机器学习库Scikit-learn

实验目标

通过本案例的学习和课后作业的练习：

1. 了解sklearn中特征工程的常规操作；
2. 了解sklearn在回归、分类、聚类算法的实现，加深对机器学习算法的应用。

你也可以将本案例相关的 ipynb 学习笔记分享到 AI Gallery Notebook (https://marketplace.huaweicloud.com/markets/aihub/notebook/list/) 版块获得成长值 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=9b8d7e7a-a150-449e-ac17-2dcf76d8b492)，分享方法请查看此文档 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=8afec58a-b797-4bf9-acca-76ed512a3acb)。

Sckit-learn介绍

机器学习是实现人工智能的重要途径，在Python中实现机器学习算法是非常容易的，我们只需要借助机器学习工具Scikit-learn，调用其中已经封装好的接口即可。

Scikit-learn(sklearn)是机器学习中常用的第三方模块，对常用的机器学习方法进行了封装，包括回归(Regression)、降维(Dimensionality Reduction)、分类(Classfication)、聚类(Clustering)等方法。

本案例推荐的理论学习视频：

- 《AI基础课程--常用框架工具》 机器学习库Scikit-learn (https://education.huaweicloud.com/courses/course-v1:HuaweiX+CBUCNXE081+Self-paced/courseware/5935e6c413ed49668fed02dbbd9fd2fc/921c9878e55447c1a1911b20af6bdf15/)

注意事项

1. 本案例推荐使用AI引擎：**XGBoost-Sklearn**；
2. 如果你是第一次使用 JupyterLab，请查看《ModelArts JupyterLab使用指导》 (https://bbs.huaweicloud.com/forum/thread-97603-1-1.html)了解使用方法；
3. 如果你在使用 JupyterLab 过程中碰到报错，请参考《ModelArts JupyterLab常见问题解决办法》 (https://bbs.huaweicloud.com/forum/thread-98681-1-1.html)尝试解决问题。

实验步骤

1. 特征工程

步骤 1 特征抽取

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

from sklearn.feature_extraction import DictVectorizer

# onehot编码
data = [{'name': '张三','age':20}, {'name': '李四','age':24}, {'name': '
王五','age':18}]

# 实例化一个转换器类
transfer = DictVectorizer(sparse=False)

# 调用fit_transform
data = transfer.fit_transform(data)

print("返回的结果:\n", data)
print("特征名字: \n", transfer.get_feature_names())
```

返回的结果:

```
 [[20.  1.  0.  0.]

 [24.  0.  1.  0.]

 [18.  0.  0.  1.]]
```

特征名字:

```
 ['age', 'name=张三', 'name=李四', 'name=王五']
```

步骤 2 归一化

对于归一化来说：异常点会影响最后的结果。

In [2]:
```
# 创建数据
import pandas as pd

l1 = [30,1,5]
l2 = [27,3,6]
l3 = [34,2,6]
df = pd.DataFrame([l1,l2,l3], columns=["a","b","c"])
print(df)
```

Out[2]:

	a	b	c
0	30	1	5
1	27	3	6
2	34	2	6

In [3]:
```
from sklearn.preprocessing import MinMaxScaler  # 归一化API

transfer = MinMaxScaler(feature_range=(0, 1))    # 实例化转换器
data = transfer.fit_transform(df[['a','b',"c"]])
print(data)
```

Out[3]:
```
array([[0.42857143, 0.        , 0.        ],
       [0.        , 1.        , 1.        ],
       [1.        , 0.5       , 1.        ]])
```

步骤 3 标准化

In [4]:
```
from sklearn.preprocessing import StandardScaler

transfer = StandardScaler()
data = transfer.fit_transform(df[['a','b',"c"]])
print(data)
```

Out[4]:
```
array([[-0.11624764, -1.22474487, -1.41421356],
       [-1.16247639,  1.22474487,  0.70710678],
       [ 1.27872403,  0.        ,  0.70710678]])
```

2. 回归算法

步骤 1 线性回归

使用线性回归进行波士顿房价预测。

波士顿房价数据集是 scikit-learn 中内置的数据集，共有506条数据包含了13个特征和1个标签，每条数据包含房屋以及房屋周围的详细信息。其中包含城镇犯罪率，一氧化氮浓度，住宅平均房间数，到中心区域的加权距离以及自住房平均房价等等。

In [5]:
```
# 导入所需工具
from sklearn.datasets import load_boston                # 导入数据波士顿房价
from sklearn.linear_model import SGDRegressor           # 线性回归
from sklearn.model_selection import train_test_split    # 划分数据集
from sklearn.preprocessing import StandardScaler        # 数据标准化
from sklearn.metrics import mean_squared_error          # 均方误差
```

In [6]:
```
# 查看数据的一些属性
boston = load_boston()  # 导入加载的数据集

