Шевченко Максим, ШІ-2

Звіт

Математичні методи побудови DataSet

Індивідуальний проєкт

Завдання

1) вибрати Dataset та сформулювати задачу, яку будете вирішувати. Можна вибрати датасет тут https://www.kaggle.com/datasets
2) Обов'язково трьома різними відборами (простим випадковим відбором, Бернуллі, систематичним), четвертий відбір-стратифікований-за бажанням, утворити датасет (тренувальної частини) меншої розмірності та подивитись на вирішення задачі на меншому датасеті.
3) Для датасетів різного розміру знайти їх точність, наприклад через learning curve.
4) Порівняти результати для різних відборів та зробити висновки.

Обрана задача та Dataset (1)

У цьому проєкті вирішувалася задача класифікації.

Класифікацію я виконував на трьох різних датасетах:

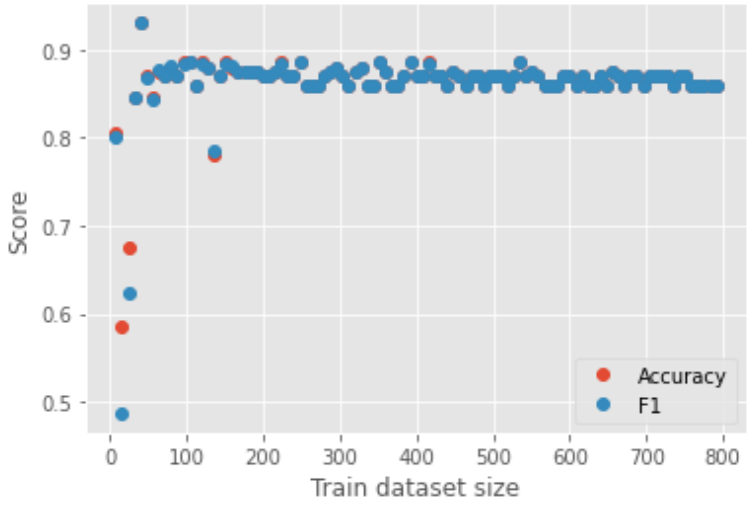
1. Breast Cancer Wisconsin (Diagnostic) Data Set (2 класи, розмір 569 записів, 30 фіч)
2. Cancer Patients Data (3 класи, розмір 1000 записів, 23 фічі)
3. Штучно згенерований за допомогою sklearn.datasets.make_classification (розміри датасету задаються вручну, зупинився на 2 класи, 1000 записів, 500 фіч)

Моделі класифікаторів використовував наступні:

1. sklearn.naive_bayes.GaussianNB()
2. sklearn.naive_bayes.BernoulliNB()
3. sklearn.linear_model.RidgeClassifier(random_state=42)

Метриками точності класифікації є Accuracy та F1-score.

Для реальних даних точність класифікації помітно знижувалася лише при малому розмірі вибірки (плавне зниження при розмірі менше 50, явне погіршення лише при зменшенні розміру до 20 й нижче):



Це не красиво виглядає на графіках, тому у звіт я включу лише результати для штучно згенерованого датасету.

GaussianNB і RidgeClassifier показали кращу точність за BernoulliNB (далі буде скріншот), але тенденції зниження точності класифікації при зменшенні датасету були подібними, тому у звіт включу графіки лише для GaussianNB.

Валідація на повному датасеті

Генерація даних:

```
[43] from sklearn.datasets import make_classification
     X, y = make_classification(n_samples=1000, n_features=500, random_state=42)
     print(X.shape, y.shape)
```

Приклад частини даних:

```
X
```

```
array([[ 1.12465652,  1.47755484, -0.76589679, ..., -0.56040493,
        -1.36103985,  1.65839723],
       [ 0.36668165, -1.10794208, -0.45756445, ...,  0.20233552,
        -1.18862788, -0.91957669],
       [-0.61600782, -1.28028683,  0.53219845, ...,  0.52195906,
         0.14587233, -0.01263937],
       ...,
       [ 0.21967654, -0.21410377, -0.41445045, ..., -1.24209409,
        -0.36848889,  1.25453839],
       [-0.32201197,  0.16208569,  0.48190163, ..., -0.11040116,
         0.33122276,  0.31912084],
       [-0.13738884, -1.36347936,  1.03924293, ..., -2.13642978,
        -0.39537547, -0.70023287]])
```

