Московский Государственный Технический Университет
им. Н.Э. Баумана

Отчет по лабораторной работе №4

по курсу
Технологии Машинного Обучения

Выполнила:
Костян Алина
ИУ5-63

_____

Проверил:
Гапанюк Ю.Е.

_____

Москва, 2019

Задание

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью трех подходящих для задачи метрик.
5. Постройте модель и оцените качество модели с использованием кросс-валидации. Проведите эксперименты с тремя различными стратегиями кросс-валидации.
6. Произведите подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.
7. Повторите пункт 4 для найденного оптимального значения гиперпараметра K. Сравните качество полученной модели с качеством модели, полученной в пункте 4.
8. Постройте кривые обучения и валидации.

Код и результаты выполнения

1. Подключим библиотеки:

```
import pandas as pd
from enum import Enum
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from operator import itemgetter
import matplotlib.pyplot as plt
import numpy as np
```

2. Подготовим набор данных к работе

Проверим наличие пропущенных значениений

```
nd = data.columns[data.isnull().any()]

print(len(nd))
```

```
0
```

Как мы видим пропущенных значений нет

Проверим наличие категориальных значений

```
cats = [col for col in data.columns if data[col].dtype=="object"]
print(len(cats))
```

```
12
```

Имеем 12 столбцов с категориальными признаками

Заменим категориальные признаки числовыми используя Label encoding

```
le = LabelEncoder()
for col in cats:
    data[col]=le.fit_transform(data[col])
```

```
newcats = [col for col in data.columns if data[col].dtype=="object"]
print(len(newcats))
```

```
0
```

3. Разделим выборку на обучающую и тестовую

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
# Размер тренировочной выборки
print(X_train.shape, y_train.shape)
# Размер тестовой выборки
print(X_test.shape, y_test.shape)
```

```
(6980, 55) (6980,)
(1746, 55) (1746,)
```

4. Построение и оценка модели

Обучим модель используя метод k близжайших значений

```
KNN = KNeighborsRegressor()
KNN.fit(X_train,y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform')
```

```
a= KNN.predict(X_test)
```

Проверим модель при помощи метрик R2, MSE, MedAE

```
R2(y_test, a)
```

```
0.35347216945585735
```

```
MSE(y_test, a)
```

```
98227.06495687284
```

```
MedAE(y_test, a)
```

```
87.82000000000001
```

Обучим модель при помощи кросс валидации тремя стратегиями и подберем гиперпараметры

```
ll = [1,2,3,4,5,10,25,40,50,100,250,500,1000]
```

1. KFold

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,cv=KFold(n_splits=5), scoring = 'r2')
print(np.mean(scores))
```

```
-0.5471605374343553
```

```
scores = -1*cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,cv=KFold(n_splits=5), scoring = 'neg_mean_squared_e
print(np.mean(scores))
```

```
196526.4650782841
```

```
scores = -1*cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,cv=KFold(n_splits=5), scoring = 'neg_median_absolut
print(np.mean(scores))
```

```
163.50959999999998
```

```
random_search_kFold = GridSearchCV(estimator= KNeighborsRegressor(),
                           param_grid= {'n_neighbors': ll},
                           scoring= 'r2',
                           cv= KFold(n_splits=5))
```

```
random_search_kFold.fit(x,y)
```

```
GridSearchCV(cv=KFold(n_splits=5, random_state=None, shuffle=False),
       error_score='raise-deprecating',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 10, 25, 40, 50, 100, 250, 500, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_squared_error', verbose=0)
```

```
random_search_kFold.best_params_
```

```
{'n_neighbors': 1000}
```

```
random_search_kFold = GridSearchCV(estimator= KNeighborsRegressor(),
                           param_grid= {'n_neighbors': ll},
                           scoring= 'neg_median_absolute_error',
                           cv= KFold(n_splits=5))
```

```
random_search_kFold.fit(x,y)
```

```
GridSearchCV(cv=KFold(n_splits=5, random_state=None, shuffle=False),
       error_score='raise-deprecating',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 10, 25, 40, 50, 100, 250, 500, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_median_absolute_error', verbose=0)
```

```
random_search_kFold
```

```
{'n_neighbors': 10}
```

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=1000),x, y,
                         cv=KFold(n_splits=5), scoring = 'r2')
print(np.mean(scores))
```

```
-0.06260425742921458
```

```
scores = -1*cross_val_score(KNeighborsRegressor(n_neighbors=1000),x, y,
                            cv=KFold(n_splits=5), scoring = 'neg_mean_squared_error')
print(np.mean(scores))
```

```
136522.2623284804
```

```
scores = -1*cross_val_score(KNeighborsRegressor(n_neighbors=10),x, y,
                            cv=KFold(n_splits=5), scoring = 'neg_median_absolute_error')
print(np.mean(scores))
```

```
163.45470000000003
```

```
random_search_Shuffled.best_params_
```

```
{'n_neighbors': 3}
```

```
random_search_Shuffled = GridSearchCV(estimator= KNeighborsRegressor(),
                           param_grid= {'n_neighbors': ll},
                           scoring= 'neg_mean_squared_error',
                           cv= ShuffleSplit(n_splits=5, test_size=0.25))
```

```
random_search_Shuffled.fit(x,y)
```

```
GridSearchCV(cv=ShuffleSplit(n_splits=5, random_state=None, test_size=0.25, train_size=None),
       error_score='raise-deprecating',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 10, 25, 40, 50, 100, 250, 500, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_squared_error', verbose=0)
```