print("数据维度：", boston.data.shape)  # 查看数据维度
print("房价数据：", boston.data)        # 查看数据
print("特征：", boston.feature_names)  # 查看数据的特征名称
print("标签：", boston.target)          # 查看标签数据
```

数据维度： (506, 13)

房价数据： [[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]

[2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]

[2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]

...

[6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]

[1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]

[4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]

特征： ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'

'B' 'LSTAT']

标签： [24. 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15. 18.9 21.7 20.4

18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8

18.4 21. 12.7 14.5 13.2 13.1 13.5 18.9 20. 21. 24.7 30.8 34.9 26.6

25.3 24.7 21.2 19.3 20. 16.6 14.4 19.4 19.7 20.5 25. 23.4 18.9 35.4

24.7 31.6 23.3 19.6 18.7 16. 22.2 25. 33. 23.5 19.4 22. 17.4 20.9

24.2 21.7 22.8 23.4 24.1 21.4 20. 20.8 21.2 20.3 28. 23.9 24.8 22.9

23.9 26.6 22.5 22.2 23.6 28.7 22.6 22. 22.9 25. 20.6 28.4 21.4 38.7

43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8

18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22. 20.3 20.5 17.3 18.8 21.4

15.7 16.2 18. 14.3 19.2 19.6 23. 18.4 15.6 18.1 17.4 17.1 13.3 17.8

```
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.
4

 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.
8

 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.
2

 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.

 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.

 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.
1

 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.
5

 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.
8

 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.
8

 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.
1

 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.
9

 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.
2

 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.
1

 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.
1

 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.
6

 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.
8

 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.
3

 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.
2

  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.

 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.
```

```
4

 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.
3

 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.
6

 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.
7

 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.
3

 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.

  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.
9

 22.  11.9]
```

In [7]:
```
# 划分数据集
x_train, x_test, y_train, y_test = train_test_split(boston.data, bosto
n.target, test_size=0.2, random_state=6)

print("训练集：", x_train)
print("训练集维度：", x_train.shape)
print("测试集：", x_test)
print("测试集维度：", x_test.shape)
```

```
训练集： [[2.53870e-01 0.00000e+00 6.91000e+00 ... 1.79000e+01 3.96900
e+02
  3.08100e+01]
 [7.52601e+00 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.04210e+02
  1.93100e+01]
 [3.30600e-02 0.00000e+00 5.19000e+00 ... 2.02000e+01 3.96140e+02
  8.51000e+00]
 ...
 [4.12380e-01 0.00000e+00 6.20000e+00 ... 1.74000e+01 3.72080e+02
  6.36000e+00]
 [3.44500e-02 8.25000e+01 2.03000e+00 ... 1.47000e+01 3.93770e+02
  7.43000e+00]
 [1.33598e+01 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.96900e+02
  1.63500e+01]]
训练集维度： (404, 13)
测试集： [[1.96091e+01 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.96900
e+02
  1.34400e+01]
 [1.87000e-02 8.50000e+01 4.15000e+00 ... 1.79000e+01 3.92430e+02
  6.36000e+00]
 [6.14700e-01 0.00000e+00 6.20000e+00 ... 1.74000e+01 3.96900e+02
  7.60000e+00]
 ...
 [1.78667e+01 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.93740e+02
  2.17800e+01]
 [1.05393e+00 0.00000e+00 8.14000e+00 ... 2.10000e+01 3.86850e+02
  6.58000e+00]
 [1.98020e-01 0.00000e+00 1.05900e+01 ... 1.86000e+01 3.93630e+02
  9.47000e+00]]
```

```
测试集维度： (102, 13)
```

In [8]:
```
# 数据标准化
transfer = StandardScaler()                    # 实例化标准化对象
x_train = transfer.fit_transform(x_train)   # 将数据进行标准化
x_test = transfer.transform(x_test)

# 数据在标准化后数值发生了改变，但是数据维度没有变化。
print("标准化：",x_train)
print('标准化后的维度：',x_train.shape)
```

```
标准化： [[-0.39262879 -0.48699244 -0.58398492 ... -0.23241985  0.4382
3462

   2.60935849]

 [ 0.49588716 -0.48699244  1.0378004  ...  0.82415806 -0.56005456

   0.97851821]

 [-0.41960753 -0.48699244 -0.83326738 ...  0.82415806  0.43004927

  -0.55305353]

 ...

