=1, marker='s')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
      facecolors='white', edgecolors='black',
      linewidth=1, marker='o')
plt.scatter(class_2[:, 0], class_2[:, 1], s=75,
      facecolors='white', edgecolors='black',
      linewidth=1, marker='^')
plt.title('Входные данные')
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
      cross_validation.train_test_split(
      X, y, test_size=0.25, random_state=5)
```

Визначимо параметри, які використовуватимемо при конструюванні класифікатора. Параметр n_estimator - це кількість дерев. Параметр max_depth – це максимальна кількість рівнів у кожному дереві. Параметр random_state - це початкове значення для генератора випадкових чисел, необхідне ініціалізації алгоритму класифікатора з урахуванням випадкового лісу.

```
# Класифікатор на основі ансамблевого навчання
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
```

Залежно від того, яке значення вхідного параметра ми надали, класифікатор конструюється на основі випадкового або гранично випадкового лісу.

```
if classifier_type == 'rf':
    classifier = RandomForestClassifier(**params)
else:
    classifier = ExtraTreesClassifier(**params)
```

Навчимо та візуалізуємо класифікатор.

```
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train,
                     'Training dataset')
```

Обчислимо результат на тестовому наборі даних та візуалізуємо його.

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test,
                     'Тестовый набор данных')
```

Перевіримо, як працює класифікатор, вивівши звіт із результатами класифікації.

```
# Перевірка роботи класифікатора
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#"*40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train,
      classifier.predict(X_train), target_names=class_names))
print("#"*40 + "\n")

print("#"*40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred,
      target_names=class_names))
print("#"*40 + "\n")
```

Виконайте цей код, запросивши створення класифікатора на основі випадкового лісу за допомогою прапорця **rf** вхідного аргументу. Введіть у вікні терміналу наступну команду:

```
$ python3 random_forests.py --classifier-type rf
```

У процесі виконання цього коду отримайте ряд зображень та занесіть їх у звіт.

Графік вхідних даних. На графіку квадрати, кола та трикутники представляють три класи. Оцініть візуально, що класи значною мірою перекриваються, проте на цьому етапі це нормально. ***Графік занесіть у звіт.***

Зображення на якому відображені границі класифікатора. ***Графік занесіть у звіт.***

Тепер виконайте той самий код, запросивши створення класифікатора на основі гранично випадкового лісу за допомогою прапорця **erf** вхідного аргументу. Введіть у вікні терміналу наступну команду:

```
$ python3 random_forests.py --classifier-type erf
```

Отримайте зображення то порівняйте його з попереднім. ***Графік занесіть у звіт.*** Зверніть увагу, що в останньому випадку були отримані більш лагідні піки. Це обумовлено тим, що в процесі навчання гранично випадкові ліси мають більше можливостей для вибору оптимальних дерев рішень, тому, як правило, вони забезпечують отримання кращих границь.

Оцінка мір достовірності прогнозів

Якщо ви подивитеся на результати, що відображаються у вікні терміналу, побачите, що для кожної точки даних виводяться ймовірності. Цими ймовірностями вимірюються рівні довірливості (рівні довіри) для кожного класу. Оцінка рівнів довіри відіграє важливу роль у машинному навчанні. Додайте в той же файл наступний рядок, який визначає масив тестових точок даних.

```
# Обчислення параметрів довірливості
test_datapoints = np.array([[5, 5], [3, 6], [6, 4],
                            [7, 2], [4, 4], [5, 2]])
```

Об'єкт класифікатора має убудований метод, призначений для обчислення рівнів довірливості. Класифікуємо кожну точку та обчислимо рівні довірливості.

```
      print("\nConfidence measure:")
      for datapoint in test_datapoints:
            probabilities =
classifier.predict_proba([datapoint])[0]
            predicted_class = 'Class-' +
str(np.argmax(probabilities))
            print('\nDatapoint:', datapoint)
            print('Predicted class:', predicted_class)
```

Візуалізуємо тестові точки даних на підставі меж класифікатора.

```
# Візуалізація точок даних
visualize_classifier(classifier, test_datapoints,
                     [0]*len(test_datapoints),
                     'Тестовые точки данных')
plt.show()
```

Результат виконання цього коду із прапором **rf** ***занесіть у звіт***

У вікні терміналу з'явиться виведена інформація ***Скріншот цієї інформації виріжте та занесіть у звіт.***

Для кожної точки даних обчислюється можливість її належності кожному з трьох класів. Ми вибираємо той клас, якому відповідає найвищий рівень довіри.

Результат виконання коду із прапором **erf** ***занесіть у звіт***.

У вікні терміналу з'явиться виведена інформація ***Скріншот цієї інформації виріжте та занесіть у звіт.***

Збережіть код робочої програми під назвою LR_4_task_1.py

Код програми, графік функції та результати оцінки якості занесіть у звіт.

Зробіть висновок

Завдання 4.2. Обробка дисбалансу класів

Використовуючи для аналізу дані, які містяться у файлі data_imbalance.txt проведіть обробку з урахуванням дисбалансу класів.

Якість класифікатора залежить від даних, що використовуються для навчання. Однією з найпоширеніших проблем, із якими доводиться зіштовхуватися у реальних завданнях, є якість даних. Щоб класифікатор працював надійно, йому необхідно надати рівну кількість точок даних для кожного класу. Однак у реальних умовах гарантувати дотримання цієї умови не завжди можливо. Якщо кількість точок даних для одного класу в 10 разів більше, ніж для іншого, то класифікатор віддаватиме перевагу першому класу. Отже, такий дисбаланс необхідно врахувати алгоритмічно.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import sys
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report
from utilities import visualize_classifier
```

Використовуємо для аналізу дані, які містяться у файлі data_imbalance.txt. У цьому файлі кожен рядок містить значення розділені комою. Перші два значення відповідають даним, останнє – цільовій мітці. У цьому наборі даних є два класи. Завантажимо дані із цього файлу.

