Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа
по дисциплине
«Технологии машинного обучения»
на тему
«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

Выполнил:
студент группы ИУ5-64Б
Турусов В. И.

Москва — 2019 г.

1. Цель лабораторной работы

Изучение сложных способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей

2. Задание

1. Выбрать набор данных (датасет) для решения задачи классификации или регрессии.
2. С использованием метода train_test_split разделить выборку на обучающую и тестовую.
3. Обучить модель k-ближайших соседей для произвольно заданного гиперпараметра K. Оценить качество модели с помощью подходящих для задачи метрик.
4. Построить модель и оценить качество модели с использованием кросс-валидации.
5. Произвести подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.

3. Ход выполнения лабораторной работы

Подключим необходимые библиотеки и загрузим набор данных

[30]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

%matplotlib inline


# Для лучшего качествоа графиков
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")

# Устанавливаем ширину экрана для отчета
pd.set_option("display.width", 70)

# Загружаем данные
data = pd.read_csv('Iris2.csv')
data.head()
```

[30]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
0	1	5.1	3.5	1.4	0.2	
1	2	4.9	3.0	1.4	0.2	
2	3	4.7	3.2	1.3	0.2	
3	4	4.6	3.1	1.5	0.2	
4	5	5.0	3.6	1.4	0.2	

```
   target
0       1
1       1
2       1
3       1
4       1
```

[31]:
```
data.shape
```

[31]:
```
(150, 6)
```

[32]:
```
data.dtypes
```

[32]:
```
Id                int64
SepalLengthCm     float64
SepalWidthCm      float64
PetalLengthCm     float64
PetalWidthCm      float64
target            int64
dtype: object
```

[33]:
```
data.isna().sum()
```

[33]:
```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
target          0
dtype: int64
```

[34]:
```
data.isnull().sum()
```

[34]:
```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
target          0
dtype: int64
```

Как видим, пустых значений нет, значет нет необходимости преобразовывать набор данных

Разделим данные на целевой столбец и признаки

[35]:
```
X = data.drop("target", axis=1)
Y = data["target"]
print(X, "\n")
print(Y)
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0   1            5.1           3.5            1.4           0.2
```

```
1    2         4.9         3.0         1.4         0.2
2    3         4.7         3.2         1.3         0.2
3    4         4.6         3.1         1.5         0.2
4    5         5.0         3.6         1.4         0.2
..  ...        ...         ...         ...         ...
145  146       6.7         3.0         5.2         2.3
146  147       6.3         2.5         5.0         1.9
147  148       6.5         3.0         5.2         2.0
148  149       6.2         3.4         5.4         2.3
149  150       5.9         3.0         5.1         1.8

[150 rows x 5 columns]
```

```
0      1
1      1
2      1
3      1
4      1
      ..
145    0
146    0
147    0
148    0
149    0
Name: target, Length: 150, dtype: int64
```

[36]:
```
X.shape
```

[36]: (150, 5)

[37]:
```
Y.shape
```

[37]: (150,)

С использованием метода train_test_split разделим выборку на обучающую и тестовую

[38]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=1)
```

[28]:
```
print("X_train:", X_train.shape)
print("X_test:", X_test.shape)
print("Y_train:", Y_train.shape)
print("Y_test:", Y_test.shape)
```

```
X_train: (112, 6)
X_test: (38, 6)
Y_train: (112,)
Y_test: (38,)
```

Обучим модель k-ближайших соседей для произвольно заданного гиперпараметра K

[39]:
```
# В моделях k-ближайших соседей большое значение k
# ведёт к большому смещению и низкой дисперсии (недообучению)
# 70 ближайших соседей
cl1_1 = KNeighborsClassifier(n_neighbors=70)
cl1_1.fit(X_train, Y_train)
target1_0 = cl1_1.predict(X_train)
target1_1 = cl1_1.predict(X_test)
accuracy_score(Y_train, target1_0), accuracy_score(Y_test, target1_1)
```

[39]: (0.9910714285714286, 0.9736842105263158)

Построим модель и оценим качество модели с использованием кросс-валидации

[40]:
```
scores = cross_val_score(KNeighborsClassifier(n_neighbors=2), X, Y, cv=3)
```

