Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Отчет по лабораторной работе №3

«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

по дисциплине «Технологии машинного обучения»

Выполнил:
студент группы ИУ5-64Б
Стукалов И.Д.
10.05.2024

Проверил:
Гапанюк Ю.Е.

2024 г.

Задание

- Выберите набор данных (датасет) для решения задачи классификации или регрессии.
- В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
- С использованием метода train_test_split разделите выборку на обучающую и тестовую.
- Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
- Произведите подбор гиперпараметра K с использованием GridSearchCV и RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Используйте не менее двух стратегий кросс-валидации.
- Сравните метрики качества исходной и оптимальной моделей.

Текст программы

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.

```
[ ] from sklearn.datasets import load_iris
    iris = load_iris()
```

```
[ ] # признаки
    iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
[ ] # данные
    iris.data[0:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

```
[ ] print("рамер датасета ", len(iris.data))
```

```
рамер датасета  150
```

```
[ ] # целевые классы
    iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
[ ] import pandas as pd

    iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
    # iris_df['target'] = iris.target
    # iris_df['target_names'] = iris.target_names[iris.target]

    iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков

```
[ ] iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

Датасет не содержит пропусков или категориальных признаков

3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.

```
[ ] from sklearn.model_selection import train_test_split

    iris = load_iris()
    X = iris.data
    y = iris.target

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
```

4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.

Обучение модели

```
[ ] from sklearn.neighbors import KNeighborsClassifier

    # Обучение модели KNN с произвольно заданным K
    knn = KNeighborsClassifier(n_neighbors=3) # Пример значения K
    knn.fit(X_train, y_train)
    target1_1 = knn.predict(X_test)
    len(target1_1), target1_1
```

```
(75,
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 0, 2, 1, 0, 0, 1, 1, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,
```

4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.

Обучение модели

```
from sklearn.neighbors import KNeighborsClassifier

# Обучение модели KNN с произвольно заданным K
knn = KNeighborsClassifier(n_neighbors=3) # Пример значения K
knn.fit(X_train, y_train)
target1_1 = knn.predict(X_test)
len(target1_1), target1_1
```

```
(75,
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 0, 2, 1, 0, 0, 1, 1, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,
        1, 2, 0, 0, 0, 1, 0, 0, 2, 2, 2, 2, 1, 1, 2, 1, 0, 2, 1, 0, 0, 2,
        0, 1, 2, 1, 1, 2, 1, 0, 1]))
```

Оценка качества модели

```
from sklearn.metrics import accuracy_score

# Оценка качества модели
accuracy = accuracy_score(y_test, target1_1)
print(f"Точность модели: {accuracy:.4f}")
```

```
Точность модели: 0.9200
```

5. Произведите подбор гиперпараметра K с использованием GridSearchCV и RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Используйте не менее двух стратегий кросс-валидации.

Подбор гиперпараметра K с использованием GridSearchCV

```
from sklearn.model_selection import GridSearchCV

# Определение диапазона значений K
k_range = list(range(1, 31))

# Создание сетки параметров
param_grid = dict(n_neighbors=k_range)

# Инициализация GridSearchCV
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
```

Подбор гиперпараметра K с использованием GridSearchCV

```python
from sklearn.model_selection import GridSearchCV

# Определение диапазона значений K
k_range = list(range(1, 31))

# Создание сетки параметров
param_grid = dict(n_neighbors=k_range)

# Инициализация GridSearchCV
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Обучение и поиск оптимального K
grid_search.fit(X_train, y_train)

# Вывод лучшего значения K и соответствующей точности
print(f"Лучшее значение K: {grid_search.best_params_['n_neighbors']}")
print(f"Точность с оптимальным K: {grid_search.best_score_}")
```

```
Лучшее значение K: 5
Точность с оптимальным K: 0.9733333333333334
```

Подбор гиперпараметра K с использованием RandomizedSearchCV

```python
from sklearn.model_selection import RandomizedSearchCV

# Определение распределения значений K
param_dist = dict(n_neighbors=k_range)

# Инициализация RandomizedSearchCV
rand_search = RandomizedSearchCV(knn, param_dist, cv=5, scoring='accuracy', n_iter=10, random_state=1)

