ТЕМА 4
ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМІВ КЛАСИФІКАЦІЇ

Надія І. Недашківська n.nedashkivska@gmail.com

ПЕРЕХРЕСНА ПЕРЕВІРКА МОДЕЛІ

Надія І. Недашківська n.nedashkivska@gmail.com

Сутність перехресної перевірки

- Навчальний набір розбивається на менший навчальний набір і перевірочний набір, відповідна ф-ія train_test_split ().
- Модель навчається на меншому навчальному наборі.
- Модель оцінюється на перевірочному наборі.

K-fold cross-validation:

- Навчальний набір розбивається на К менших наборів (блоків).
- Модель навчається К разів, кожний раз вибирає для перевірки один інший блок і навчається на К-1-му блоці, що залишилися.
- Відповідна функція: sklearn. model_selection. cross_val_score

train_test_split ()

```
from sklearn import model_selection, datasets
import numpy as np
iris = datasets.load_iris()

train_data, test_data, train_labels, test_labels =
        model_selection.train_test_split (iris.data,
        iris.target, test_size = 0.2)
```

train_test_split ()

```
print ('Size of train set: {} objects \n
Size of test set: {} objects'.format
(len(train_data), len(test_data)))
```

```
Size of train set: 120 objects
Size of test set: 30 objects
```

1. KFold cross-validation

Всі наявні дані розбиваємо на K підмножин, при цьому навчаємо модель на (K-1)-й підмножині і перевіряємо на одній підмножині, що залишилася. Таким чином, кожна підмножина 1 раз бере участь у тестуванні і (K-1) раз бере участь у навчанні.

- KFold повертає пару індексів: індекси з навчання та індекси з тестів, за допомогою яких в подальшому можна розбити вибірку.

1. KFold приклад

```
X = range(0,15)

kf = model_selection.KFold(n_splits = 5)
for train_indices, test_indices in kf.split(X):
    print (train_indices, test_indices)
```

```
[ 3  4  5  6  7  8  9 10 11 12 13 14] [0 1 2]
[ 0  1  2  6  7  8  9 10 11 12 13 14] [3 4 5]
[ 0  1  2  3  4  5  9 10 11 12 13 14] [6 7 8]
[ 0  1  2  3  4  5  6  7  8 12 13 14] [9 10 11]
[ 0  1  2  3  4  5  6  7  8  9 10 11] [12 13 14]
```

У результаті кожен фолд складається з 2 об'єктів:
- навчальна вибірка, яка кожного разу містить 12 об'єктів,
- тестова, яка містить 3 об'єкти.

1. KFold приклад

```
X = range(0,15)

kf = model_selection.KFold(n_splits = 5, shuffle = True)
for train_indices, test_indices in kf.split(X):
    print (train_indices, test_indices)
```

```
[ 0  1  3  4  5  6  7  9 10 12 13 14] [ 2  8 11]
[ 0  2  3  4  5  6  8  9 11 12 13 14] [ 1  7 10]
[ 0  1  2  4  5  6  7  8 10 11 13 14] [ 3  9 12]
[ 0  1  2  3  5  7  8  9 10 11 12 14] [ 4  6 13]
[ 1  2  3  4  6  7  8  9 10 11 12 13] [ 0  5 14]
```

Отримали об'єкти в кожній тестовій множині, що розташовані не по порядку. Проте, якщо будемо викликати цю функцію декілька разів, то отримаємо кожного разу різні розбиття.

1. KFold приклад

```
X = range(0,15)

kf = model_selection.KFold(n_splits = 5, shuffle = True,
                           random_state = 1)
for train_indices, test_indices in kf.split(X):
    print(train_indices, test_indices)
```

```
[ 0  1  2  4  5  8  9 10 11 12 13 14] [3 6 7]
[ 0  1  3  5  6  7  8  9 11 12 13 14] [ 2  4 10]
[ 2  3  4  5  6  7  8  9 10 11 12 14] [ 0  1 13]
[ 0  1  2  3  4  5  6  7 10 11 12 13] [ 8  9 14]
[ 0  1  2  3  4  6  7  8  9 10 13 14] [ 5 11 12]
```

Повторні запуски функції KFold з random_state = 1 призводять до таких самих розбиттів.

