Московский Государственный Технический Университет
им. Н. Э. Баумана

Лабораторная работа №4
по курсу: «Технологии машинного обучения»

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

Выполнила:
Студентка группы ИУ5-63
Нурлыева Д.Д.

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.

Текстовое описание набора данных

В качестве набора данных мы будем использовать набор данных Heart Disease UCI - https://www.kaggle.com/ronitf/heart-disease-uci В датасете отражено наличие сердечного заболевания у пациента в зависимости от разных признаков.
Датасет содержит следующие колонки:
age - возраст в годах
sex - (1 = мужчина; 0 = женщина)
cp - тип боли в груди
trestbps - артериальное давление в состоянии покоя (в мм рт. ст. при поступлении в стационар)
chol - холестерин в мг/дл
fbs - уровень сахара в крови натощак > 120 мг / дл) (1 = да; 0 = нет)
restecg- электрокардиографические результаты покоя
thalach - максимальная ЧСС
exang - стенокардия, вызванная физическими упражнениями (1 = да; 0 = Нет)
oldpeak - Депрессия, вызванная физическими упражнениями относительно покоя
slope - наклон пика упражнения сегмента
ca - количество крупных сосудов (0-3)
thal - 3 = нормальный; 6 = фиксированный дефект; 7 = реверзибельный дефект
target - заболевание 1-есть или 0-нет

Текст программы:

```
import numpy as np
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor,
KNeighborsClassifier
from sklearn.metrics import accuracy_score,
balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score,
f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score,
cross_validate
from sklearn.model_selection import KFold, RepeatedKFold,
ShuffleSplit, StratifiedKFold
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve,
validation_curve
```

In [59]:

```
data=pd.read_csv("/Users/user/Desktop/data2.csv")
data.head()
```

Out[59]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [60]:

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

Out[60]:

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
```

```
ca          0
thal        0
target      0
dtype: int64
In [61]:
data_X_train, data_X_test, data_y_train, data_y_test =
train_test_split(
    data, data.target, test_size=0.2, random_state=1)
In [62]:
data_X_train.shape, data_y_train.shape
Out[62]:
((242, 14), (242,))
In [63]:
data_X_test.shape, data_y_test.shape
Out[63]:
((61, 14), (61,))
In [64]:
simple_knn = KNeighborsClassifier()
In [65]:
simple_knn.fit(data_X_train,data_y_train)
Out[65]:
KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5,
p=2,
           weights='uniform')
In [66]:
target1 = simple_knn.predict(data_X_test)
target1
Out[66]:
array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
1, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0])
In [67]:
#оценка
accuracy_score(data_y_test,target1)
Out[67]:
0.5737704918032787
In [68]:
precision_score(data_y_test, target1, average='micro')
Out[68]:
0.5737704918032787
In [69]:
```

```
recall_score(data_y_test,target1)
```
Out[69]:
```
0.6774193548387096
```
In [70]:
```
#Кроссвалидация
kfold = cross_val_score(KNeighborsClassifier(),
                        data, data['target'],
                        cv=KFold(n_splits=5))
kfold
```
Out[70]:
```
array([0.47540984, 0.63934426, 0.68852459, 0.4       ,
0.31666667])
```
In [71]:
```
shufflesplit = cross_val_score(KNeighborsClassifier(),
                        data, data['target'],
                        cv=ShuffleSplit(n_splits=5,
test_size=0.2))
shufflesplit
```
Out[71]:
```
array([0.67213115, 0.59016393, 0.63934426, 0.68852459,
0.70491803])
```
In [72]:
```
stratifiedkfold = cross_val_score(KNeighborsClassifier(),
                        data, data['target'],
                        cv=StratifiedKFold(n_splits=5))
stratifiedkfold
```
Out[72]:
```
array([0.60655738, 0.6557377 , 0.57377049, 0.73333333, 0.65
])
```
In [73]:
```
n_range = np.array(range(1,10,1))
tuned_parameters = [{'n_neighbors': n_range}]
clf_gs = GridSearchCV(KNeighborsClassifier(),
tuned_parameters,
                      cv=StratifiedKFold(n_splits=5),
scoring='accuracy')
clf_gs.fit(data_X_train, data_y_train)
```

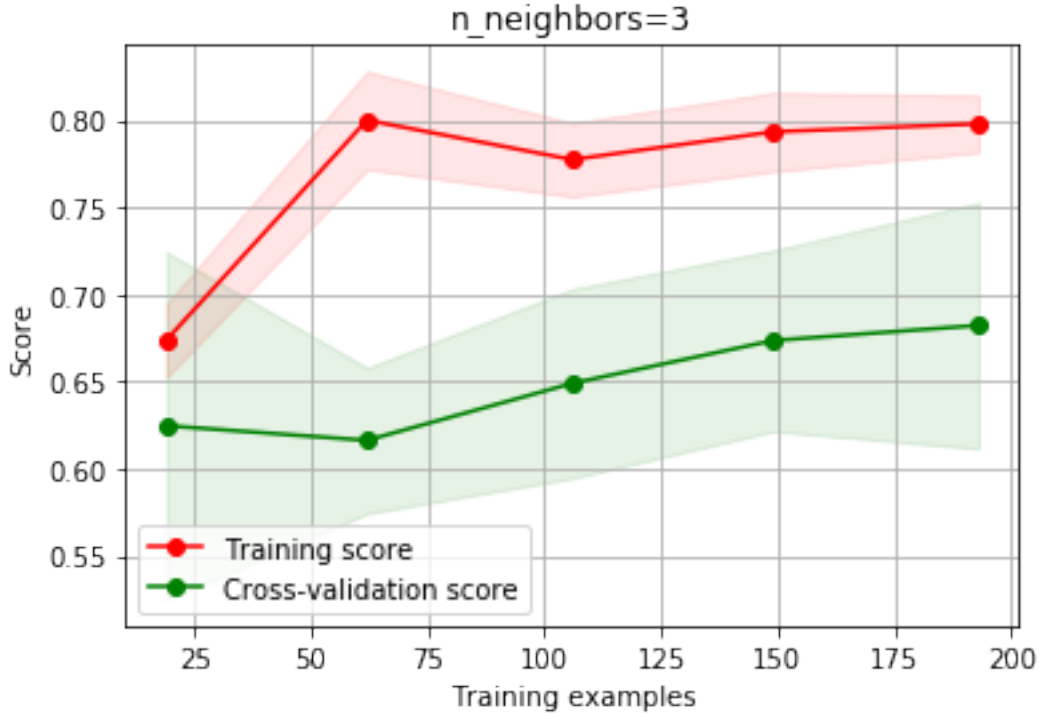
```
/anaconda3/lib/python3.6/site-packages/sklearn/
model_selection/_search.py:841: DeprecationWarning: The
default of the `iid` parameter will change from True to False
in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
```
Out[73]:

```
GridSearchCV(cv=StratifiedKFold(n_splits=5,
random_state=None, shuffle=False),
       error_score='raise-deprecating',
       estimator=KNeighborsClassifier(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': array([1, 2, 3, 4, 5, 6,
7, 8, 9])}],
       pre_dispatch='2*n_jobs', refit=True,
return_train_score='warn',
       scoring='accuracy', verbose=0)
In [74]:
clf_gs.best_params_
Out[74]:
{'n_neighbors': 3}
In [75]:
simple_knn_best = KNeighborsClassifier(n_neighbors=3)
In [76]:
simple_knn_best.fit(data_X_train, data_y_train)
Out[76]:
KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=3,
p=2,
           weights='uniform')
In [77]:
target2 = simple_knn_best.predict(data_X_test)
In [78]:
accuracy_score(data_y_test, target2)
Out[78]:
0.5737704918032787
In [79]:
precision_score(data_y_test, target2)
Out[79]:
0.5609756097560976
In [80]:
recall_score(data_y_test, target2)
Out[80]:
0.7419354838709677
In [84]:
def plot_learning_curve(estimator, title, X, y, ylim=None,
cv=None,
```

```
                        n_jobs=None,
train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and training learning
curve.

    Parameters
    ----------
    estimator : object type that implements the "fit" and
"predict" methods
        An object of that type which is cloned for each
validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of
samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples,
n_features), optional
        Target relative to X for classification or
regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : int, cross-validation generator or an iterable,
optional
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:
          - None, to use the default 3-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as
arrays of indices.

        For integer/None inputs, if ``y`` is binary or
multiclass,
        :class:`StratifiedKFold` used. If the estimator is
not a classifier
```

```
        or if ``y`` is neither binary nor
multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the
various
        cross-validators that can be used here.

    n_jobs : int or None, optional (default=None)
        Number of jobs to run in parallel.
        ``None`` means 1 unless in
a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors.
See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like, shape (n_ticks,), dtype float
or int
        Relative or absolute numbers of training examples
that will be used to
        generate the learning curve. If the dtype is float,
it is regarded as a
        fraction of the maximum size of the training set
(that is determined
        by the selected validation method), i.e. it has to be
within (0, 1].
        Otherwise it is interpreted as absolute sizes of the
training sets.
        Note that for classification the number of samples
usually have to
        be big enough to contain at least one sample from
each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs,
train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
```

```
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean -
train_scores_std,
                     train_scores_mean + train_scores_std,
alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean -
test_scores_std,
                     test_scores_mean + test_scores_std,
alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
In [85]:
plot_learning_curve(KNeighborsClassifier(n_neighbors=3),
'n_neighbors=3',
                    data_X_train, data_y_train,
cv=StratifiedKFold(n_splits=5))
Out[85]:
<module 'matplotlib.pyplot' from '/anaconda3/lib/python3.6/
site-packages/matplotlib/pyplot.py'>
```



In [86]:
```
def plot_validation_curve(estimator, title, X, y,
                          param_name, param_range, cv,
                          scoring="accuracy"):

    train_scores, test_scores = validation_curve(
        estimator, X, y, param_name=param_name,
param_range=param_range,
        cv=cv, scoring=scoring, n_jobs=1)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.title(title)
    plt.xlabel(param_name)
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.plot(param_range, train_scores_mean, label="Training
score",
                 color="darkorange", lw=lw)
    plt.fill_between(param_range, train_scores_mean -
train_scores_std,
                     train_scores_mean + train_scores_std,
alpha=0.2,
                     color="darkorange", lw=lw)
    plt.plot(param_range, test_scores_mean, label="Cross-
validation score",
                 color="navy", lw=lw)
    plt.fill_between(param_range, test_scores_mean -
test_scores_std,
                     test_scores_mean + test_scores_std,
alpha=0.2,
                     color="navy", lw=lw)
    plt.legend(loc="best")
    return plt
```

In [88]:
```
plot_validation_curve(KNeighborsClassifier(), 'knn',
                      data_X_train, data_y_train,
                      param_name='n_neighbors',
param_range=n_range,
                      cv=StratifiedKFold(n_splits=5),
scoring="accuracy")
```

Out[88]:

```
<module 'matplotlib.pyplot' from '/anaconda3/lib/python3.6/
site-packages/matplotlib/pyplot.py'>
```