Вхідний датасет я розділив на тренувальний та випробувальний (train/test) у співвідношенні 800/200 (стандартна практика):

```
[6] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    print(X_train.shape, y_train.shape, X_test.shape, y_test.shape, )

    (800, 23) (800,) (200, 23) (200,)
```

Навчаю моделі класифікаторів на тестовому датасеті та підраховую точність класифікації на випробувальному:

```
from pandas.core.common import random_state
import sklearn.naive_bayes
import sklearn.linear_model
from sklearn.metrics import (classification_report, confusion_matrix,
                             accuracy_score, f1_score)
models = [
    sklearn.naive_bayes.GaussianNB(),
    sklearn.naive_bayes.BernoulliNB(),
    sklearn.linear_model.RidgeClassifier(random_state=42),
]
for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f'\nModel: \t{model}')
    #print(confusion_matrix(y_test,y_pred))
    #print(classification_report(y_test,y_pred))
    print(f'accuracy: {accuracy_score(y_test, y_pred)}, f1: {f1_score(y_test, y_pred, average="weighted")}')
```

Отримані результати:

```
Model:  GaussianNB()
accuracy: 0.86, f1: 0.8601611721611723

Model:  BernoulliNB()
accuracy: 0.39, f1: 0.21884892086330932

Model:  RidgeClassifier(random_state=42)
accuracy: 0.935, f1: 0.9345004666324906
```

Зменшення розміру тренувального датасету (2)

Для побудови більш точних і докладних графіків, коефіцієнт зменшення розміру тренувальної вибірки змінювався плавно від 99% до 1% від початкової тренувальної (800 записів). Розмір випробувальної вибірки не змінювався, щоб виділити вплив розміру тренувальних даних.

```
import numpy as np
sampling_coefficients = list(np.linspace(1, 0, 100, False))[1:]
sampling_coefficients[:5], sampling_coefficients[-5:],
```

```
([0.99, 0.98, 0.97, 0.96, 0.95],
 [0.04999999999999993,
  0.04000000000000036,
  0.03000000000000027,
  0.02000000000000018,
  0.01000000000000009])
```

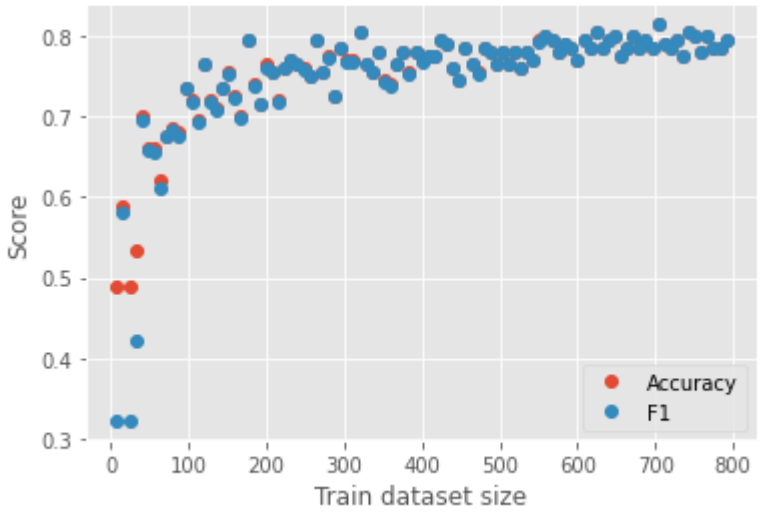
Простий випадковий відбір

Відбір без повернення, бо дублювання даних призведе до спотворення результатів навчання.

Для реалізації відбору я використав стандартну утиліту sklearn.model_selection.train_test_split, яка використовує ПВВБП для поділу вхідного датасету на train/test set. У параметри я передаю бажаний розмір вибірки, яку використовуватиму як тренувальну.

```
results = []
for sc in sampling_coefficients:
    print(f'\t\t {sc}')
    iter_res = [0] * (2 * len(models) + 1)
    X_sample, _, y_sample, _ = train_test_split(X_train, y_train, test_size=1-sc)
    iter_res[0] = X_sample.shape[0]
    print(f'Train dataset size: {X_sample.shape[0]}')
    i = -1
    for model in models:
        i += 2
        model.fit(X_sample, y_sample)
        y_pred = model.predict(X_test)
        # print(f'Model: \t{model} accuracy: {accuracy_score(y_test, y_pred)}, f1
        iter_res[i] = accuracy_score(y_test, y_pred)
        iter_res[i+1] = f1_score(y_test, y_pred, average="weighted")
    print(iter_res)
    results.append(iter_res)
print(results)
```

```
import matplotlib.pyplot as plt
results = np.array(results)
plt.style.use('ggplot')
plt.plot(results[:, 0], results[:, 1], 'o', label = 'Accurac
plt.plot(results[:, 0], results[:, 2], 'o', label = 'F1')
plt.ylabel('Score')
plt.xlabel('Train dataset size')
plt.legend()
plt.show()
```



Бачимо тенденцію поступового зменшення точності класифікації при зменшенні вибірки.