2. StratifiedKFold

- Аналогічна до попередньої, але є істотна відмінність - зберігається співвідношення класів в навчальній і тестовій підвибірках.
- Аргументи цієї функці такі самі як у KFold. Однак, при виклику методу split, який власне виконує розбиття, потрібно передати не тільки самі об'єкти, а й мітки класів на цих об'єктах. Розбиття відбувається з урахуванням міток.

2. StratifiedKFold приклад

```
X = range(0,15)
y = np.array([0] * 5 + [1] * 5 + [2] * 5)
print (y)

skf = model_selection.StratifiedKFold(n_splits = 5,  shuffle = True,
                                      random_state = 0)
for train_indices, test_indices in skf.split(X, y):
    print (train_indices, test_indices)
```

```
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2]
[ 0  2  3  4  6  7  8  9 10 11 12 14] [ 1  5 13]
[ 0  1  3  4  5  6  8  9 11 12 13 14] [ 2  7 10]
[ 1  2  3  4  5  7  8  9 10 11 12 13] [ 0  6 14]
[ 0  1  2  4  5  6  7  8 10 12 13 14] [ 3  9 11]
[ 0  1  2  3  5  6  7  9 10 11 13 14] [ 4  8 12]
```

3. ShuffleSplit

- Дозволяє будувати так звані випадкові перестановки.
- Можна отримати дуже багато вибірок, при цьому немає обмежень на те, скільки разів кожен об'єкт повинен з'явитися в навчальній або тестовій множинах.
- Щоразу діємо з поверненням: отримали одне розбиття і далі можемо будувати інше незалежно від попереднього.
- В якості аргументів функції потрібно вказати кількість розбиттів і розмір тестової множини.
- Аргументи цієї функці такі самі як у KFold. Однак, при виклику методу split, який власне виконує розбиття, потрібно передати не тільки самі об'єкти, а й мітки класів на цих об'єктах. Розбиття відбувається з урахуванням міток.

3. ShuffleSplit приклад

```
X = range(0,15)

ss = model_selection.ShuffleSplit(n_splits = 7, test_size = 0.2)

for train_indices, test_indices in ss.split(X):
    print (train_indices, test_indices)
```

```
[ 3 12  6 13  1  0  2  9 14  5 10 11] [7 4 8]
[ 9  3  5  7 13 11 10 14  0  4  1 12] [8 2 6]
[ 0 12  3 10  5 11  4  1  9 14  2  6] [13  8  7]
[ 7  0  2  4  5  1 14  3 13 12 10 11] [9 8 6]
[ 5  1 10  8 12  6  9  7  0  3 11 14] [2  4 13]
[12 11  9  2  1  5  0  4  8 13  3  7] [10 14  6]
[ 6 12  0 14 10  9  4  3 13 11  1  7] [8 2 5]
```

4. StratifiedShuffleSplit приклад

```
X = range(0,15)
target = np.array([0] * 5 + [1] * 5 + [2] * 5)
print (target)

sss = model_selection.StratifiedShuffleSplit(n_splits = 4,
                                             test_size = 0.2)
for train_indices, test_indices in sss.split(X, target):
    print (train_indices, test_indices)
```

```
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2]
[10  9  0  1  7  4 13 14  5  3 12  8] [11  6  2]
[12  7  0 10  9 14  2 13  5  1  3  8] [ 6  4 11]
[14  1  7  3  8  0 12 11  9  2  5 13] [ 4 10  6]
[ 1 14  4 11  3  5  6  2 13  9 12  8] [ 7  0 10]
```

5. LeaveOneOut приклад

```
X = range(0,15)
loo = model_selection.LeaveOneOut()

for train_indices, test_index in loo.split(X):
    print (train_indices, test_index)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14] [0]
[ 0  2  3  4  5  6  7  8  9 10 11 12 13 14] [1]
[ 0  1  3  4  5  6  7  8  9 10 11 12 13 14] [2]
[ 0  1  2  4  5  6  7  8  9 10 11 12 13 14] [3]
[ 0  1  2  3  5  6  7  8  9 10 11 12 13 14] [4]
[ 0  1  2  3  4  6  7  8  9 10 11 12 13 14] [5]
[ 0  1  2  3  4  5  7  8  9 10 11 12 13 14] [6]
[ 0  1  2  3  4  5  6  8  9 10 11 12 13 14] [7]
...
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13] [14]
```

Решітчатий пошук (grid search)

sklearn. model_selection. GridSearchCV

- Оцінює всі можливі комбінації значень заданих гіперпараметрів, використовуючи перехресну перевірку. Список значень кожного гіперпараметру задається.
- Шукається найкраща комбінація значень гіперпараметрів.

