Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №4

з дисципліни
“Програмування інтелектуальних інформаційних систем”

Виконала ІП-21 Скрипець О. О.

Перевірив Баришич Л. М.

Київ 2024

Завдання

1 Потюнити параметри за цим туторіалом. Порівняти з дефолтними. Зрозуміти роботу алгоритмічного підбору параметрів.

https://www.kaggle.com/code/shreayan98c/hyperparameter-tuning-tutorial

2 Зробити ансамблі і бустинг за цим туторіалом. Пояснити відмінності. Порівняти з просто тюнингом. Пояснити коли і що використовувати.

https://www.kaggle.com/code/pavansanagapati/ensemble-learning-techniques-tutorial

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
from sklearn.ensemble import VotingClassifier, AdaBoostClassifier
```

Алгоритмічний підбір параметрів

```
from google.colab import drive
drive.mount('/content/drive')
data = "/content/drive/My Drive/data/data.csv"
data = pd.read_csv(data)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Підготовка даних

```
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poi
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 33 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
```

```
memory usage: 146.8+ KB
```

Визначення цільової змінної

```
data['diagnosis'].value_counts()
```

diagnosis	count
B	357
M	212

dtype: int64

Прибирання непотрібних стовпців

```
data.drop(['Unnamed: 32', 'id'], axis=1, inplace=True)
```

розбиття даних на навчальні та тестові набори

```
X = data.drop(['diagnosis'], axis=1)
y = data['diagnosis']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
print("Size of training set:", X_train.shape)
print("Size of test set:", X_test.shape)
```

```
Size of training set: (398, 30)
Size of test set: (171, 30)
```

Метод деревв прийняття рішень

```
from sklearn.tree import DecisionTreeClassifier
clf1 = DecisionTreeClassifier(random_state=1)
clf1.fit(X_train, y_train)
y_pred = clf1.predict(X_test)
print("Accuracy of the model: {:.3f}".format(metrics.accuracy_score(y_test, y_pred)))
```

```
Accuracy of the model: 0.930
```

Оптимізація гіперпараметрів

```
parameters = {
    'max_features': ['log2', 'sqrt', 'auto'],
    'criterion': ['entropy', 'gini'],
    'max_depth': [2, 3, 5, 10, 50],
    'min_samples_split': [2, 3, 5, 10, 100],
    'min_samples_leaf': [1, 5, 8, 10],
    'random_state': [1]
}
```

```
clf2 = DecisionTreeClassifier()
grid_obj = GridSearchCV(clf2, parameters)
grid_obj = grid_obj.fit(X_train, y_train)
clf2 = grid_obj.best_estimator_
clf2.fit(X_train, y_train)
y_pred2 = clf2.predict(X_test)
print('Accuracy of best model: {:.3f}'.format(metrics.accuracy_score(y_test, y_pred2)))
```

```
Accuracy of best model: 0.901
```

Виведення різниць у параметрах між дефолтною і кращою моделлю

```
params1 = clf1.get_params()
params2 = clf2.get_params()
diff_params = {k: (params1[k], params2[k]) for k in params1 if k in params2 and params1[k] != params2[k]}
print("Different parameters:")
for param, values in diff_params.items():
    print(f"{param}: {values[0]} (default) vs {values[1]} (best)")
```

```
Different parameters:
criterion: gini (default) vs entropy (best)
max_depth: None (default) vs 10 (best)
max_features: None (default) vs log2 (best)
min_samples_split: 2 (default) vs 3 (best)
```

Метод k найближчих сусідів

```
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier()
knn1.fit(X_train, y_train)
y_pred1 = knn1.predict(X_test)
print('Accuracy of default model: {:.3f}'.format(metrics.accuracy_score(y_test, y_pred1)))
```

```
Accuracy of default model: 0.930
```

```
parameters = {
    'n_neighbors': [3, 4, 5, 10],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30, 50]
}

grid_obj = GridSearchCV(knn1, parameters)
grid_obj = grid_obj.fit(X_train, y_train)
knn2 = grid_obj.best_estimator_
knn2.fit(X_train, y_train)
y_pred2 = knn2.predict(X_test)
print('Accuracy of best model: {:.3f}'.format(metrics.accuracy_score(y_test, y_pred2)))
```

```
Accuracy of best model: 0.930
```

```
params1 = knn1.get_params()
params2 = knn2.get_params()
diff_params = {k: (params1[k], params2[k]) for k in params1 if k in params2 and params1[k] != params2[k]}
print("Different parameters:")
for param, values in diff_params.items():
    print(f"{param}: {values[0]} (default) vs {values[1]} (best)")
```

```
Different parameters:
leaf_size: 30 (default) vs 10 (best)
n_neighbors: 5 (default) vs 10 (best)
```