 [-0.37326191 -0.48699244 -0.6868864  ... -0.4621107   0.17091843

  -0.85794976]

 [-0.4194377   2.9643063  -1.29125144 ... -1.70244128  0.40452391

  -0.70621071]

 [ 1.20866434 -0.48699244  1.0378004  ...  0.82415806  0.43823462

   0.5587541 ]]
```

```
标准化后的维度： (404, 13)
```

In [9]:
```
# 实现线性回归算法
estimator = SGDRegressor()          # 线性回归
estimator.fit(x_train, y_train)  # 使用fit方法填充数据进行训练
```

Out[9]:
```
SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsil
on=0.1,
       eta0=0.01, fit_intercept=True, l1_ratio=0.15,
       learning_rate='invscaling', loss='squared_loss', max_iter=Non
e,
       n_iter=None, n_iter_no_change=5, penalty='l2', power_t=0.25,
       random_state=None, shuffle=True, tol=None, validation_fraction
=0.1,
       verbose=0, warm_start=False)
```

In [10]:
```
# predict方法进行测试
y_predict = estimator.predict(x_test)
print(y_predict)
```

```
[25.66304629 25.13714668 28.42876701 23.44349367 19.90986507 19.86896
904

 15.94287184 43.14224108 21.66178292 21.05633559 24.56497577 12.24033
423

 32.11913973 31.61838943 33.23555082 20.04931646 13.69584467 28.06995
325

 16.74515625 16.76005613 16.7865095  28.16809444 22.98951999 19.14878
36

 19.53210285 19.89009861 20.86688403 28.14894805 19.87563046 24.72814
102

 23.89937672 22.37810222 32.07104609 37.19616578 19.62817949 23.26671
136

 15.06178493 20.17716645 17.75280607  3.81723883 11.77523546 12.00069
766

 14.02708847 12.88946119 20.73311633 24.48011455 31.71314515 22.72889
116

 19.6056008  18.10438766 20.69767215 14.45312521 23.30689574 20.02632
177

 35.80145077 26.06149924 13.25211449 28.73421701 17.70497502 20.34105
026

 23.30200967 22.73341718 12.63311598 28.2506297  17.48306914 15.31717
122

 31.65019712 14.79432464 25.9704725  33.85217279  9.61769256 21.48102
57

 13.71112357 25.12007422 30.70275588 22.42531838 16.91148871 39.17309
965

 13.8060709  20.69564284 20.72464963 13.62760777 -0.36074673 34.20958