Відбір Бернуллі

Цей відбір передбачає, що кожен елемент вхідної вибірки може бути включений до семплу з певною константною ймовірністю. Зрозуміло, що в моєму випадку ця ймовірність рівна коефіцієнту зменшення вибірки (sc). Для створення вибірки я спочатку заповнюю масив індексів, які будуть обрані. Його розмір рівний розміру початкової тренувальної вибірки (800), а елементи належать множині {0, 1}. 1 із імовірністю p=sc, інакше 0. Далі я обираю лише ті записи, індекс яких 1.

```
for sc in sampling_coefficients:
    print(f'\t\t {sc}')
    iter_res = [0] * (2 * len(models) + 1)
    sample_indices = np.random.binomial(n=1, p=sc, size=X_train.shape[0])
    # print(sum(sample_indices))
    X_sample, y_sample = X_train[sample_indices==1], y_train[sample_indices==1]
    iter_res[0] = X_sample.shape[0]
    print(f'Train dataset size: {X_sample.shape[0]}')
    i = -1
    for model in models:
        i += 2
        model.fit(X_sample, y_sample)
        y_pred = model.predict(X_test)
        # print(f'Model: \t{model} accuracy: {accuracy_score(y_test, y_pred)}, f
        iter_res[i] = accuracy_score(y_test, y_pred)
        iter_res[i+1] = f1_score(y_test, y_pred, average="weighted")
    print(iter_res)
    results.append(iter_res)
print(results)
```

Систематичний відбір

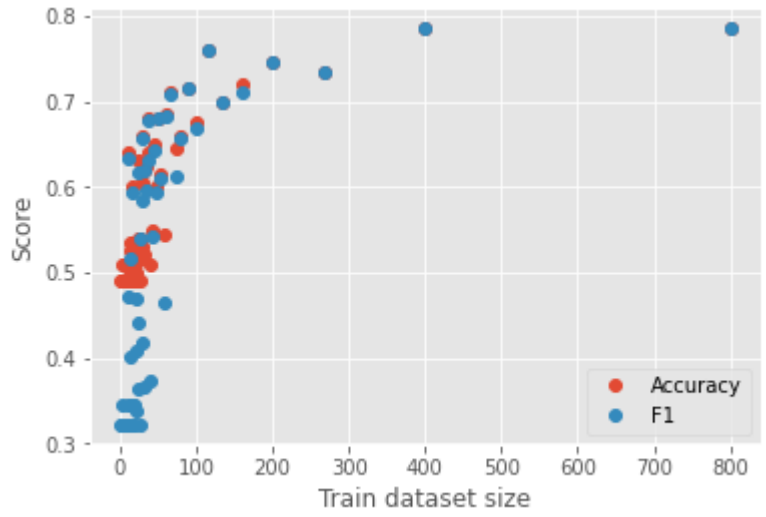
Вхідний датасет розбивається на a однакових інтервалів. У цьому інтервалі випадково рівноймовірно обирається індекс r Є [0, a) при індексації з 0. У зменшену вибірку потрапляють лише дані під індексами [r + a*k], де k Є {N (натуральні числа з 0), які не призведуть до виходу за межі масива}.