Метод опорних векторів

Оптимізація гіперпараметрів для SVM

```
from sklearn import svm
svc1 = svm.SVC(random_state=1)
svc1.fit(X_train, y_train)
y_pred1 = svc1.predict(X_test)
print('Accuracy of default model: {:.3f}'.format(metrics.accuracy_score(y_test, y_pred1)))
```

```
Accuracy of default model: 0.918
```

```
parameters = [
    {'C': [1, 10, 100], 'kernel': ['linear'], 'random_state': [1]},
    {'C': [1, 10, 100], 'gamma': ['0.001, 0.0001'],  'kernel': ['rbf'], 'random_state': [1]}
]
grid_obj = GridSearchCV(svm.SVC(), parameters)
grid_obj = grid_obj.fit(X_train, y_train)
svc2 = grid_obj.best_estimator_
svc2.fit(X_train, y_train)
y_pred2 = svc2.predict(X_test)
print('Accuracy of best model: {:.3f}'.format(metrics.accuracy_score(y_test, y_pred2)))
```

```
Accuracy of best model: 0.953
```

```
params1 = svc1.get_params()
params2 = svc2.get_params()
diff_params = {k: (params1[k], params2[k]) for k in params1 if k in params2 and params1[k] != params2[k]}
print("Different parameters:")
for param, values in diff_params.items():
```

```
    print(f"{param}: {values[0]} (default) vs {values[1]} (best)")
```

```
Different parameters:
kernel: rbf (default) vs linear (best)
```

Використання ансамблів та бустингу

Створення ансамблю моделей

```
estimator = []
estimator.append(('DTC', clf2))
params2['probability']=True
estimator.append(('SVC', svm.SVC(**params2)))
estimator.append(('KNN', knn2))
```

Voting Classifier з жорстким голосуванням (hard voting)

```
hard_voting = VotingClassifier(estimators=estimator, voting='hard')
hard_voting.fit(X_train, y_train)
y_pred = hard_voting.predict(X_test)
score = metrics.accuracy_score(y_test, y_pred)
print("Hard Voting Score: {:.3f}".format(score))
```

```
Hard Voting Score: 0.959
```

Voting Classifier з м'яким голосуванням (soft voting)

```
soft_voting = VotingClassifier(estimators=estimator, voting='soft')
soft_voting.fit(X_train, y_train)
y_pred = soft_voting.predict(X_test)
score = metrics.accuracy_score(y_test, y_pred)
print("Soft Voting Score: {:.3f}".format(score))
```

```
Soft Voting Score: 0.930
```

Використання AdaBoost для підвищення точності Decision Tree Classifier

```
dtc_common = DecisionTreeClassifier(random_state=2)
dtc_boosted = AdaBoostClassifier(
    dtc_common,
    n_estimators=300,
    random_state=2
)

dtc_common.fit(X_train, y_train)
dtc_boosted.fit(X_train, y_train)

y_pred1 = dtc_common.predict(X_test)
y_pred2 = dtc_boosted.predict(X_test)

print("Common DTC: {:.3f}".format(metrics.accuracy_score(y_test, y_pred1)))
print("Boosted DTC: {:.3f}".format(metrics.accuracy_score(y_test, y_pred2)))
```

```
Common DTC: 0.953
Boosted DTC: 0.942
```

Висновки

Ансамблі комбінують прогнози декількох моделей для підвищення ефективності загальної моделі. Існує кілька основних підходів до голосування в ансамблях:

- **Жорстке голосування (Hard Voting):** В ансамблі кожна модель пропонує свій прогноз, і перемагає той, хто набрав більшість голосів. Наприклад, якщо більшість моделей вказує на клас A, то A і стає вибором ансамблю.
- **М'яке голосування (Soft Voting):** Моделі надають ймовірності для кожного з класів, які потім усереднюються, і клас з найвищим середнім значенням ймовірності вибирається як прогноз.
- **Адаптивний бустинг (AdaBoost):** Ця техніка бустингу послідовно додає нові моделі, акцентуючи на помилках, зроблених попередніми моделями. Кожна нова модель спрямована на виправлення цих помилок, що сприяє зростанню точності

ансамблю.

Аналіз результатів показав: ансамблі, налаштовані з урахуванням тонкого настроювання параметрів, показали вищу продуктивність порівняно з індивідуальними моделями. У той час як моделі без тюнінгу демонстрували лише прийнятні результати.

Оптимальні застосування:

- **Ансамблі:** Ефективні для зниження варіативності і забезпечення стабільності передбачень, особливо з великими даними, зменшуючи ризик перенавчання. Вони надзвичайно корисні для комбінування кількох ефективних моделей.
- **Бустинг:** Ідеально підходить для мінімізації помилок і покращення точності, особливо при використанні незбалансованих даних. AdaBoost добре функціонує, коли потрібно покращити продуктивність слабших моделей за допомогою навчання на помилках.
- **Тюнінг:** Необхідний для швидкої оптимізації моделей для досягнення максимальної продуктивності. Такий підхід дозволяє детально налаштувати гіперпараметри для підвищення ефективності, як в ансамблях, так і в бустингу, але особливо важливий для точного налаштування окремих моделей.