205

 17.78183316 34.30594876 11.74266837 17.9188072  14.15639955 37.88846
762

 12.62355268  2.56038196 15.36426526 26.66514194 20.73388127 17.48638
697

 26.09076527 18.33979653 25.7373555  16.66175383 20.68164875 23.38957
968]
```

In [11]:
```
# 使用sklearn中的均方误差函数计算误差，在回归实验中MSE较为常用。
error = mean_squared_error(y_test, y_predict)
print(error)
```

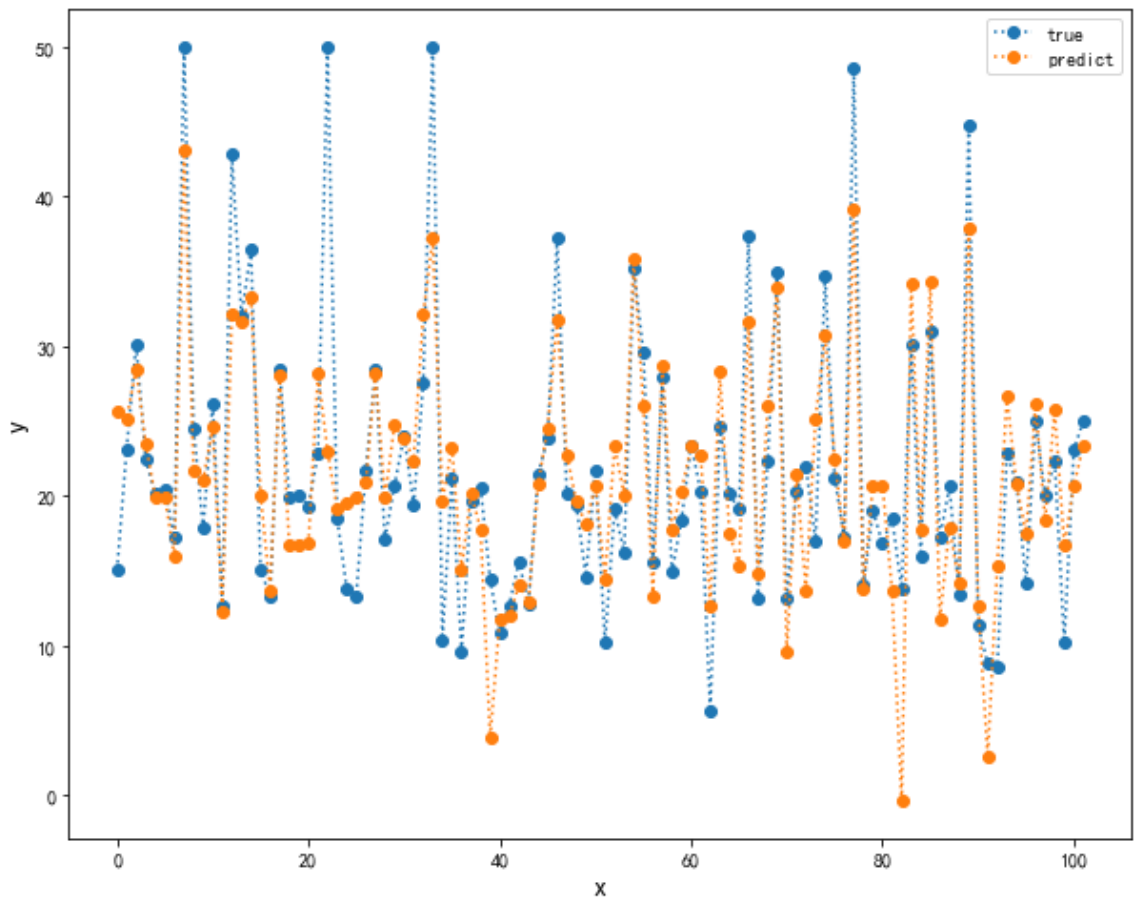
```
26.738819934899958
```

In [12]:
```
# 使用matplotlib进行可视化来查看预测结果和真实结果的差异性
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize=(10,8))
plt.xlabel("x",fontsize=14)
plt.ylabel("y", fontsize=14)

plt.plot([i for i in range(len(y_test))],
         y_test,linestyle=':',
         marker='o' ,
         label="true")

plt.plot([i for i in range(len(y_test))],
         y_predict, linestyle=':',
         marker='o' ,
         label="predict")
plt.legend()
plt.show()
```



3. 分类算法

步骤 1 逻辑回归

使用逻辑回归来进行癌症分类预测。

Breast-Cancer数据集

数据集源于威斯康星州临床科学中心。每个记录代表一个乳腺癌的随访数据样本。这些是DR Wolberg自1984~1995随访搜集连续乳腺癌患者数据，数据仅包括那些具有侵入性的病例乳腺癌并没有远处转移的医学指标数据集。

（数据地址：https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/ (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/) ）：

数据特征：

有11个列，第1个列为id号，第2-10列为特征，11列为标签（2为良性、4为恶性）

- 数据集中共有699条信息；
- 16处缺失值，缺失值使用"?"表示；
- 良性数据有458条，恶性数据有241条

由于实验中所用到的数据是开源数据，这些数据已经被专业人士处理过了，所以我们不需要做过多处理（下同）。

In [13]:
```
import numpy as np

from sklearn.model_selection import train_test_split  # train_test_split用来划分数据集
from sklearn.linear_model import LogisticRegression   # 逻辑回归
from sklearn.metrics import classification_report     # 使用classification_report模块测评

column_name = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                   'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
                   'Normal Nucleoli', 'Mitoses', 'Class']

data = pd.read_csv(r"https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data",
                   names=column_name)