```
# Завантаження вхідних даних
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Розіб'ємо вхідні дані на два класи.

```
# Поділ вхідних даних на два класи на підставі міток
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
```

Візуалізуємо вхідні дані, використовуючи точкову діаграму.

```
# Візуалізація вхідних даних
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
                  facecolors='black', edgecolors='black',
                  linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
                  facecolors='white', edgecolors='black',
                  linewidth=1, marker='o')
plt.title('Входные данные')
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбиття даних на навчальний та тестовий набори
```

```
X_train, X_test, y_train, y_test =
            cross_validation.train_test_split(
            X, y, test_size=0.25, random_state=5)
```

Визначимо параметри для класифікатора з урахуванням гранично випадкових лісів. Зверніть увагу на вхідний параметр balance, який керує тим, чи враховуватиметься алгоритмічно дисбаланс класів.

У разі врахування цього фактора ми повинні додати ще один параметр, class_weight, що балансує ваги таким чином, щоб вони були пропорційні до кількості точок даних у кожному класі.

```
# Класифікатор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
if len(sys.argv) > 1:
      if sys.argv[1] == 'balance':
            params = {'n_estimators': 100, 'max_depth': 4,
                      'random_state': 0, 'class_weight': 'balanced'}
      else:
            raise TypeError("Invalid input argument; should be
                            'balance'")
```

Створимо, навчимо і візуалізуємо класифікатор, використовуючи тренувальні дані.

```
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')
```

Передбачимо та візуалізуємо результат для тестового набору даних.

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Тестовый набор данных')
```

```
# Обчислення показників ефективності класифікатора
class_names = ['Class-0', 'Class-1']
print("\n" + "#"*40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train,
      classifier.predict(X_train), target_names=class_names))
print("#"*40 + "\n")
```

```
print("#"*40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred,
      target_names=class_names))
print("#"*40 + "\n")

plt.show()
```

Графік вхідних даних занесіть у звіт.
Графік даних класифікатора для тестового набору занесіть у звіт.

Зверніть увагу що, класифікатору не вдалося визначити фактичну межу між двома класами. В даному випадку обчислену межу представляє чорна пляма у верхній частині малюнка.

У вікні терміналу також відобразиться інформація. ***Скрін з інформацією виріжте та занесіть у звіт!***

Також буде виведено попередження про наявність нульових значень у першому рядку з числовими даними, що призводить до виникнення помилки поділу на нуль (виключення ZeroDivisionError) при спробі обчислення показника fl-score. Щоб це попередження не з'являлося, запустіть код у вікні терміналу із прапором ignore.

```
python3 --W ignore class_imbalance.py
```

Далі для врахування дисбалансу класів виконайте код:

```
python3 class_imbalance.py balance
```

Графік даних класифікатора занесіть у звіт.

У вікні терміналу також відобразиться інформація. ***Скрін з інформацією виріжте та занесіть у звіт!***

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_4_task_2.py

Зробіть висновок

Завдання 4.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Використовуючи дані, що містяться у файлі `data_random_forests.txt`. знайти оптимальних навчальних параметрів за допомогою сіткового пошуку.

У процесі роботи з класифікаторами вам не завжди відомо, які параметри є найкращими. Їх підбір вручну методом грубої сили (шляхом перебору всіх можливих комбінацій) практично нереалізований.

І тут на допомогу приходить сіточний пошук (grid search). Розглянемо як це робиться.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation, grid_search
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report

from utilities import visualize_classifier
```

Використовуємо для нашого аналізу дані, що містяться у файлі data_random_forests.txt.

```
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Розіб'ємо дані на три класи.

```
# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
class_2 = np.array(X[y==2])
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
            cross_validation.train_test_split(
            X, y, test_size=0.25, random_state=5)
```

Задамо сітку значень параметрів, де будемо тестувати класифікатор.

Зазвичай ми підтримуємо постійним значення одного параметра та варіюємо інші. Потім ця процедура повторюється кожного з параметрів.

На разі ми хочемо знайти найкращі значення параметрів n_estimators і max_depth. Визначимо сітку значень параметрів.

```
# Визначення сітки значень параметрів
parameter_grid = [ {'n_estimators': [100],
          'max_depth': [2, 4, 7, 12, 16]},
          {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}
                ]
```

Визначимо метричні характеристики, які має використовувати класифікатор для знаходження найкращої комбінації параметрів.

```
metrics = ['precision_weighted', 'recall_weighted']
```

Для кожної метрики необхідно виконати сітковий пошук, під час якого ми навчатимемо класифікатор конкретної комбінації параметрів.

```
for metric in metrics:
      print("\n##### Searching optimal parameters for", metric)

      classifier = grid_search.GridSearchCV(
                  ExtraTreesClassifier(random_state=0),
                  parameter_grid, cv=5, scoring=metric)
      classifier.fit(X_train, y_train)
```

Виведемо оцінку для кожної комбінації параметрів.

```
print("\nGrid scores for the parameter grid:")
for params, avg_score, _ in classifier.grid_scores_:
     print(params, '-->', round(avg_score, 3))
print("\nBest parameters:", classifier.best_params_)
```

Виведемо звіт із результатами роботи класифікатора.

