Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И

ОПТИКИ

Факультет систем управления и робототехники

Отчет по практической работе №1 «Классификация»

по дисциплине «Прикладной искусственный интеллект»

Выполнил: студент гр. **R32362**

Михин Н. С.

Преподаватель: Евстафьев О. А.,

Санкт-Петербург 2022

1) **Цель работы:** познакомиться с основами sklearn. Обучить несколько моделей, сравнить результаты при изменении параметров, а также сравнить результаты работы моделей между собой.
2) **Ход работы**

Импорт библиотек

Первым делом импортируем необходимые библиотеки для работы с данными:

```
import os
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import RidgeClassifier
import matplotlib.pyplot as plt
target_names = ['Walking', 'Walking Upstairs', 'Walking Downstairs', 'Sitting
', 'Standing', 'Laying']
```

Считываем набор данных

В прикладных задачах машинного обучения очень важен процесс извлечения признаков (feature extraction), в ходе которого данные интерпретируются в информативные признаки. Также этот процесс может называться проектирование признаков (feature engineering), это весьма трудоемкая и творческая задача. В рамках работы мы опустим эту часть и воспользуемся предобработанными данными.

```
def read_data(path, filename):
    return pd.read_csv(os.path.join(path, filename))

df = read_data('/data/notebook_files', 'train.csv')
df.head()
```

Теперь, загрузим полный набор данных и сохранить его под следующими четырьмя переменными:

- train_X: признаки, используемые для обучения модели
- train_y: метки, используемые для обучения модели
- test_X: признаки, используемые для проверки модели
- test_y: метки, используемые для проверки модели

```
def load_dataset(label_dict):
    train_X = read_data('/data/notebook_files', 'train.csv').values[:,:-2]
    train_y = read_data('/data/notebook_files', 'train.csv')['Activity']
    train_y = train_y.map(label_dict).values
    test_X = read_data('/data/notebook_files', 'test.csv').values[:,:-2]
    test_y = read_data('/data/notebook_files', 'test.csv')
    test_y = test_y['Activity'].map(label_dict).values
    return(train_X, train_y, test_X, test_y)
label_dict = {'WALKING':0, 'WALKING_UPSTAIRS':1, 'WALKING_DOWNSTAIRS':2, 'SIT
```

```
TING':3, 'STANDING':4, 'LAYING':5}
train_X, train_y, test_X, test_y = load_dataset(label_dict)
```

Выбор модели

Импортируйте выбранную вами модель из библиотеки sklearn и инициализируйте её в объект model:

Также необходимо выбрать **несколько** моделей ML для сравнения полученных результатов.

```
# from sklearn.модель import название
# ниже замените None на инициализацию вашей модели
model = DecisionTreeClassifier(min_samples_split=2) # значение по умолчанию
```

Обучение модели

Обучите модель, используя признаки из обучающего набора (train_X) и метки в качестве базовой истины (train_y).

```
model.fit(train_X, train_y)
```

```
DecisionTreeClassifier()
```

Оценка модели

Используйте обученную модель для прогнозирования активности движения, используя признаки из тестового набора (test_X). Прогнозы сохраните в списке yhat.

```
yhat = model.predict(test_X)
yhat
```

Выведите отчет о классификации, сравнив предсказания (yhat) с базовой истиной (test_y).

```
print(classification_report(test_y, yhat, target_names=target_names))
```

```
                    precision    recall  f1-score   support

           Walking       0.82      0.91      0.86       496
  Walking Upstairs       0.82      0.76      0.79       471
Walking Downstairs       0.86      0.82      0.84       420
           Sitting       0.85      0.78      0.81       491
          Standing       0.81      0.87      0.84       532
            Laying       1.00      1.00      1.00       537

          accuracy                           0.86      2947
         macro avg       0.86      0.86      0.86      2947
      weighted avg       0.86      0.86      0.86      2947
```