# 删除缺失值
data = data.replace(to_replace='?', value=np.nan)
data = data.dropna()

x = data[column_name[1:10]]  # 取出特征值
y = data[column_name[10]]    # 取出标签值

# 分割数据集 测试集占比30%
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

# 进行标准化
std = StandardScaler()

x_train = std.fit_transform(x_train)
x_test= std.transform(x_test)

# 使用逻辑回归
lr = LogisticRegression()
lr.fit(x_train, y_train)
print("得出来的权重：", lr.coef_)

# 预测类别
y_pred  = lr.predict(x_test)
print("预测的类别：", y_pred)

# 得出准确率
print("预测的准确率:", lr.score(x_test, y_test))
print('预测结果准确性:',classification_report(y_test,y_pred,target_names=['良性', '恶性']),sep='\n')
```

得出来的权重： [[1.03238923 0.75652064 0.74341775 0.85052613 0.18235045 0.99631293

0.90916744 0.45606448 0.73397132]]

预测的类别： [4 2 4 4 2 2 2 4 2 2 4 2 4 4 2 2 4 2 4 4 4 4 2 2 4 2 2 2 4 2 2 4 4 2 2 4 2

2 4 2 4 2 2 2 4 2 2 4 2 4 4 4 4 2 4 4 2 2 2 2 4 2 2 2 2 4 2 4 2 2 2 2 4 2

2 4 4 4 2 2 2 2 4 2 2 2 4 2 2 2 2 4 2 4 4 2 2 2 2 4 2 2 4 2 2 2 4 4 2 2

4 4 4 4 4 4 2 4 2 2 4 4 2 2 2 4 2 4 2 4 2 2 4 2 2 2 4 2 2 2 2 2 4 2 4 4 4

2 2 2 4 4 2 2 4 2 2 2 4 2 2 2 2 2 2 4 2 2 2 4 4 2 4 2 2 4 2 2 4 2 2 2 2 2

2 4 4 4 2 2 2 2 2 4 4 2 2 2 2 2 2 4 4 2]

预测的准确率：0.975609756097561

预测结果准确性：

	precision	recall	f1-score	support
良性	0.98	0.98	0.98	128
恶性	0.96	0.97	0.97	77
micro avg	0.98	0.98	0.98	205
macro avg	0.97	0.98	0.97	205
weighted avg	0.98	0.98	0.98	205

步骤 2 KNN

鸢尾花识别是一个经典的机器学习分类问题，它的数据样本中包括了4个特征变量，1个类别变量，样本总数为150。

它的目标是为了根据花萼长度（sepal length）、花萼宽度（sepal width）、花瓣长度（petal length）、花瓣宽度（petal width）这四个特征来识别出鸢尾花属于山鸢尾（iris-setosa）、变色鸢尾（iris-versicolor）和维吉尼亚鸢尾（iris-virginica）中的哪一种。

下面我们使用KNN算法实现鸢尾花种类预测。

In [14]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()
# x_train,x_test,y_train,y_test为训练集特征值、测试集特征值、训练集目标值、测试集目标值
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=22)
transfer = StandardScaler()
x_train = transfer.fit_transform(x_train)
x_test = transfer.transform(x_test)

# 实例化KNN分类器
estimator = KNeighborsClassifier(n_neighbors=9)
estimator.fit(x_train, y_train)

# 模型评估
y_predict = estimator.predict(x_test)

print("预测结果为:\n", y_predict)
print("比对真实值和预测值: \n", y_predict == y_test)

score = estimator.score(x_test, y_test)
print("准确率为: \n", score)
```

预测结果为:

```
 [0 2 1 2 1 1 1 1 1 0 2 1 2 2 0 2 1 1 1 1 0 2 0 1 2 0 2 2 2 2]
```

比对真实值和预测值:

```
 [ True  True  True  True  True  True  True False  True  True  True
True

  True  True  True  True  True  True False  True  True  True  True  T
rue

  True  True  True  True  True  True]
```

准确率为:

```
 0.9333333333333333
```

步骤 3 决策树

使用决策树算法实现泰坦尼克号乘客生存预测。（数据地址：https://www.kaggle.com/c/titanic (https://www.kaggle.com/c/titanic) ）