У цьому відборі я перебрав усі можливі значення інтервалів a. Щоб відфільтрувати дані, я заповнюю масив із нулів довжини a, випадковим чином присвоюю якомусь індексу значення 1, та дублюю отриманий масив, допоки він буде рівним довжині початкового тренувального набору даних. По цьому масиву фільтрую вхідний датасет.

```
results = []
old_sample_lengths = set()
for a in range(1, X_train.shape[0]):
    r = np.random.randint(0, a)  # init index
    sample_indices = np.zeros(a)
    sample_indices[r] = 1
    sample_indices = np.resize(sample_indices, X_train.shape[0])
    X_sample, y_sample = X_train[sample_indices==1], y_train[sample_indices==1]

    if X_sample.shape[0] in old_sample_lengths: continue
    old_sample_lengths.add(X_sample.shape[0])

    print(f'\t\t {X_sample.shape[0] / X_train.shape[0]}')
    print(f'{a=}, {r=}')
    print(f'Train dataset size: {X_sample.shape[0]}')
    iter_res = [0] * (2 * len(models) + 1)
    iter_res[0] = X_sample.shape[0]
    i = -1
    for model in models:
        i += 2
        model.fit(X_sample, y_sample)
        y_pred = model.predict(X_test)
        # print(f'Model: \t{model} accuracy: {accuracy_score(y_test, y_pred)}, f
        iter_res[i] = accuracy_score(y_test, y_pred)
        iter_res[i+1] = f1_score(y_test, y_pred, average="weighted")
    print(iter_res)
    results.append(iter_res)
print(results)
```



Дані на графіку зміщені ліворуч через особливість відбору (вже при a=2 розмір вибірки зменшується вдвічі, тому мало даних для великих розмірів отриманої вибірки).

Використання learning curve (3)

sklearn.model_selection.learning_curve використовується для пошуку оптимального співвідношення train/test set. Розміри тренувальної вибірки, які хочеться перевірити, задаються списком у параметрі train_sizes як частина від усього датасету. На test set виділяється всі інші дані. Також цей засіб виконує крос-валідацію для пошуку найкращого результату.

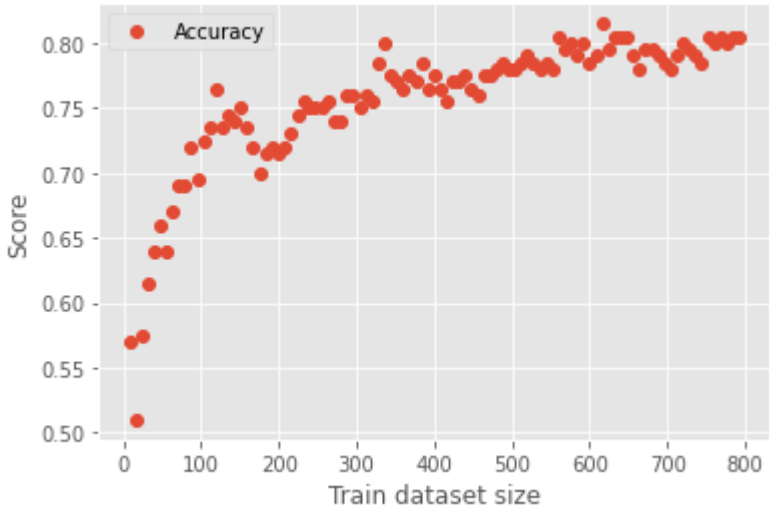
```python
from sklearn.model_selection import learning_curve

train_size_abs, train_scores, test_scores = learning_curve(
    sklearn.naive_bayes.GaussianNB(), X, y, train_sizes=[0.01, 0.1, 0.3, 0.6, 0.9],
    scoring='accuracy'
)
print('train_size', *train_size_abs, sep='\n')
test_scores
```

Отримані результати точності:

```
train_size
8           array([[0.5  , 0.57 , 0.56 , 0.505, 0.525],
80                 [0.67 , 0.7  , 0.735, 0.725, 0.705],
240                [0.725, 0.745, 0.74 , 0.735, 0.725],
480                [0.785, 0.78 , 0.79 , 0.82 , 0.755],
720                [0.795, 0.8  , 0.79 , 0.835, 0.725]])
```

Для всіх коефіцієнтів зменшення, які розглядалися в цій роботі, отримали такий графік:



Бачимо плавне зниження точності при зменшенні розміру датасета, як і при використанні інших методів відбору.

Висновки (4)

Загалом, зменшення розміру вибірки призводить до погіршення точності класифікації. Експерименти показують, що при деякому зменшенні розміру точність страждає не сильно. Це дає змогу зменшити розміри вибірки без значної втрати якості моделі, проте зі збереженням ресурсів і часу навчання.

Те, що на реальних даних точність падала лише при дуже малих розмірах вибірки, можна пояснити простотою класифікації даних датасетів. Вони мають добре розділені класи по фічах, що дає змогу побудувати гарну дискримінанту функцію навіть на малій вибірці.

Код роботи доступний за посиланням:

GitHub або Colab