T8_1NN_RANDOMFOREST

1NN ПРОТИ RANDOMFOREST

У цьому завданні буде використовуватися датасет **digits з sklearn.datasets**. Залиште останні 25% об'єктів для контролю якості, розділивши **X і y на X_train, y_train і X_test, y_test**.

Метою завдання буде реалізувати найпростіший метричний класифікатор - метод найближчого сусіда, а також порівняти якість роботи реалізованого вами 1NN з RandomForestClassifier з sklearn на 1000 деревах.

Встановлення найновіших біблотек, щоб не було проблем з числовими відповідями

In [1]:
```
import sklearn
import numpy as np
```

In [2]:
```
# Перевірочні дані для лабораторної роботи отримані з використанням таких версій бібліотек
!pip install "scikit-learn == 0.24.2"
!pip install "numpy == 1.22.4"
```

```
Requirement already satisfied: scikit-learn==0.24.2 in c:\users\admin\anaconda3\lib\site-packages (0.24.2)
Requirement already satisfied: joblib>=0.11 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn==0.24.2) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn==0.24.2) (2.2.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn==0.24.2) (1.22.4)
Requirement already satisfied: scipy>=0.19.1 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn==0.24.2) (1.9.1)
Requirement already satisfied: numpy==1.22.4 in c:\users\admin\anaconda3\lib\site-packages (1.22.4)
```

In [3]:
```
# Перевірити версії бібліотек можна таким чином (перед цим їх потрібно імпортувати)
print('sklearn',sklearn.__version__)
print('numpy',np.__version__)
```

```
sklearn 0.24.2
numpy 1.22.4
```

Завантажуємо необхідні бібліотеки.

In [4]:
```
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

Завантажуємо датасет **digits**. Та позначаємо незалежні та залежні змінні.

In [5]:
```
digits = datasets.load_digits()
X = digits.data
y = digits.target
```

Розділимо **X і y на X_train, y_train і X_test, y_test**, залишивши останні 25% об'єктів для контролю якості.

In [6]:
```
X.shape
```

Out[6]:
```
(1797, 64)
```

In [7]:
```
index = round(X.shape[0]*0.75)
print('index =', index)

X_train, X_test = np.split(X, [index])
y_train, y_test = np.split(y, [index])
```

```
index = 1348
```

In [8]:
```
print('75% =', round(X.shape[0]*0.75))
print('25% =', round(X.shape[0]*0.25))
```

```
75% = 1348
25% = 449
```

In [9]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(1348, 64)
(449, 64)
```

Завдання 1

Реалізуйте самостійно метод одного найближчого сусіда з евклідовою метрикою для задачі класифікації. Можна не брати корінь з суми квадратів відхилень, тому що корінь - монотонне перетворення і не впливає на результат роботи алгоритму.

Ніякої додаткових перетворень з ознаками в цьому завданні робити не потрібно. Ваша реалізація програми може бути наступною: можна для кожного класифікованого об'єкта складати список пар (відстань до точки з навчальної вибірки, мітка класу в цій точці), потім сортувати цей список (стандартно сортування буде спочатку по першому елементу пари, потім по другому) , а потім брати перший елемент (з найменшою відстанню).

Сортування масиву довжиною N вимагає порядку N log N порівнянь (точніше кажучи, вона працює за O (N log N)). Подумайте, як можна легко зменшити час роботи. Крім простого способу знайти найближчий об'єкт за все за N порівнянь, можна спробувати придумати, як розбити простір ознак на частини і зробити структуру даних, яка дозволить швидко шукати сусідів кожної точки. За вибір методу пошуку найближчих сусідів в KNeighborsClassifier з sklearn відповідає параметр algorithm - якщо у вас вже є певний бекграунд в алгоритмах і структурах даних, вам може бути цікаво познайомитися зі структурами даних ball tree і kd tree.

Частка помилок, що допускаються 1NN на тестовій вибірці, – відповідь в завданні 1. Результат округліть до 4-х цифр після коми та запишіть його в відповідну комірку тесту classroom.

In [10]:
```
def metric_euclid(x, y):
    return np.sqrt(np.sum((x - y)**2))
```

In [11]:
```
y_pred1 = []
for test_x in X_test:
    index_min_metric = 0
    min_metric = metric_euclid(test_x, X_train[0])

    for index, train_x in enumerate(X_train):
        metric = metric_euclid(test_x, train_x)
        if (metric < min_metric):
            min_metric = metric
            index_min_metric = index

    y_pred1.append(y_train[index_min_metric])
```

Розрахуємо частку помилок віднявши від 1 таку величину, як *точність(accuracy)*.

In [12]:
```
err_rate1 = 1 - accuracy_score(y_test, y_pred1)
ans1 = round(err_rate1, 4)
print(ans1)
```

```
0.0379
```

Завдання 2

Тепер навчіть на навчальній вибірці RandomForestClassifier (n_estimators = 1000) з sklearn. Зробіть прогнози на тестовій вибірці та оцініть частку помилок класифікації на ній. Ця частка помилок - відповідь в завданні 2. Зверніть увагу на те, як співвідноситься якість роботи випадкового лісу з якістю роботи, мабуть, одного з найпростіших методів - 1NN. Така відмінність - особливість даного датасета, але потрібно завжди пам'ятати, що така ситуація теж може мати місце, і не забувати про прості методи.

Результат округліть до 4-х цифр після коми та запишіть його в відповідну комірку тесту classroom.

In [13]:
```
randforest_clf = RandomForestClassifier(n_estimators = 1000)
randforest_clf.fit(X_train, y_train)
```

Out[13]:
```
RandomForestClassifier(n_estimators=1000)
```

In [14]:
```
y_pred2 = randforest_clf.predict(X_test)
```

Розрахуємо частку помилок віднявши від 1 таку величину, як *точність(accuracy)*.

In [15]:
```
err_rate2 = 1 - accuracy_score(y_test, y_pred2)
ans2 = round(err_rate2, 4)
print(ans2)
```

```
0.0668
```

Завдання 3

Завантажте в classroom створений Jnotebook.