数据中共有12个特征，此次试验我们选取其中的 'pclass', 'age', 'sex' 三列数据作为实验中的特征。

In [15]:
```
# 从公共OBS桶中拷贝泰坦尼克号数据,路径不用修改。
import os
import moxing as mox

if not os.path.isdir("./sk_learn_data"):
    mox.file.copy_parallel("obs://modelarts-labs-bj4/course/hwc_edu/python_module_framework/datasets/sk_learn_data/kaggle_titanic-master/", "sk_learn_data/kaggle_titanic-master/")
```

```
INFO:root:Using MoXing-v1.17.3-

INFO:root:Using OBS-Python-SDK-3.20.7
```

In [16]:
```
from sklearn.tree import DecisionTreeClassifier

titanic = pd.read_csv("sk_learn_data/kaggle_titanic-master/train.csv")
x = titanic[['Pclass', 'Age', 'Sex']]
y = titanic['Survived']

# 缺失值需要处理，将特征当中有类别的这些特征进行字典特征抽取
x['Age'].fillna(x['Age'].mean(), inplace=True)

dict = DictVectorizer(sparse=False)
x = dict.fit_transform(x.to_dict(orient="records"))
print(dict.get_feature_names())
print(x)

# 分割训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
3)

# 进行决策树的建立和预测，指定树的深度大小为5
dc = DecisionTreeClassifier(criterion='entropy', max_depth=5)
dc.fit(x_train, y_train)
print("预测的准确率为：", dc.score(x_test, y_test))
```

```
['Age', 'Pclass', 'Sex=female', 'Sex=male']

[[22.          3.          0.          1.        ]

 [38.          1.          1.          0.        ]

 [26.          3.          1.          0.        ]

 ...

 [29.69911765  3.          1.          0.        ]

 [26.          1.          0.          1.        ]

 [32.          3.          0.          1.        ]]

预测的准确率为： 0.8059701492537313
```

4. 聚类算法

步骤 1 K-means

k-means对于大型数据集也是简单高效、时间复杂度、空间复杂度低。 最重要是数据集大时结果容易局部最优；需要预先设定K值，对最先的K个点选取很敏感；对噪声和离群值非常敏感。

下面我们使用k-means聚类算法实现鸢尾花的聚类操作。

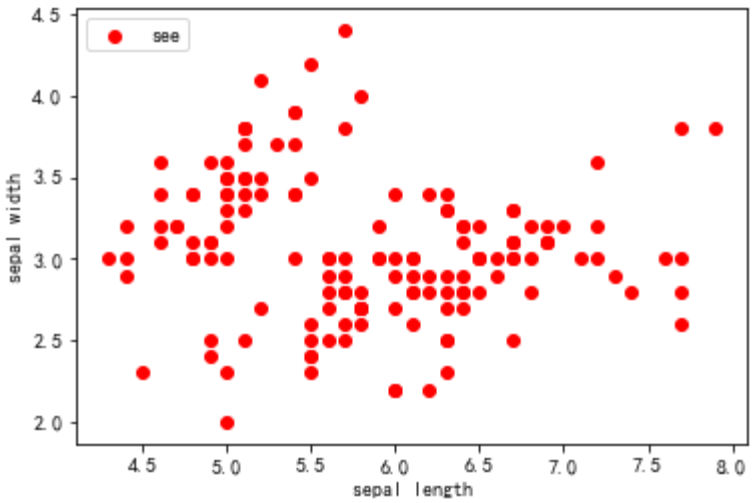
In [17]:
```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.cluster import KMeans
from sklearn import datasets

iris = datasets.load_iris()  # 引入鸢尾花数据
X = iris.data[:, :4]         # 表示我们取特征空间中的4个维度
print(X.shape)

# 绘制数据分布图
plt.scatter(X[:, 0], X[:, 1], c="red", marker='o', label='see')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)
plt.show()
```