Попробуем несколько гиперпараметров и моделей

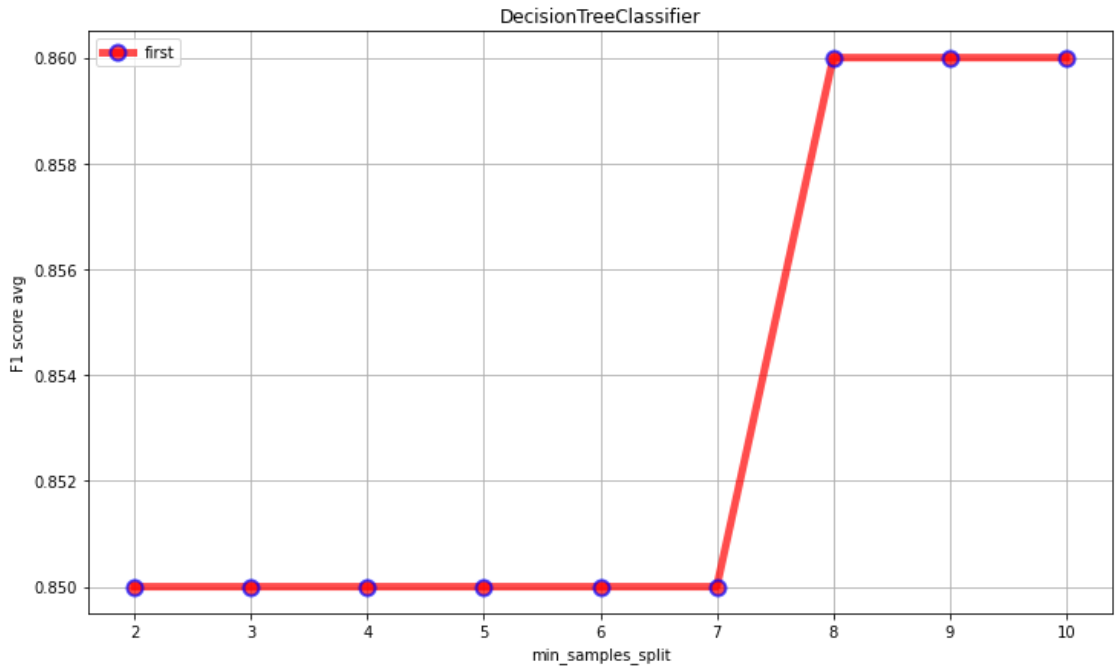
Подстановка нескольких разных гиперпараметров в исходный тип модели. Для сравнения будем использовать параметр F1, так как он в себе сразу же обобщает precision и recall. Чем ближе F1 к 1, тем лучше мы подобрали параметры. Для упрощения оценки будет браться среднее арифметическое по всем значениям F1 категорий.

Начнем менять параметр min_samples_split, он отвечает за минимальное количество выборок, необходимое для разделения внутреннего узла.

```
parameters = []
f1_result = []
for min_samples_split in range(2, 11):
    model_test = DecisionTreeClassifier(min_samples_split=min_samples_split)
    model_test.fit(train_X, train_y)
    yhat_test = model_test.predict(test_X)
    report = classification_report(test_y, yhat_test, target_names=target_nam
es, output_dict=True)
    F1_list = [report.get(target_name).get('f1-score') for target_name in tar
get_names]
    F1_average = round(sum(F1_list) / len(F1_list), 2)
    parameters.append(min_samples_split)
    f1_result.append(F1_average)
    print(f'min_samples_split({min_samples_split}): {F1_average}')

plt.figure(figsize=(12, 7))
plt.plot(parameters, f1_result, 'o-r', alpha=0.7, label="first", lw=5, mec='b
', mew=2, ms=10)
plt.legend()
plt.xlabel('min_samples_split')
plt.ylabel('F1 score avg')
plt.title('DecisionTreeClassifier')
plt.grid(True)
```

```
min_samples_split(2): 0.85
min_samples_split(3): 0.85
min_samples_split(4): 0.85
min_samples_split(5): 0.85
min_samples_split(6): 0.85
min_samples_split(7): 0.85
min_samples_split(8): 0.86
min_samples_split(9): 0.86
min_samples_split(10): 0.86
```