```
y_pred = classifier.predict(X_test)
print("\nPerformance report:\n")
print(classification_report(y_test, y_pred))
```

Після виконання цього коду у вікні терміналу з'явиться інформація. ***Скріншот цієї інформації занесіть у звіт.***

Виходячи з комбінацій значень параметрів, використаних у сіточному пошуку, тут виведені результати, що відповідають найбільш оптимальній комбінації для показника precision.

Отримайте іншу комбінацію значень параметрів, що забезпечує отримання найкращого значення показника recall. ***Скріншот цієї інформації занесіть у звіт.***

Вона відрізняється від першої, що цілком зрозуміло, оскільки precision і recall - різні метричні характеристики, що вимагають використання різних комбінацій параметрів.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_4_task_3.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок

Завдання 4.4. Обчислення відносної важливості ознак

Коли ми працюємо з наборами даних, що містять N-вимірні точки даних, необхідно розуміти, що не всі ознаки однаково важливі. Одні з них відіграють більшу роль, ніж інші. Маючи в своєму розпорядженні цю інформацією, можна зменшити кількість розмірностей, що враховуються. Ми можемо використовувати цю можливість зниження складності алгоритму та його прискорення. Іноді деякі ознаки виявляються зайвими. Отже, їх можна безболісно виключити із набору даних.

Для обчислення важливості ознак будемо використовувати регресор AdaBoost. Скорочення походить від алгоритму Adaptive Boosting (адаптивна підтримка), який часто застосовується у поєднанні з іншими алгоритмами машинного навчання для підвищення їх ефективності. AdaBoost витягує навчальні точки даних для тренування поточного класифікатора, використовуючи деякий розподіл їх ваг. Цей розподіл ітеративно оновлюється, тому наступні класифікатори фокусуються на складніших точках. (Важкі точки - це точки, які були класифіковані неправильно.) Завдяки цьому точки даних, які раніше були неправильно класифіковані, отримують великі ваги у вибірковому наборі даних, що використовується для навчання класифікаторів. Алгоритм об'єднує ці класифікатори в "комітет", який приймає остаточне рішення на підставі виваженої більшості голосів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
from sklearn.metrics import mean_squared_error,
    explained_variance_score
from sklearn import cross_validation
from sklearn.utils import shuffle
```

Ми будемо використовувати вбудований набір даних із цінами на нерухомість, доступний у бібліотеці scikit-learn.

```
# Завантаження даних із цінами на нерухомість
housing_data = datasets.load_boston()
```

Перемішаємо дані, щоби підвищити об'єктивність нашого аналізу.

```
# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target,
               random_state=7)
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
cross_validation.train_test_split(
            X, y, test_size=0.2, random_state=7)
```

Визначимо і навчимо регресор AdaBoost, застосовуючи регресор на основі дерева рішень у якості індивідуальної моделі.

```
# Модель на основі регресора AdaBoost
regressor = AdaBoostRegressor(
            DecisionTreeRegressor(max_depth=4),
            n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)
```

Оцінимо ефективність регресора.

```
# Обчислення показників ефективності регресора AdaBoost
```

```
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred )
print("\nADABOOST REGRESSOR")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
```

Цей регресор має вбудований метод, який можна викликати для обчислення відносної важливості ознак.

```
# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names
```

Нормалізуємо значення відносної ваги ознак.

```
# Нормалізація значень важливості ознак
feature_importances = 100.0 * (feature_importances /
max(feature_importances))
```

Відсортуємо ці значення відображення у вигляді діаграми.

```
# Сортування та перестановка значень
index_sorted = np.flipud(np.argsort(feature_importances))
```

Розставимо мітки вздовж осі X для побудови стовпчастої діаграми.

```
# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5
```

Побудуємо стовпчасту діаграму.

```
# Побудова стовпчастої діаграми
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Оценка важности признаков с использованием регрессора
          AdaBoost')
plt.show()
```

Після виконання цього коду на екрані з'явиться діаграма. ***Діаграму занесіть у звіт та проаналізуйте.***

Відповідно до проведеного аналізу зробіть висновки, які ознаки мають найбільшу роль, а якими можна знехтувати. Висновки занесіть у звіт.

Код програми та результати занесіть у звіт.
Програмний код збережіть під назвою LR_4_task_4.py

Завдання 4.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

Проведіть прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів. Використайте набір даних, доступний за адресою [4]:

https://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor.

Цей набір містить дані про інтенсивність дорожнього руху під час проведення бейсбольних матчів на стадіоні Доджер-стедіум у Лос-Анджелесі.

Щоб зробити дані більш придатними для аналізу, їх необхідно піддати попередній обробці. Попередньо оброблені дані містяться у файлі `traffic_data.txt`. У цьому файлі кожен рядок містить рядкові значення, розділені комою. Як приклад розглянемо перший рядок. Значення в цьому рядку відформатовані наступним чином: день тижня, час доби, команда суперника, двійкове значення, що вказує, чи проходить матч (yes/no), кількість транспортних засобів, що проїжджають.

Метою завдання є прогнозування кількості транспортних засобів, що проїжджають дорогою, на підставі наданої інформації.

Отже, необхідно створити регресор, здатний прогнозувати вихідний результат. Створіть такий регресор на основі гранично випадкових лісів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,
    mean_absolute_error
from sklearn import cross_validation, preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import classification_report
```

Завантажимо дані із файлу traffic_data.txt.

```
input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
      for line in f.readlines():
            items = line[:-1].split(',')
            data.append(items)

data = np.array(data)
```

Нечислові ознаки, що містяться серед цих даних, потребують кодування. Крім того, ми повинні простежити за тим, щоб числові ознаки не піддавалися кодуванню. Для кожної ознаки, що потребує кодування необхідно передбачити окремий кодувальник. Ми повинні відстежувати ці кодувальники, оскільки вони знадобляться нам, коли ми захочемо вирахувати результат для невідомої точки даних. Створимо вказані кодувальники.