[41]:
```
# Значение метрики accuracy для 3 фолдов
scores
```

[41]: array([0.76, 1. , 0.74])

[42]:
```
# Усредненное значение метрики accuracy для 3 фолдов
np.mean(scores)
```

[42]: 0.8333333333333334

Произведем подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации

[43]:
```
# Список настраиваемых параметров
n_range = np.array(range(1, 50, 2))
tuned_parameters = [{'n_neighbors': n_range}]
n_range
```

[43]: array([1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
35, 37, 39, 41, 43, 45, 47, 49])

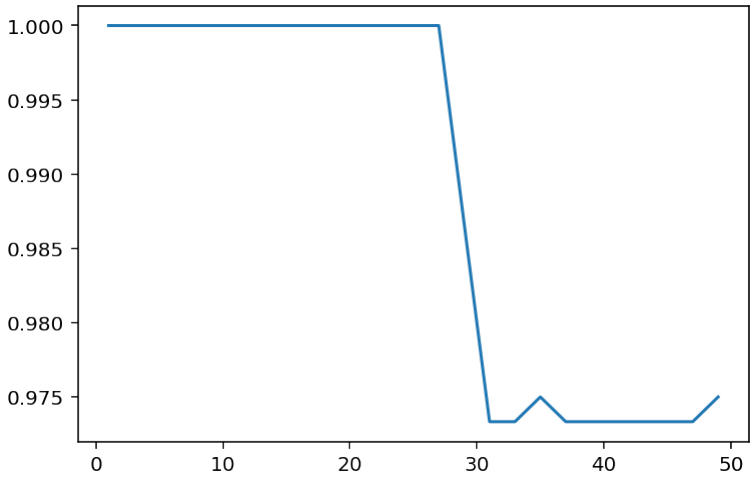
[44]:
```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5,
    ↪scoring='accuracy', return_train_score=True)
clf_gs.fit(X, Y)
clf_gs.best_params_
```

CPU times: user 1.59 s, sys: 13.8 ms, total: 1.6 s
Wall time: 1.61 s

[44]: {'n_neighbors': 1}

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

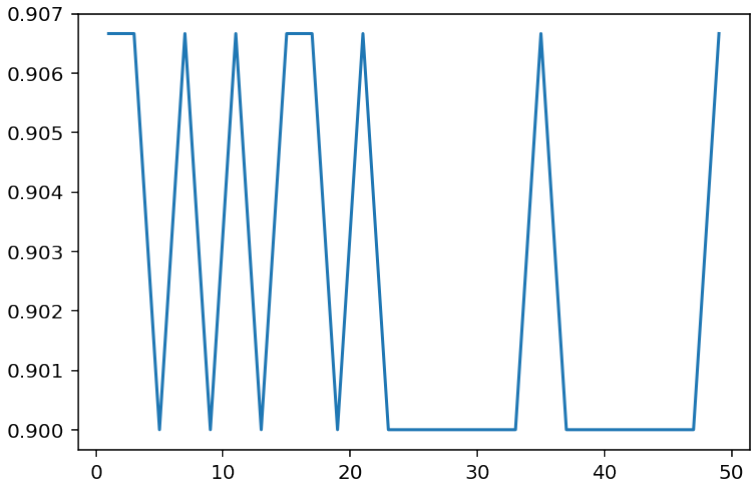
[45]:
```
plt.plot(n_range, clf_gs.cv_results_["mean_train_score"]);
```



Очевидно, что для $K = 1$ на тренировочном наборе данных мы находим ровно ту же точку, что и нужно предсказать, и чем больше её соседей мы берём — тем меньше точность.

Посмотрим на тестовом наборе данных

[46]:
```
plt.plot(n_range, clf_gs.cv_results_["mean_test_score"]);
```



Проверим получившуюся модель:

[47]:
```
cl1_2 = KNeighborsClassifier(**clf_gs.best_params_)
cl1_2.fit(X_train, Y_train)
target2_0 = cl1_2.predict(X_train)
target2_1 = cl1_2.predict(X_test)
accuracy_score(Y_train, target2_0), accuracy_score(Y_test, target2_1)
```

[47]: (1.0, 1.0)

Как видим, точность модели улучшилось