# Обучение и поиск оптимального K
rand_search.fit(X_train, y_train)

# Вывод лучшего значения K и соответствующей точности
print(f"Лучшее значение K (RandomizedSearch): {rand_search.best_params_['n_neighbors']}")
print(f"Точность с оптимальным K (RandomizedSearch): {rand_search.best_score_}")
```

```
Лучшее значение K (RandomizedSearch): 18
Точность с оптимальным K (RandomizedSearch): 0.9733333333333334
```

Кросс-валидация

```python
from sklearn.model_selection import cross_val_score, cross_validate

scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                         iris.data, iris.target, cv=3)
```

Кросс-валидация

```
[ ] from sklearn.model_selection import cross_val_score, cross_validate

    scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                             iris.data, iris.target, cv=3)
    scores
```

```
array([0.96, 0.94, 0.94])
```

```
[ ] import numpy as np

    np.mean(scores)
```

```
0.9466666666666667
```

Используем cross_validate

```
scoring = {'precision': 'precision_weighted',
           'recall': 'recall_weighted',
           'f1': 'f1_weighted'}
scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        iris.data, iris.target, scoring=scoring,
                        cv=3, return_train_score=True)
scores
```

```
{'fit_time': array([0.00110984, 0.0006628 , 0.00064826]),
 'score_time': array([0.01089048, 0.00719643, 0.0069859 ]),
 'test_precision': array([0.96421053, 0.94088889, 0.94097222]),
 'train_precision': array([0.9725    , 0.98111111, 0.98114286]),
 'test_recall': array([0.96, 0.94, 0.94]),
 'train_recall': array([0.97, 0.98, 0.98]),
 'test_f1': array([0.95977778, 0.9398894 , 0.93994805]),
 'train_f1': array([0.96995987, 0.97997321, 0.97998162])}
```

Стратегии кросс-валидации

K-fold

```
[ ] from sklearn.model_selection import KFold, LeaveOneOut

    # K = 10
    X = range(10)
    kf = KFold(n_splits=3)
    for train, test in kf.split(X):
        print("%s %s" % (train, test))
```

```
[4 5 6 7 8 9] [0 1 2 3]
[0 1 2 3 7 8 9] [4 5 6]
[0 1 2 3 4 5 6] [7 8 9]
```

Стратегии кросс-валидации

K-fold

```
from sklearn.model_selection import KFold, LeaveOneOut

# K = 10
X = range(10)
kf = KFold(n_splits=3)
for train, test in kf.split(X):
    print("%s %s" % (train, test))
```

```
[4 5 6 7 8 9] [0 1 2 3]
[0 1 2 3 7 8 9] [4 5 6]
[0 1 2 3 4 5 6] [7 8 9]
```

Leave One Out

```
X = range(12)
# Эквивалент KFold(n_splits=n)
kf = LeaveOneOut()
for train, test in kf.split(X):
    print("%s %s" % (train, test))
```

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

Оптимизация гиперпараметров

Grid-search

```
n_range = np.array(range(5,55,5))
tuned_parameters = [{'n_neighbors': n_range)]
tuned_parameters
```

```
[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]
```

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy')
```

Оптимизация гиперпараметров

Grid-search

```
[ ] n_range = np.array(range(5,55,5))
    tuned_parameters = [{'n_neighbors': n_range}]
    tuned_parameters
```

```
[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]
```

```
[ ] %%time
    clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy')
    clf_gs.fit(X_train, y_train)
```

```
CPU times: user 277 ms, sys: 2.69 ms, total: 280 ms
Wall time: 201 ms
```

GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier

```
clf_gs.cv_results_
```

```
{'mean_fit_time': array([0.00072708, 0.00056467, 0.00064201, 0.00058045, 0.00054936,
        0.00049791, 0.00050979, 0.00050716, 0.00058784, 0.00063577]),
 'std_fit_time': array([2.16938405e-04, 1.16874723e-05, 1.05878295e-04, 2.92906127e-05,
        8.74499068e-06, 3.58816714e-05, 1.21358570e-05, 1.76232460e-05,
        6.79663754e-05, 1.05835680e-04]),
 'mean_score_time': array([0.00298152, 0.00225019, 0.00252705, 0.00228047, 0.00221081,
        0.00306082, 0.00287132, 0.00286403, 0.0047173 , 0.00383577]),
 'std_score_time': array([9.83910391e-04, 4.74765569e-05, 5.13155689e-04, 6.47679330e-05,
        5.47238732e-05, 4.18818213e-04, 9.82612490e-05, 4.94815001e-05,
        1.16938059e-03, 1.21738204e-03]),
 'param_n_neighbors': masked_array(data=[5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 5},
  {'n_neighbors': 10},
  {'n_neighbors': 15},
  {'n_neighbors': 20},
  {'n_neighbors': 25},
  {'n_neighbors': 30},
  {'n_neighbors': 35},
  {'n_neighbors': 40},
  {'n_neighbors': 45},
  {'n_neighbors': 50}],
 'split0_test_score': array([0.93333333, 0.93333333, 0.93333333, 1.        , 1.        ,
        0.93333333, 1.        , 0.66666667, 0.66666667, 0.66666667]),
 'split1_test_score': array([1.        , 1.        , 1.        , 1.        , 0.93333333,
        1.        , 0.93333333, 0.66666667, 0.66666667, 0.66666667]),
 'split2_test_score': array([1.        , 1.        , 0.93333333, 1.        , 0.93333333,
```

```
 'std_fit_time': array([2.16938405e-04, 1.16874723e-05, 1.05878295e-04, 2.92906127e-05,
        8.74499068e-06, 3.58816714e-05, 1.21358570e-05, 1.76232460e-05,
        6.79663754e-05, 1.05835680e-04]),
 'mean_score_time': array([0.00298152, 0.00225019, 0.00252705, 0.00228047, 0.00221081,
        0.00306082, 0.00287132, 0.00286403, 0.0047173 , 0.00383577]),
 'std_score_time': array([9.83910391e-04, 4.74765569e-05, 5.13155689e-04, 6.47679330e-05,
        5.47238732e-05, 4.18818213e-04, 9.82612490e-05, 4.94815001e-05,
        1.16938059e-03, 1.21738204e-03]),
 'param_n_neighbors': masked_array(data=[5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 5},
  {'n_neighbors': 10},
  {'n_neighbors': 15},
  {'n_neighbors': 20},
  {'n_neighbors': 25},
  {'n_neighbors': 30},
  {'n_neighbors': 35},
  {'n_neighbors': 40},
  {'n_neighbors': 45},
  {'n_neighbors': 50}],
 'split0_test_score': array([0.93333333, 0.93333333, 0.93333333, 1.        , 1.        ,
        0.93333333, 1.        , 0.66666667, 0.66666667, 0.66666667]),
 'split1_test_score': array([1.        , 1.        , 1.        , 1.        , 0.93333333,
        1.        , 0.93333333, 0.66666667, 0.66666667, 0.66666667]),
 'split2_test_score': array([1.        , 1.        , 0.93333333, 1.        , 0.93333333,
        0.86666667, 0.86666667, 0.66666667, 0.66666667, 0.66666667]),
 'split3_test_score': array([0.93333333, 0.93333333, 0.93333333, 0.93333333, 0.93333333,
        0.86666667, 0.86666667, 0.6       , 0.66666667, 0.6       ]),
 'split4_test_score': array([1.        , 1.        , 0.93333333, 0.93333333, 0.93333333,
        0.93333333, 0.93333333, 0.73333333, 0.53333333, 0.33333333]),
 'mean_test_score': array([0.97333333, 0.97333333, 0.94666667, 0.97333333, 0.94666667,
        0.92      , 0.92      , 0.66666667, 0.64      , 0.58666667]),
 'std_test_score': array([0.03265986, 0.03265986, 0.02666667, 0.03265986, 0.02666667,
        0.04988877, 0.04988877, 0.0421637 , 0.05333333, 0.12927146]),
 'rank_test_score': array([ 1,  1,  4,  1,  4,  6,  6,  8,  9, 10], dtype=int32)}
```

```
[ ] clf_gs.best_estimator_
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
[ ] clf_gs.best_score_
```

```
0.9733333333333334
```

```
[ ] clf_gs.best_params_
```

```
{'n_neighbors': 5}
```