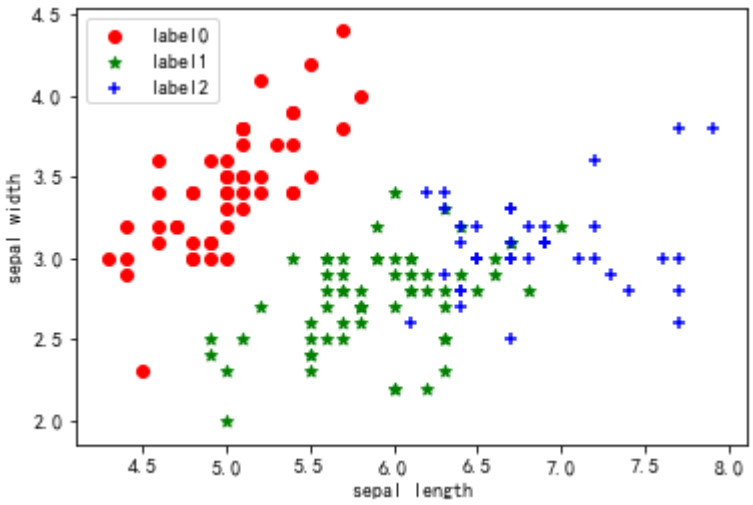
```
(150, 4)
```



In [18]:
```
estimator = KMeans(n_clusters=3)   # 构造聚类器
estimator.fit(X)                   # 聚类
label_pred = estimator.labels_     # 获取聚类标签

# 绘制k-means结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]

plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='*', label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+', label='label2')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)
plt.show()
```



步骤 2 DBSCAN

对于同样的鸢尾花数据使用DBSCAN进行聚类操作。

DBSCAN对噪声不敏感；能发现任意形状的聚类。 但是聚类的结果与参数有很大的关系；DBSCAN用固定参数识别聚类，但当聚类的稀疏程度不同时，相同的判定标准可能会破坏聚类的自然结构，即较稀的聚类会被划分为多个类或密度较大且离得较近的类会被合并成一个聚类。

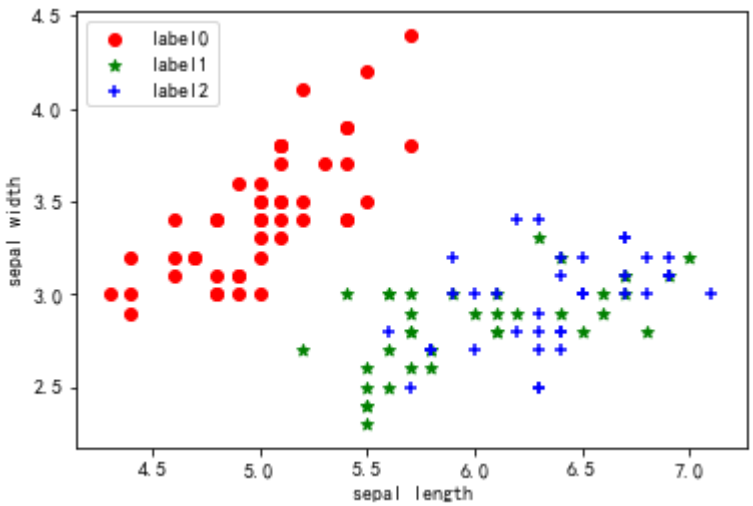
DBSCAN中有两个非常重要的参数eps和min_samples， 这两个参数表示数据的稠密性。当min_samples增加或者 eps 减小的时候，意味着一个簇分类有更大的密度要求。

In [19]:
```
from sklearn.cluster import DBSCAN

estimator = DBSCAN(eps = 0.4,min_samples = 4)    # 构造聚类器
estimator.fit(X)                                 # 聚类
label_pred = estimator.labels_                   # 获取聚类标签

# 绘制k-means结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]

plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='*', label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+', label='label2')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)
plt.show()
```



以上是 Scikit_learn 的基本使用，受限于篇幅原因，本案例未完全覆盖 Scikit_learn 中的全部操作，欢迎你将更全面的 Scikit_learn 学习笔记分享到 AI Gallery Notebook (https://marketplace.huaweicloud.com/markets/aihub/notebook/list/) 版块获得成长值 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=9b8d7e7a-a150-449e-ac17-2dcf76d8b492)，分享方法请查看此文档 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=8afec58a-b797-4bf9-acca-76ed512a3acb)。

作业

请你利用本课程中学到的知识和已掌握的知识，完成以下编程题：

1. 利用案例中的癌症分类数据集Breast-Cancer，使用**SVM**算法训练，并验证模型准确率。(https://marketplace.huaweicloud.com/markets/aihub/notebook/detail/?id=d59e1783-55d4-4c6e-935c-d9eae504408e)