Якщо невідомо, які значення має приймати гіперпараметр, то можна спробувати послідовні ступені 10 або меншого числа.

Застосовується, коли досліджується відносно невелика кількість комбінацій.

Рандомізований пошук (randomized search)

sklearn. model_selection. RandomizedSearchCV

- Оцінює задану кількість випадкових комбінацій, вибирає випадкове значення для кожного гіперпараметра на кожній ітерації.

Переваги:

- Якщо рандомізований пошук виконується X разів, то буде досліджено X різних значень для кожного гіперпараметра, а не лише декілька значень на гіперпараметр як при решітчатому пошуку.
- Ефективне використання обчислювальних ресурсів.

ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМІВ КЛАСИФІКАЦІЇ

Надія І. Недашківська n.nedashkivska@gmail.com

Матриця неточностей (confusion matrix)

Бінарна класифікація

		Спрогнозовані моделлю значення класів	
		F(x) = 0	**F(x) = 1**
Точні значення класів	**y=0**	True Negative (TN)	False Positive (FP)
	y=1	False Negative (FN)	True Positive (TP)

В задачі медичної діагностики, де *y=1* означає наявність хвороби,
y=0 - її відсутність,
клас помилок *False Positive* несе в собі менше загрози ніж *False Negative.*

Намагаємося мінімізувати кількість помилок класів *FP* та *FN.*

cross_val_predict ( ) - Отримання прогнозів на основі моделі

confusion_matrix ( ) – Побудова матриці неточностей

```
Out: array ( [ [ 53272 , 0 ] ,
               [ 0 , 4344 ] ] )
```

Матриця неточностей (confusion matrix)

```
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
```

```
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

```
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
confusion_matrix(y_true, y_pred, labels= ["ant",
"bird", "cat"])
```

```
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

Правильність (Accuracy)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Доля вірних відповідей у загальній кількості відповідей.

Нехай у вибірці із 100 людей маємо 5 хворих. Нехай модель діагностує всіх як «здорових». Отже, 95 здорових людей діагностовано вірно, а 5 насправді хворих – помилково. Accuracy становить 95%.

Тому $Accuracy$ не є найкращим показником якості класифікатора.

Accuracy вважається ефективною, коли маємо справу із збалансованими класами. Однак, якщо кількість об’єктів одного класу значно перевищує інший, то Accuracy покаже хороші результати, що будуть неправдивими.

Оцінка ефективності алгоритмів класифікації

- **Матриця неточностей (confusion matrix)**
- **Точність і повнота, міра F1 (F1 score)**
- **Співвідношення точність / повнота**
- **ROC-крива**
- **Перехресна перевірка**

В Python вони реалізовані в **sklearn. metrics.**

Точність (Precision)

precision_score()

$$Precision = \frac{TP}{TP + FP}$$

TP – кількість позитивних прикладів, які коректно виявлені класифікатором,
FP – кількість хибно виявлених позитивних прикладів.

Наприклад, вказує, яка частка пацієнтів, в яких діагностували хворобу, насправді були хворими.

Нехай маємо 100 об’єктів, з яких 5 мають захворювання. Нехай модель в кожному випадку дає відповідь «хворий». Precision становить 5%.

Precision зазвичай використовують разом з метрикою **Повнота (Recall).**

Повнота (Recall) або Чутливість (Sensitivity)

або доля істинно позитивних класифікацій (True Positive Rate, TPR)

recall_score()

$$Recall = \frac{TP}{TP + FN}$$

Відсоток позитивних прикладів, які коректно виявлені класифікатором.

