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

по дисциплине «Технология машинного обучения»

Выполнил:
студент группы ИУ5-63Б Кузнецов В.А.
подпись, дата

Проверил:
Гапанюк Ю.Е.
подпись, дата

2024 г.

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
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 подходящих для задачи метрик.
5. Произведите подбор гиперпараметра K с использованием GridSearchCV и RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Используйте не менее двух стратегий кросс-валидации.
6. Сравните метрики качества исходной и оптимальной моделей.

Текст программы:

Основные характеристики датасета

sepal length - длина наружной доли околоцветника

sepal width - ширина наружной доли околоцветника

petal length - длина внутренней доли околоцветника

petal width - ширина внутренней доли околоцветника

target - тип ирисов (Iris setosa, Iris virginica, Iris versicolor)

Импорт библиотек

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, KFold, LeaveOneOut, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

RANDOM_STATE=125
```

Подготовка

```
iris = load_iris()
data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                    columns= iris['feature_names'] + ['target'])
```

```
data.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

```
data.isnull().sum()
```

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

```
data.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
from sklearn.preprocessing import MinMaxScaler

mmScaler = MinMaxScaler()

scaled_data = mmScaler.fit_transform(data)
scaled_data = pd.DataFrame(scaled_data, columns=data.columns)
scaled_data['target'] = data['target']


scaled_data.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	0.428704	0.440556	0.467458	0.458056	1.000000
std	0.230018	0.181611	0.299203	0.317599	0.819232
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.222222	0.333333	0.101695	0.083333	0.000000
50%	0.416667	0.416667	0.567797	0.500000	1.000000
75%	0.583333	0.541667	0.694915	0.708333	2.000000
max	1.000000	1.000000	1.000000	1.000000	2.000000

```
data = scaled_data
```

Пропусков в датасете нет.

Категориальный признак уже закодирован.

Разделение на выборки

```
X = data.iloc[:, :-1]
y = data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=RANDOM_STATE)
```

Обучение с константным K

```
K = 4
knn = KNeighborsClassifier(n_neighbors=K)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

```
y_pred = knn.predict(X_test)


print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[16,  0,  0],
       [ 0, 15,  0],
       [ 0,  0, 14]])
```

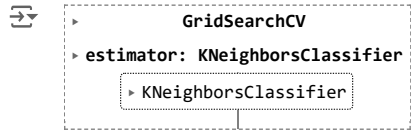
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        16
         1.0       1.00      1.00      1.00        15
         2.0       1.00      1.00      1.00        14

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Подбор гиперпараметра K

```
param_grid = {'n_neighbors': np.arange(1, 31)}
```

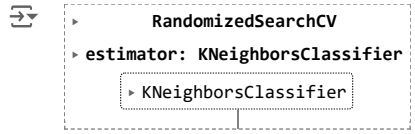
```
skf = StratifiedKFold(n_splits=5, random_state=RANDOM_STATE, shuffle=True)
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=skf, scoring='accuracy')
grid_search.fit(X, y)
```




```
print("Лучший результат GridSearchCV: {:.3f} с K={}".format(grid_search.best_score_, grid_search.best_params_['n_neighbors']))
```

```
Лучший результат GridSearchCV: 0.973 с K=6
```

```
loo = LeaveOneOut()
random_search = RandomizedSearchCV(KNeighborsClassifier(), param_grid, n_iter=20, cv=loo, scoring='accuracy', random_state=RANDOM_STATE)
random_search.fit(X, y)
```




```
print("Лучший результат RandomizedSearchCV: {:.3f} с K={}".format(random_search.best_score_, random_search.best_params_['n_neighbors']))
```

```
Лучший результат RandomizedSearchCV: 0.967 с K=7
```

Сравнение

Исходная модель была обучена на 70% датасета со случайно выбраным K=4 и показала точность 0.9(5).

Для второй модели коэффициент подбирался при помощи кросс-валидации **StratifiedKFold** для сохранения соотношения классов и **GridSearchCV**. Был подобран K=5 с точностью 0.973. Был подобран более оптимальный коэффициент, и получена более высокая точность.

Для 3 модели использовалась кросс-валидация **LeaveOneOut** с **RandomizedSearchCv** для компенсации большого времени выполнения. Получена точность 0.98 при K=19. Разница в точности с предыдующими моделями мала и может быть снова объяснена оценкой точности. Также возможно, LOO обеспечила максимальное использование данных, что при малом размере датасета (150 строк) привело к лучшему результату