```
random_search_Shuffled.best_params_
```

```
{'n_neighbors': 3}
```

```
random_search_Shuffled = GridSearchCV(estimator= KNeighborsRegressor(),
                           param_grid= {'n_neighbors': ll},
                           scoring= 'neg_median_absolute_error',
                           cv= ShuffleSplit(n_splits=5, test_size=0.25))
```

```
random_search_Shuffled.fit(x,y)
```

2. ShuffleSplit

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,
                         cv=ShuffleSplit(n_splits=5, test_size=0.25), scoring = 'r2')
print(np.mean(scores))
```

```
0.39628696314739453
```

```
scores = -1 * cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,
                              cv=ShuffleSplit(n_splits=5, test_size=0.25),
                              scoring = 'neg_mean_squared_error')
print(np.mean(scores))
```

```
82540.63936477728
```

```
scores = -1* cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,
                             cv=ShuffleSplit(n_splits=5, test_size=0.25),
                             scoring = 'neg_median_absolute_error')
print(np.mean(scores))
```

```
88.6202
```

```
random_search_Shuffled = GridSearchCV(estimator= KNeighborsRegressor(),
                           param_grid= {'n_neighbors': ll},
                           scoring= 'r2',
                           cv= ShuffleSplit(n_splits=5, test_size=0.25))
```

```
random_search_Shuffled.fit(x,y)
```

```
random_search_Shuffled.best_params_
```

```
{'n_neighbors': 3}
```

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=3),x, y,
                         cv=ShuffleSplit(n_splits=5, test_size=0.25),
                         scoring = 'r2')
print(np.mean(scores))
```

```
0.39840144338554095
```

```
scores = -1 * cross_val_score(KNeighborsRegressor(n_neighbors=3),x, y,
                              cv=ShuffleSplit(n_splits=5, test_size=0.25),
                              scoring = 'neg_mean_squared_error')
print(np.mean(scores))
```

```
76819.15214048987
```

```
scores = -1* cross_val_score(KNeighborsRegressor(n_neighbors=3),x, y,
                             cv=ShuffleSplit(n_splits=5, test_size=0.25),
                             scoring = 'neg_median_absolute_error')
print(np.mean(scores))
```

```
81.00933333333333
```

3. RepeatedKFold

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,
                         cv=RepeatedKFold(n_splits=5, n_repeats=5),
                         scoring = 'r2')
print(np.mean(scores))
```

```
0.38647630284365564
```

```
scores = -1 * cross_val_score(KNeighborsRegressor(n_neighbors=5),x, y,
                              cv=RepeatedKFold(n_splits=5, n_repeats=5),
                              scoring = 'neg_mean_squared_error')
print(np.mean(scores))
```

```
81875.4771649273
```

```
scores = -1* cross_val_score(KNeighborsRegressor(n_neighbors=5),x,
                             y,cv=RepeatedKFold(n_splits=5, n_repeats=5),
                             scoring = 'neg_median_absolute_error')
print(np.mean(scores))
```

```
88.27748
```

```
random_search_n_kFold = GridSearchCV(estimator= KNeighborsRegressor(),
                             param_grid= {'n_neighbors': ll},
                             scoring= 'r2',
                             cv= RepeatedKFold(n_splits=5, n_repeats=5))
```

```
random_search_n_kFold.fit(x, y)
```

```
random_search_n_kFold.best_params_
```

```
{'n_neighbors': 2}
```

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=3),x, y,
                         cv=RepeatedKFold(n_splits=5, n_repeats=5),
                         scoring = 'r2')
print(np.mean(scores))
```

```
0.40049027597379394
```

```
scores = -1 * cross_val_score(KNeighborsRegressor(n_neighbors=3),x, y,
                              cv=RepeatedKFold(n_splits=5, n_repeats=5),
                              scoring = 'neg_mean_squared_error')
print(np.mean(scores))
```

```
79892.63383489785
```

```
scores = -1* cross_val_score(KNeighborsRegressor(n_neighbors=2),x, y,
                             cv=RepeatedKFold(n_splits=5, n_repeats=5),
                             scoring = 'neg_median_absolute_error')
print(np.mean(scores))
```

```
80.9564
```

```
random_search_n_kFold.best_params_
```

```
{'n_neighbors': 3}
```

```
random_search_n_kFold = GridSearchCV(estimator= KNeighborsRegressor(),
                              param_grid= {'n_neighbors': ll},
                              scoring= 'neg_mean_squared_error',
                              cv= RepeatedKFold(n_splits=5, n_repeats=5))
```

```
random_search_n_kFold.fit(x, y)
```

```
GridSearchCV(cv=<sklearn.model_selection._split.RepeatedKFold object at 0x1256aa908>,
       error_score='raise-deprecating',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 10, 25, 40, 50, 100, 250, 500, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_squared_error', verbose=0)
```

```
random_search_n_kFold.best_params_
```

```
{'n_neighbors': 3}
```

```
random_search_n_kFold = GridSearchCV(estimator= KNeighborsRegressor(),
                              param_grid= {'n_neighbors': ll},
                              scoring= 'neg_median_absolute_error',
                              cv= RepeatedKFold(n_splits=5, n_repeats=5))
```

```
random_search_n_kFold.fit(x, y)
```

5. Кривая обучения и валидации