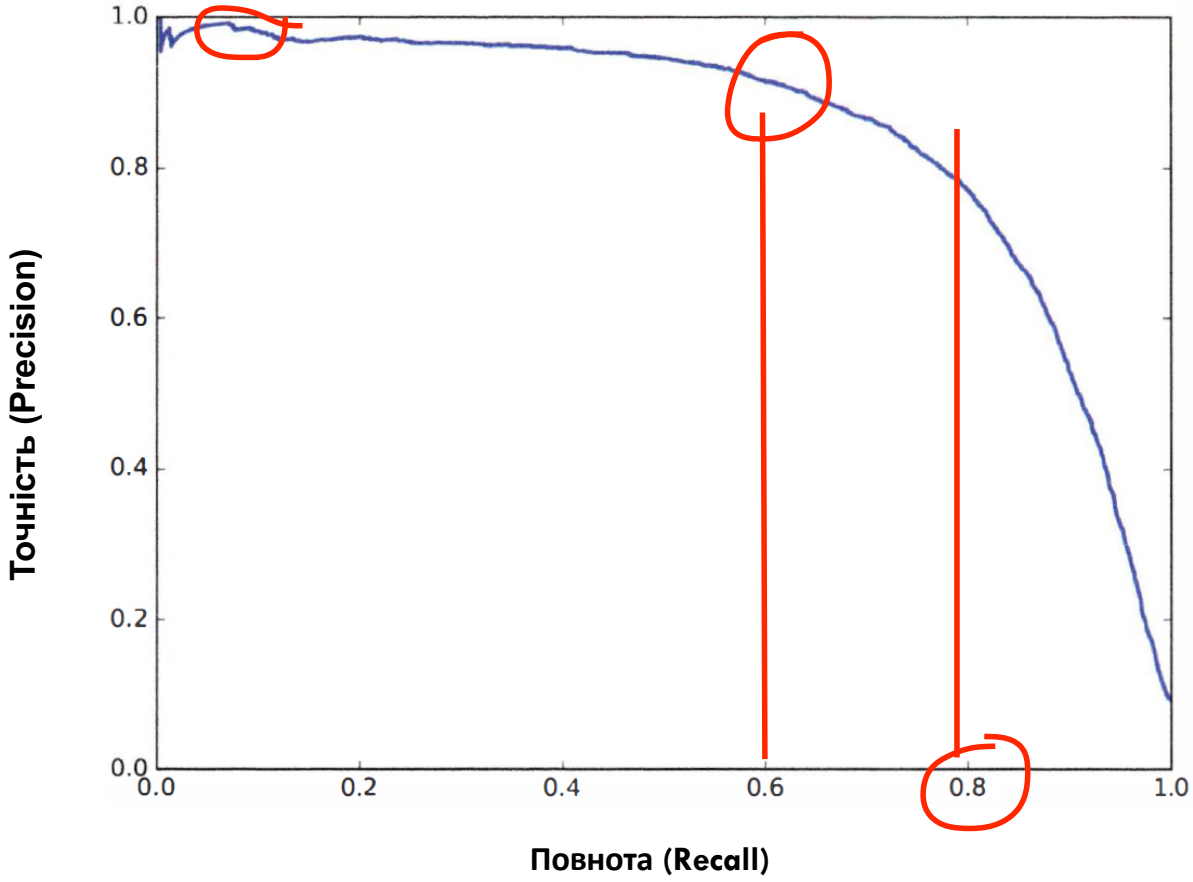
Нехай із 100 пацієнтів 5 мають захворювання. Нехай модель в кожному випадку діагностує захворювання.

Повнота моделі становить 100%. В той час як Точність дорівнює 5%.

Якщо треба мінімізувати FN, то основна увага – на підвищенні значення $Recall$.

Якщо треба мінімізувати FP, то основна увага – на підвищенні $Precision$.

Співвідношення точність/повнота.
Крива точності-повноти (precision-recall (PR) curve)



F-міра (F1 score)

$$F = \frac{2 * precision * recall}{precision + recall}$$

f1_score ( )

Це *середнє гармонічне* двох метрик: $precision$ і $recall$.

Гармонічне середнє надає низьким значенням більшу вагу, на відміну від арифметичного середнього.

F- міра приймає велике значення тільки якщо великі значення мають обидві міри: і $precision$ і $recall$.

Специфічність (Specificity)

або доля істино негативних класифікацій (True Negative Rate, TNR)