```
# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
      if item.isdigit():
            X_encoded[:, i] = data[:, i]
      else:
            label_encoder.append(preprocessing.LabelEncoder())
            X_encoded[:, i] = label_encoder[-1]
            .fit_transform(data[:, i])
X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
            cross_validation.train_test_split(
            X, y, test_size=0.25, random_state=5)
```

Навчимо регресор на основі гранично випадкових лісів.

```
# Регресор на основі гранично випадкових лісів
```

```
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)
```

Обчислимо показники ефективності регресора на тестових даних.

```
# Обчислення характеристик ефективності регресора на тестових
даних
 y_pred = regressor.predict(X_test)
 print("Mean absolute error:",
        round(mean_absolute_error(y_test, y_pred), 2))
```

Розглянемо як обчислюється результат для невідомої точки даних.

Для перетворення нечислових ознак на числові значення ми використовуємо кодувальники.

```
# Тестування кодування на одиночному прикладі
 test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
 test_datapoint_encoded = [-1] * len(test_datapoint)
 count = 0
 for i, item in enumerate(test_datapoint):
       if item.isdigit():
             test_datapoint_encoded[i] = int(test_datapoint[i])
       else:
             test_datapoint_encoded[i] =
 int(label_encoder[count].transform(test_datapoint[i]))
             count = count + 1

 test_datapoint_encoded = np.array(test_datapoint_encoded)
```

Спрогнозуємо результат.

```
# Прогнозування результату для тестової точки даних
print("Predicted traffic:",
int(regressor.predict([test_datapoint_encoded])[0]))
```

Виконавши цей код, ви отримаєте як вихідний результат значення 26, яке дуже близько до фактичного значення. У цьому не важко переконатися, звернувшись до файлу даних.

Код програми та результати занесіть у звіт.
Програмний код збережіть під назвою LR_4_task_5.py

Завдання 4.6. Створення навчального конвеєра (конвеєра машинного навчання)

Зазвичай, системи машинного навчання будуються на модульній основі. Конкретна кінцева мета досягається з допомогою формування відповідних комбінацій окремих модулів. У бібліотеці scikit-learn містяться функції, що дозволяють об'єднувати різні моди в єдині конвеєр

Конвеєр може формуватися з модулів, що виконують різні функції, такі як відбір ознак, попередня обробка даних, побудова випадкових лісів, кластеризація і т.п.

Необхідно створити конвеєр, призначений для вибору найбільш важливих ознак з вхідних даних і їх подальшої класифікації з використанням класифікатора на основі гранично випадкового лісу.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
from sklearn.datasets import samples_generator
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
```

Згенеруємо марковані вибіркові дані для процесів навчання та тестування. Пакет scikit-learn включає вбудовану функцію, яка справляється із цим завданням. У наведеному нижче рядку коду створюються 150 точок даних, кожна з яких є 25-мірним вектором. Числові значення у кожному векторі ознак генеруватимуться з використанням генератора випадкових вибірок. Кожна точка даних включає шість інформативних ознак і не містить жодної надлишкової.

Використовуємо наступний код.

```
# Генерування даних
X, y = samples_generator.make_classification(n_samples=150,
        n_features=25, n_classes=3, n_informative=6,
        n_redundant=0, random_state=7)
```

Першим блоком цього конвеєра є селектор ознак. Цей блок відбирає k "найкращих" ознак. Встановимо для k значення 9.

```
# Вибір k найважливіших ознак
k_best_selector = SelectKBest(f_regression, k=9)
```

Наступний блок конвеєра - класифікатор на основі гранично випадкового лісу з 60 деревами та максимальною глибиною, що дорівнює чотирьом. Використовуємо наступний код.

```
# Ініціалізація класифікатора на основі гранично випадкового лісу
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)
```

Створимо конвеєр за допомогою об'єднання описаних блоків. Ми можемо присвоїти ім’я кожному блоку, щоб їх легше було відслідковувати

```
# Створення конвеєра
processor_pipeline = Pipeline([('selector', k_best_selector),
                               ('erf', classifier)])
```

Можна змінювати параметри окремих блоків. Давайте надамо значення 7 параметру k у першому блоці і значення 30 кількість дерев (n_estimators) у другому блоці. Для визначення областей видимості змінних ми використовуємо імена, раніше надані блокам.

```
# Встановлення параметрів
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
```

Навчимо конвеєр, використовуючи згенеровані перед цим вибіркові дані.

```
# Навчання конвеєра
processor_pipeline.fit(X, y)
```

Спрогнозуємо результати для всіх вхідних значень та виведемо їх.

```
# Прогнозування результатів для вхідних даних
output = processor_pipeline.predict(X)
print("\nPredicted output:\n", output)
```

Обчислимо оцінку, використовуючи марковані тренувальні дані.

```
# Виведення оцінки
print("\nScore:", processor_pipeline.score(X, y))
```

Витягнемо ознаки, відібрані блоком селектора. Ми вказали, що хочемо вибрати 7 таких ознак із загальної кількості 25. Використовуємо наступний код.

```
# Виведення ознак, відібраних селектором конвеєра
```

```
status = processor_pipeline.named_steps['selector']
                           .get_support()
# Вилучення та виведення індексів обраних ознак
selected = [i for i, x in enumerate(status) if x]
print("\nIndices of selected features:",
        ', '.join([str(x) for x in selected]))
```

Після виконання цього коду у вікні терміналу відобразиться інформація
Зробіть скрін вікна терміналу та вставте його у звіт.

