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.*

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО 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. Однак у разі нелінійно розділених даних, пряма лінія не може бути використана як межа прийняття рішення. Натомість використовується модифікована версія SVM, звана 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. 1.1).

Рис. 1.1. 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

```
Ключи 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. 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("Назви відповідей:
{}".format(iris_dataset['target_names']))
```

Значення feature_names – це список рядків із описом кожної ознаки:

```
print("Назва ознак: \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("Форма масиву 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("Тип масиву 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 має 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.

Коди комітити на GitHab. У кожному звіти повинно бути посилання на GitHab.

Назвіть бланк звіту СШІ-ЛР-2-NNN-XXXXX.doc
де NNN – позначення групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в СШІ-ЛР-2-NNN-XXXXX.pdf