$$Specificity = \frac{TN}{TN + FP}$$

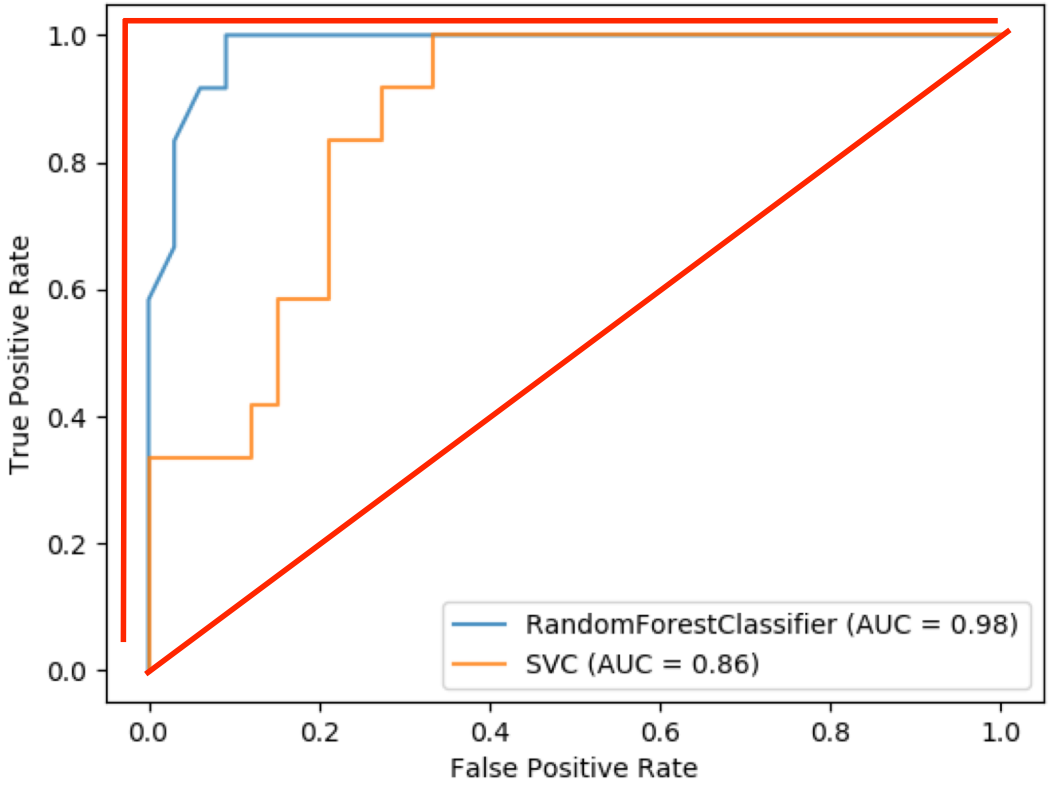
Вказує на частку «негативних» прикладів, які правильно класифіковані як негативні.

ROC крива (Receiver Operating Characteristics Curve)

Вказує **долю істинно позитивних класифікацій (**Recall,TPR**)** по відношенню до **долі хибно позитивних класифікацій** (False Positive Rate, FPR = 1–– Specificity).

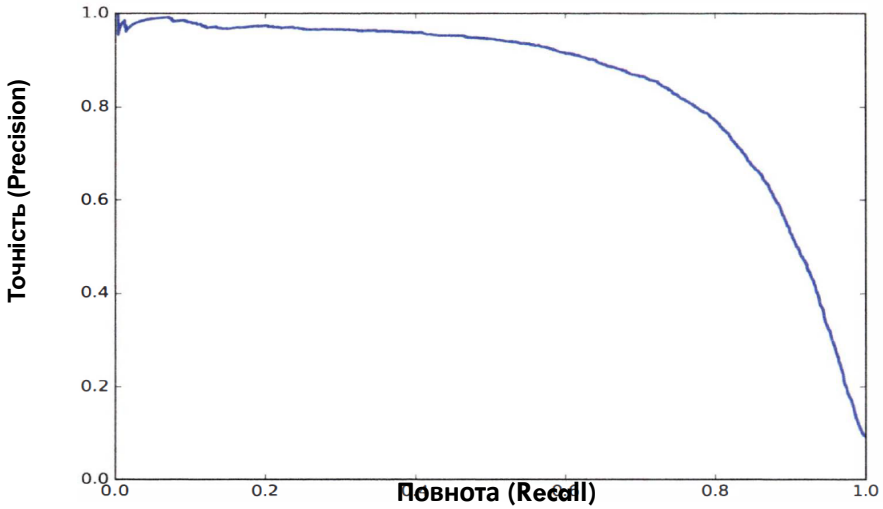
sklearn.metrics. roc_curve() обчислює TPR і FPR.

AUC - Area Under the Curve - площа під ROC-кривою - один із способів порівняння класифікаторів.



Ідеальна модель: AUC=1, випадкова модель: AUC = 0.5.

Яку криву вибрати? Точності-повноти (PR curve) чи ROC криву?



Кривій PR слід віддавати перевагу:

- Якщо хибно позитивні класифікації важливіші за хибно негативні класифікації.
- Якщо мало позитивних класифікацій.