Напишіть висновок у звіт. У висновках поясніть:
Що міститься у першому списку?
Що означає значення Score?
Що міститься в останньому рядку ?
Збережіть код робочої програми під назвою LR_4_task_6.py

Завдання 4.7. Пошук найближчих сусідів

Для формування ефективних рекомендацій у рекомендаційних системах використовується поняття найближчих сусідів (nearest neighbours), суть якого полягає у знаходженні тих точок заданого набору, які розташовані на найближчих відстанях від зазначеної. Такий підхід часто застосовується для створення систем, що класифікують точку даних на підставі її близькості до різних класів.

Здійсніть пошук найближчих сусідів заданої точки даних.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
```

Визначте вибірку двовимірних точок даних.

```
# Вхідні дані
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4],
        [3.8, 0.9], [7.3, 2.1], [4.2, 6.5], [3.8, 3.7],
        [2.5, 4.1], [3.4, 1.9], [5.7, 3.5], [6.1, 4.3],
        [5.1, 2.2], [6.2, 1.1]])
```

Визначте кількість найближчих сусідів, котрі хочемо витягти.

```
k = 5
```

Визначимо тестову точку даних, на яку будемо вилучати k найближчих сусідів.

```
# Тестова точка даних
test_datapoint = [4.3, 2.7]
```

Відобразимо на графіку вхідні дані, використовуючи як маркери чорні кружки.

```
# Відображення вхідних даних на графіку
plt.figure()
plt.title(' Вхідні дані ')
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')
```

Створимо та навчимо модель на основі методу *k* найближчих сусідів, використовуючи вхідні дані. Застосуємо цю модель для отримання найближчих сусідів нашої тестової точки даних.

```
# Побудова моделі на основі методу k найближчих сусідів
knn_model = NearestNeighbors(n_neighbors=k,
                             algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors(test_datapoint)
```

Виведемо витягнуті з моделі точки даних, що є найближчими сусідами.

```
# Виведемо 'k' найближчих сусідів
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])
```

Візуалізуємо найближчих сусідів.

```
# Візуалізація найближчих сусідів разом із тестовою точкою даних
plt.figure()
plt.title('Ближайшие соседи')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')
```

```
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1],
        marker='o', s=250, color='k', facecolors='none')
plt.scatter(test_datapoint[0], test_datapoint[1],
        marker='x', s=75, color='k')

plt.show()
```

У процесі виконання цього коду на екрані відобразяться два рафіка.
Обидва графіка занесіть у звіт.
У вікні терміналу з'явиться інформація. ***Інформацію занесіть у звіт***

Зробіть висновки в яких укажіть:
Що відображено на першому графіку.
Що відображено на другому графіку.
Що відображено у вікні терміналу.

Збережіть код робочої програми під назвою LR_4_task_7.py

Завдання 4.8. Створити класифікатор методом *k* найближчих сусідів

Класифікатор на основі ***k*** найближчих сусідів – це модель класифікації, в якій задана точка класифікується з використанням алгоритму найближчих сусідів [8]. Для визначення категорії вхідної точки, даний алгоритм знаходить у навчальному наборі ***k*** точок, що є найближчими по відношенню до заданої. Після цього призначений точці даних клас визначається "голосуванням". Ми переглядаємо класи ***k*** елементів отриманим списком і вибираємо з них той клас, якому відповідає найбільша кількість "голосів". Значення ***k*** залежить від конкретного завдання.

Використовуючи для аналізу дані, які містяться у файлі `data.txt`. **створіть класифікатор методом *k* найближчих сусідів**.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets
```

Завантажте вхідні дані з файлу data.txt. Кожен рядок цього файлу містить значення, розділені комою, причому дані представляють чотири класи.

```
# Завантаження вхідних даних
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(np.int)
```

Візуалізуйте вхідні дані, використовуючи чотири маркери різної форми. Нам потрібно перетворити мітки у відповідні маркери, і саме для цього призначена змінна mapper.

```
# Відображення вхідних даних на графіку
plt.figure()
plt.title('Входные данные')
marker_shapes = 'v^os'
mapper = [marker_shapes[i] for i in y]
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')
```

Визначимо кількість найближчих сусідів, які ми хочемо використати.

```
# Кількість найближчих сусідів
num_neighbors = 12
```

Визначимо крок сітки, яку будемо використовувати для візуалізації

```
step_size = 0.01
```

Створимо модель класифікатора методом ***k*** найближчих сусідів.

```
# Створення класифікатора на основі методу k найближчих сусідів
classifier = neighbors.KNeighborsClassifier(num_neighbors,
weights='distance')
```

Навчимо модель, використовуючи тренувальні дані.

```
# Навчання моделі на основі методу k найближчих сусідів
classifier.fit(X, y)
```

Створимо поділки, які використовуватимемо для візуалізації сітки.

```
# Створення сітки для відображення меж на графіку
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_values, y_values = np.meshgrid(np.arange(x_min, x_max,
        step_size), np.arange(y_min, y_max, step_size))
```

Виконаємо класифікатор на всіх точках сітки.

```
# Виконання класифікатора на всіх точках сітки
output = classifier.predict(np.c_[x_values.ravel(),
        y_values.ravel()])
```

Створимо сітку з виділенням кольорів областей для візуалізації результату.

```
# Візуалізація передбачуваного результату
output = output.reshape(x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)
```