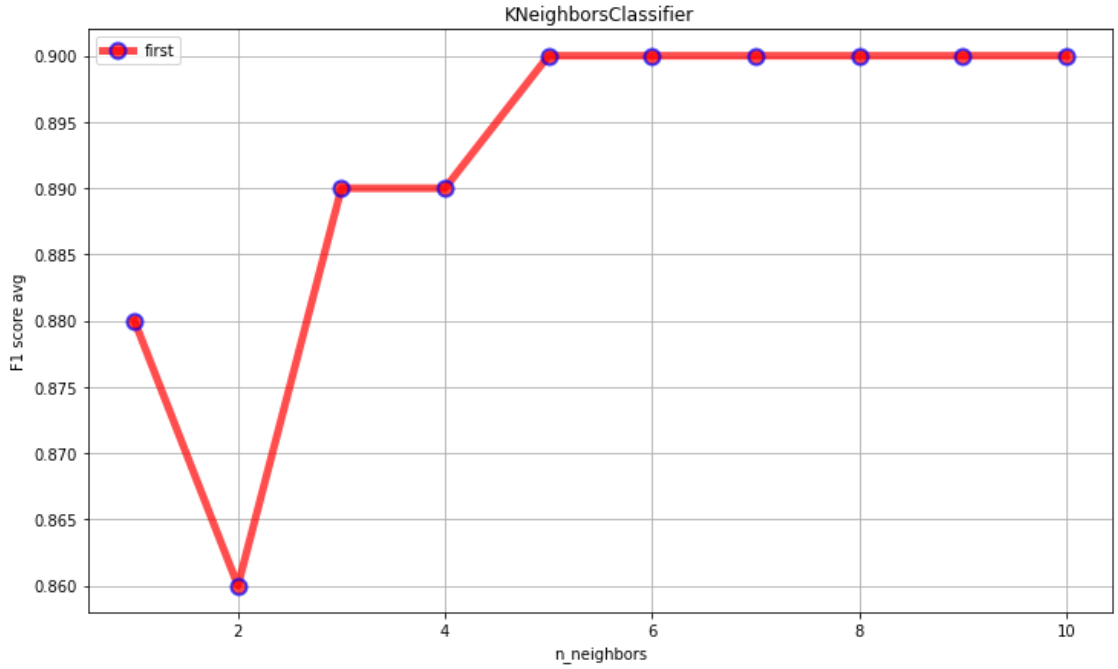


Теперь возьмем модель KNeighborsClassifier и поиграемся с ее параметром n_neigghbors(Количество соседей, используемых по умолчанию для kneighbors запросов, по умолчанию 5)

```
parameters = []
f1_result = []
for n_neighbors in range(1, 11):
    model_test = KNeighborsClassifier(n_neighbors=n_neighbors)
    model_test.fit(train_X, train_y)
    yhat_test = model_test.predict(test_X)
    report = classification_report(test_y, yhat_test, target_names=target_nam
es, output_dict=True)
    F1_list = [report.get(target_name).get('f1-score') for target_name in tar
get_names]
    F1_average = round(sum(F1_list) / len(F1_list), 2)
    parameters.append(n_neighbors)
    f1_result.append(F1_average)
    print(f'n_neighbors({n_neighbors}): {F1_average}')
plt.figure(figsize=(12, 7))
plt.plot(parameters, f1_result, 'o-r', alpha=0.7, label="first", lw=5, mec='b
', mew=2, ms=10)
plt.legend()
plt.xlabel('n_neighbors')
plt.title('KNeighborsClassifier')
plt.ylabel('F1 score avg')
plt.grid(True)
```

```
n_neighbors(1): 0.88
n_neighbors(2): 0.86
n_neighbors(3): 0.89
n_neighbors(4): 0.89
n_neighbors(5): 0.9
n_neighbors(6): 0.9
n_neighbors(7): 0.9
n_neighbors(8): 0.9
```

```
n_neighbors(9): 0.9
n_neighbors(10): 0.9
```