Відобразимо навчальні дані поверх колірної карти для візуалізації розташування точок щодо меж.

```
# Накладання навчальних точок на карту
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=50, edgecolors='black', facecolors='none')
```

Задамо граничні значення для осей X та У та вкажемо заголовок.

```
plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())
```

Щоб оцінити ефективність класифікатора, визначимо тестову точку даних. Відобразимо на графіку навчальні точки даних разом із тестовою точкою для візуальної оцінки їхнього взаємного розташування.

```
# Тестування вхідної точки даних
test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Тестовая точка данных')
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')
```

```
plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
        linewidth=6, s=200, facecolors='black')
```

Витягнемо *K* найближчих сусідів тестової точки даних, використовуючи модель класифікатора.

```
# Вилучення K найближчих сусідів
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(np.int)[0]
```

Відобразимо на графіку *K* найближчих сусідів, отриманих на попередньому кроці.

```
# Відображення K найближчих сусідів на графіку
plt.figure()
plt.title('K ближайших соседей')
for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
            linewidth=3, s=100, facecolors='black')
```

Відобразимо на тому ж графіку тестову точку.

```
plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
        linewidth=6, s=200, facecolors='black')
```

Відобразимо на тому ж графіку вхідні дані.

```
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')
```

Виведемо прогнозований результат.

```
print("Predicted output:",
      classifier.predict([test_datapoint])[0])
plt.show()
```

У процесі виконання цього коду на екрані відобразяться чотири графіки. ***Занесіть їх у звіт. Підпишіть що міститься на кожному графіку.***

У вікні терміналу визначте, до якого класу відноситься тестова точка. Дані з терміналу занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_4_task_8.py.

Зробіть висновок.

Завдання 4.9. Обчислення оцінок подібності

При побудові рекомендаційних систем дуже важливу роль відіграє вибір способу порівняння різних об'єктів, що входять до набору даних. Припустимо, що наш набір даних включає інформацію про користувачів та їх переваги. Для того, щоб щось рекомендувати, ми повинні розуміти, як порівнювати смаки різних людей. І тут першому плані виходять оцінки *подібності* (*similarity scores*) [1]. Оцінка подібності дає уявлення про те у якій мірі два об'єкта можуть вважатися аналогічними один одному. Для цієї мети часто використовують оцінки двох типів: евклідові і по Пірсону. В основу евклідової оцінки (метрики) покладено відстань між двома точками даних. Якщо вам необхідно освіжити знання щодо того, що таке евклідова відстань, то загляньте в Вікіпедію (https://ru.wikipedia.org/wiki/Евклідова_метрика). Евклідова відстань не обмежена за величиною. Тому ми беремо відповідне значення та перетворимо його таким чином, щоб нове значення знаходилося в діапазоні від 0 до 1. Якщо евклідова відстань між двома об'єктами велика, то відповідна евклідова оцінка повинна мати невелику величину, оскільки низька оцінка вказує на малу міру подібності між об'єктами. Отже, евклідова відстань обернено пропорційно евклідовій оцінці.

Оцінка подібності за Пірсоном (Pearson score) – це математична міра кореляції двох об'єктів. Для її обчислення використовують коваріацію (covariance) двох об'єктів та їх індивідуальні стандартні відхилення (Standard deviations). Значення цієї оцінки можуть бути змінені в межах від -1 до +1. Оцінка +1 свідчить про високий рівень подібності об'єктів, тоді як оцінка -1 - великі відмінності з-поміж них. Оцінка 0 свідчить про відсутність кореляції між двома об'єктами.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import argparse
import json
import numpy as np
```

Створимо парсер для обробки вхідних аргументів. Ними будуть служити імена двох користувачів та тип оцінки, яка використовуватиметься для обчислення ступеня подібності.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Compute
similarity score')
    parser.add_argument('--user1', dest='user1', required=True,
            help='First user')
    parser.add_argument('--user2', dest='user2', required=True,
            help='Second user')
    parser.add_argument("--score-type", dest="score_type",
            required=True, choices=['Euclidean', 'Pearson'],
            help='Similarity metric to be used')
    return parser
```

Визначимо функцію, що обчислює евклідову оцінку для двох заданих наборів

```
# Обчислення оцінки евклідова відстані між
# користувачами user1 та user2
def euclidean_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')
    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')
```

Визначимо змінну, яка використовуватиметься для відстеження фільмів, які отримали рейтингову оцінку від обох користувачів.

```
# Фільми, оцінені обома користувачами, user1 та user2
common_movies = {}
```

Витягнемо фільми, які отримали рейтингову оцінку від обох користувачів.

```
for item in dataset[user1]:
    if item in dataset[user2]:
        common_movies[item] = 1
```

У разі відсутності фільмів, оцінених обома користувачами, обчислення оцінки подібності стає неможливим.

```
# За відсутності фільмів, оцінених обома користувачами,
```

```
# оцінка приймається рівною 0
if len(common_movies) == 0:
    return 0
```

Обчислимо квадрат різниці між рейтинговими оцінками та використовуємо його для отримання евклідової оцінки.

```
squared_diff = []

for item in dataset[user1]:
    if item in dataset[user2]:
        squared_diff.append(np.square(dataset[user1][item] -
                                      dataset[user2][item]))
return 1 / (1 + np.sqrt(np.sum(squared_diff)))
```

Визначимо функцію, яка обчислює оцінку подібності за Пірсоном для двох заданих користувачів з числа включених до набору даних. У разі відсутності інформації про цих користувачів генерується виняток.

```
def pearson_score(dataset, user1, user2):
    if user1 not in dataset:
    raise TypeError('Cannot find ' + user1 + ' in the dataset')
if user2 not in dataset:
    raise TypeError('Cannot find ' + user2 + ' in the dataset')
```

Визначимо змінну, яка використовуватиметься для відстеження фільмів, які отримали рейтингову оцінку від обох користувачів.

```
# Фільми, оцінені обома користувачами, user1 та user2
common_movies = {}
```

Витягнемо фільми, які отримали рейтингову оцінку від обох користувачів.

```
for item in dataset[user1]:
    if item in dataset[user2]:
        common_movies[item] = 1
```

У разі відсутності фільмів, оцінених обома користувачами, обчислення оцінки подібності стає неможливим.

```
    num_ratings = len(common_movies)
```

```
# За відсутності фільмів, оцінених обома користувачами,
# оцінка приймається рівною 0
if num_ratings == 0:
    return 0
```

Обчислимо суму рейтингових оцінок усіх фільмів, оцінених обома користувачами.

```
# Обчислення суми рейтингових оцінок усіх фільмів,
# оцінених обома користувачами

user1_sum = np.sum([dataset[user1][item] for item in
                    common_movies])
user2_sum = np.sum([dataset[user2][item] for item in
                    common_movies])
```

Обчислимо суму квадратів рейтингових оцінок усіх фільмів, оцінених обома користувачами.

```
# Обчислення Суми квадратів рейтингових оцінок всіх
# фільмів, оцінених обома користувачами
user1_squared_sum = np.sum([np.square(dataset[user1][item])
                            for item in common_movies])
user2_squared_sum = np.sum([np.square(dataset[user2][item])
                            for item in common_movies])
```

Обчислимо суму творів рейтингових оцінок усіх фільмів, оцінених обома користувачами.

```
# Обчислення суми творів рейтингових оцінок всіх
# фільмів, оцінених обома користувачами
sum_of_products = np.sum([dataset[user1][item] *
        dataset[user2][item] for item in common_movies])
```

Обчислимо різні параметри, необхідні обчислення оцінки подібності по Пірсону з допомогою результатів попередніх обчислень.

```
# Обчислення коефіцієнта кореляції Пірсона
Sxy = sum_of_products - (user1_sum * user2_sum /
      num_ratings)
Sxx = user1_squared_sum - np.square(user1_sum) / num_ratings
Syy = user2_squared_sum - np.square(user2_sum) / num_ratings
```

У відсутності відхилення оцінка дорівнює 0

```
if Sxx * Syy == 0:
    return 0
```

Повертаємо значення оцінки за Пірсоном.

```
return Sxy / np.sqrt(Sxx * Syy)
```

Визначимо основну функцію та розберемо вхідні аргументи.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user1 = args.user1
    user2 = args.user2
    score_type = args.score_type
```

Завантажимо рейтингові оцінки з файлу ratings.json у словнику.

```
    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())
```

Обчислимо оцінку подібності виходячи з вхідних аргументів.

```
if score_type == 'Euclidean':
    print("\nEuclidean score:")
    print(euclidean_score(data, user1, user2))
else:
    print("\nPearson score:")
    print(pearson_score(data, user1, user2))
```

Виконаємо цей код для деяких комбінацій вхідних даних. Припустимо, ми хочемо обчислити евклідову оцінку подібності користувачів David Smith та Bill Duffy.

```
$ python3 compute_scores.py --user1 "David Smith" --user2
"Bill Duffy" --score-type Euclidean
```

Після виконання наведеної команди у вікні терміналу відобразиться інформація.

Інформацію занесіть у звіт.

Щоб обчислити оцінку подібності Пірсона для тієї ж пари користувачів, виконайте наступну команду.

```
$ python3 compute_scores.py --user1 "David Smith" --user2
"Bill Duffy" --score-type Pearson
```

У вікні терміналу відобразиться інформація.

Інформацію занесіть у звіт.

Виконайте аналогічні команди для інших поєднань параметрів

Обчисліть оцінки для:

David Smith та Brenda Peterson
David Smith та Samuel Miller
David Smith та Julie Hammel
David Smith та Clarissa Jackson
David Smith та Adam Cohen
David Smith та Chris Duncan

Інформацію занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_4_task_9.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок.

Завдання 4.10. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

Термін колаборативна фільтрація (collaborative filtering) відноситься до процесу ідентифікації шаблонів поведінки об'єктів набору даних з метою прийняття рішень щодо нового об'єкта [1, 5]. У контексті рекомендаційнихсистем метод колаборативної фільтрації використовують дляпрогнозування уподобань нового користувача на підставі наявної інформації про уподобання інших користувачів з аналогічними смаками.

Складаючи прогнози для переваг індивідуальних користувачів, ми використовуємо спільну інформацію про уподобання інших користувачів. Саме тому цей метод фільтрації називається колаборативним.

В даному випадку основне припущення полягає в тому, що якщо дві людини дають однакові рейтингові оцінки деякому набору фільмів, то їх оцінки

фільмів з невідомого набору також будуть приблизно однаковими. Знаходячи загальні оціночні судження щодо одних фільмів, ми можемо прогнозувати оцінки щодо інших фільмів. З попереднього завдання ми дізналися, як порівнювати між собою різних користувачів у межах одного набору даних. Ці методики оцінки подібності використовуються для пошуку користувачів зі схожими уподобаннями в нашому наборі даних. Як правило, колаборативне фільтрування застосовують, коли мають справу з наборами даних великого розміру. Методи такого типу можна використовувати в різних областях, включаючи фінансовий аналіз, онлайн-покупки, маркетингові дослідження, вивчення купівельних звичок і т.п.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import argparse
import json
import numpy as np
from compute_scores import pearson_score
```

Визначимо функцію для парсингу вхідних аргументів. У разі єдиним вхідним аргументом є ім'я користувача.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description=
            'Find users who are similar to the input user ')
    parser.add_argument('--user', dest='user', required=True,
            help='Input user')
    return parser
```

Визначимо функцію, яка знаходитиме в наборі даних користувачів, аналогічних зазначеному. Якщо інформація про вказаний користувач відсутня, генерується виняток.