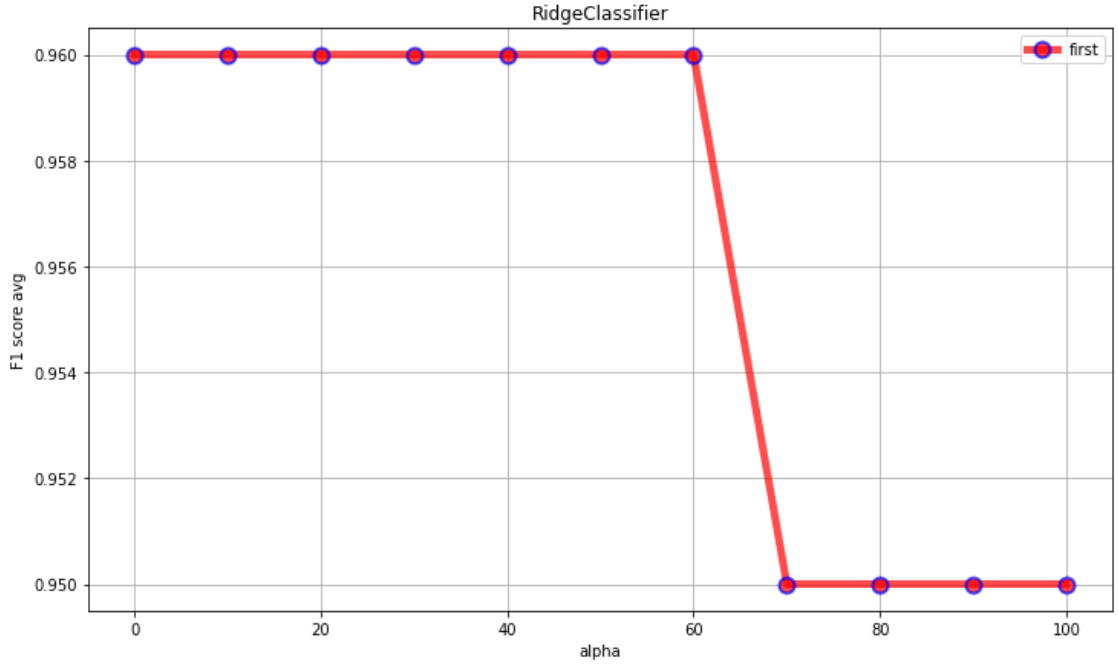


Теперь используем RidgeClassifier и поизменяем ее параметр alpha (сила регуляризации; должен быть положительным числом с плавающей запятой. Регуляризация улучшает обусловленность задачи и уменьшает дисперсию оценок. Большие значения определяют более сильную регуляризацию)

```
parameters = []
f1_result = []
for alpha in range(0, 101, 10):
    model_test = RidgeClassifier(alpha=alpha)
    model_test.fit(train_X, train_y)
    yhat_test = model_test.predict(test_X)
    report = classification_report(test_y, yhat_test, target_names=target_nam
es, output_dict=True)
    F1_list = [report.get(target_name).get('f1-score') for target_name in tar
get_names]
    F1_average = round(sum(F1_list) / len(F1_list), 2)
    parameters.append(alpha)
    f1_result.append(F1_average)
    print(f'max_iter_predict({alpha}): {F1_average}')
plt.figure(figsize=(12, 7))
plt.plot(parameters, f1_result, 'o-r', alpha=0.7, label="first", lw=5, mec='b
', mew=2, ms=10)
plt.legend()
plt.xlabel('alpha')
plt.title('RidgeClassifier')
plt.ylabel('F1 score avg')
plt.grid(True)
```

```
max_iter_predict(0): 0.96
max_iter_predict(10): 0.96
max_iter_predict(20): 0.96
max_iter_predict(30): 0.96
```

```
max_iter_predict(40): 0.96
max_iter_predict(50): 0.96
max_iter_predict(60): 0.96
max_iter_predict(70): 0.95
max_iter_predict(80): 0.95
max_iter_predict(90): 0.95
max_iter_predict(100): 0.95
```