```
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')
```

Ми вже імпортували функцію, необхідну для обчислення оцінки подібності Пірсона. Обчислимо з її допомогою оцінку подібності за Пірсоном між вказаним користувачем та всіма іншими користувачами у наборі даних.

Обчислення оцінки подібності за Пірсоном між

```
# вказаним користувачем та всіма іншими
# користувачами в наборі даних
scores = np.array([[x, pearson_score(dataset, user,
        x)] for x in dataset if x != user])
```

Відсортуємо оцінки щодо спадання.

```
# Сортування оцінок за спаданням
scores_sorted = np.argsort(scores[:, 1])[::-1]
```

Витягнемо перші num_users користувачів і повернемо масив.

```
# Вилучення оцінок перших 'num_users' користувачів
top_users = scores_sorted[:num_users]
return scores[top_users]
```

Визначимо основну функцію і розберемо вхідні аргументи, щоб витягти ім'я користувача.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user = args.user
```

Завантажимо дані з файлу ratings.json, в якому містяться імена користувачів та рейтингові оцінки фільмів.

```
    ratings_file = 'ratings.json'

with open(ratings_file, 'r') as f:
    data = json.loads(f.read())
```

Знайдемо перших трьох користувачів, аналогічних користувачеві, вказаному за допомогою вхідного аргументу. Якщо хочете, можете вказати як аргумент іншу кількість користувачів. Виведемо імена користувачів разом із оцінками подібності.

```
    print('\nUsers similar to ' + user + ':\n')
    similar_users = find_similar_users(data, user, 3)
    print('User\t\t\tSimilarity score')
    print('-'*41)
    for item in similar_users:
        print(item[0], '\t\t', round(float(item[1]), 2))
```

Виконайте цей код і знайдіть користувачів, аналогічних користувачеві Bill Duffy.

```
$ python3 collaborative_filtering.py --user "Bill Duffy"
```

У вікні терміналу з'явиться інформація про подібних користувачів.
Інформацію занесіть у звіт.

Виконайте той самий код і знайдіть користувачів, аналогічних користувачеві Clarissa Jackson.
Інформацію занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_4_task_10.py
Код програми занесіть у звіт.
Зробіть висновок.

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 4

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КІМ-ЛР-4-NNN-XXXXX.doc
де NNN – номер групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КІМ- ЛР-4-NNN-XXXXX.pdf
Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Як працюють випадкові ліси?
2. Як працюють гранично випадкові ліси?
3. Які типи рекомендаційних систем ви знаєте?
4. В чому сутність рекомендаційних систем, що використовують фільтрацію на основі вмісту?
5. В чому сутність рекомендаційних систем, що використовують спільну фільтрацію.

ЗАКІНЧЕННЯ

Темаика навчальної дисципліни «Штучний інтелект в задачах комп’ютерної інженерії» дуже широка і охоплює декілька наукових напрямків. Зважаючи на досить великий обсяг матеріалу методичну розробку довелося розділити на дві частини.

У частині 1 викладено теоретичний матеріал для підготовки, завдання та методичні рекомендації до виконання лабораторних робіт № 1-4 за темою «Машинне навчання».

У частині 2 методичних рекомендацій планується видати матеріал до лабораторних робіт № 5-8 за іншими темами навчальної дисципліни «Штучний інтелект в задачах комп’ютерної інженерії».

ЛІТЕРАТУРА

1. Alberto Artasanchez, Prateek Joshi. Artifcial Intelligence with Python. Second Edition. Birmingham – Mumbai: Packt Publishing 2020. – 592 p. ISBN 978-1-83921-953-5.
2. Melanie Mitchell. Artificial Intelligence. A Guide for Thinking Humans, London. Penguin 2020. — 448 p. — ISBN 978-0-241-40483-6 . (укр.)
3. Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). Mathematics for machine learning. Cambridge University Press. Available: https://mml-book.github.io/book/mml-book.pdf
4. Булгакова О. С., Зосімов В. В., Поздєєв В. О. Методи та системи штучного інтелекту. Теорія та практика. Навчальний посібник. – Олді плюс, 2020, - 356 с.
5. Методи та системи штучного інтелекту: навч. посіб. / укл. Д.В. Лубко, С.В. Шаров. – Мелітополь: ФОП Однорог Т.В., 2019. – 264 с.
6. Системи штучного інтелекту. Лабораторний практикум. Навч. посібник для здобувачів ступеня магістр за спеціальністю 123 «Комп'ютерні системи та мережі» / Стіренко С., Кочура Ю . К.: КПІ ім. Ігоря Сікорського, 2022. – 24 с. [Електронний ресурс], http:// comsys.kpi.ua
7. Russell, S., & Norvig, P. (3d or 4th Edition). Artificial intelligence: a modern approach. 5 Goodfellow I, Bengio Y, Courville A., Deep Learning // MIT, 2017 – 800 с.
8. Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285. 7 Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.
9. Christopher M. Bishop. Pattern Recognition and Machine Learning. 2006 Springer Science+Business Media, 2006 – 756 p.
10. Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction Second edition, in progress. The MIT Press Cambridge, Massachusetts London, England. 2015. – 338 p.