Теперь получим отчеты по каждой модели при значениях по умолчанию

```
model_DecisionTreeClassifier = DecisionTreeClassifier()
model_DecisionTreeClassifier.fit(train_X, train_y)
yhat_DecisionTreeClassifier = model_DecisionTreeClassifier.predict(test_X)
print('DecisionTreeClassifier')
print(classification_report(test_y, yhat_DecisionTreeClassifier, target_names
=target_names, output_dict=False))

model_KNeighborsClassifier = KNeighborsClassifier()
model_KNeighborsClassifier.fit(train_X, train_y)
yhat_KNeighborsClassifier = model_KNeighborsClassifier.predict(test_X)
print('KNeighborsClassifier')
print(classification_report(test_y, yhat_KNeighborsClassifier, target_names=t
arget_names, output_dict=False))

model_RidgeClassifier = RidgeClassifier()
model_RidgeClassifier.fit(train_X, train_y)
yhat_RidgeClassifier = model_RidgeClassifier.predict(test_X)
print('RidgeClassifier')
print(classification_report(test_y, yhat_RidgeClassifier, target_names=target
_names, output_dict=False))
```

```
      DecisionTreeClassifier
                    precision    recall  f1-score   support

           Walking       0.84      0.91      0.87       496
```

Walking Upstairs	0.83	0.77	0.80	471
Walking Downstairs	0.85	0.83	0.84	420
Sitting	0.84	0.77	0.80	491
Standing	0.80	0.86	0.83	532
Laying	1.00	1.00	1.00	537
accuracy			0.86	2947
macro avg	0.86	0.86	0.86	2947
weighted avg	0.86	0.86	0.86	2947

KNeighborsClassifier

	precision	recall	f1-score	support
Walking	0.85	0.98	0.91	496
Walking Upstairs	0.89	0.90	0.90	471
Walking Downstairs	0.95	0.79	0.86	420
Sitting	0.91	0.79	0.85	491
Standing	0.83	0.93	0.88	532
Laying	1.00	0.99	1.00	537
accuracy			0.90	2947
macro avg	0.91	0.90	0.90	2947
weighted avg	0.91	0.90	0.90	2947

RidgeClassifier

	precision	recall	f1-score	support
Walking	0.97	0.99	0.98	496
Walking Upstairs	0.97	0.98	0.97	471
Walking Downstairs	1.00	0.98	0.99	420
Sitting	0.96	0.87	0.91	491
Standing	0.86	0.96	0.91	532
Laying	1.00	0.96	0.98	537
accuracy			0.96	2947
macro avg	0.96	0.96	0.96	2947
weighted avg	0.96	0.96	0.96	2947

3) Заключение

Изначально я хотел использовать GaussianProcessClassifier и так как обучение данной модели занимает довольно большое время, то было принято решение по использованию asyncio. Но, к сожалению, среда разработки Datalore не позволила мне запустить сразу несколько задач и поэтому было принято решение использовать более быструю в обучении `DecisionTreeClassifier`.

4) Ответы на вопросы

В чем разница между показателями precision и recall?

-Precision (точность) показывает: сколько реальных объектов класса среди всех тех, что классификатор отнес к этому классу. Однако, у этой метрики есть явный минус - она никак не учитывает количество объектов данного класса, которые классификатор в него “забыл” включить. Для этого и нужны другие метрики (чем больше значение Precision, тем лучше).

-Значение recall (полнота) показывает: какую долю из общих объектов класса составляют найденные объекты. Фактически, recall учитывает слабое место precision (чем больше значение, тем лучше).

Что такое показатель F1? — это среднее гармоническое значение между precision и recall

https://datalore.jetbrains.com/notebook/cQefsgA4s8j85f5VnSAiIS/KH6zCxWxjYWKfc9v93fsWp/