SVM实战

-----BY YANGZHONGXIU 2020.04

内容

- SVC之鸢尾花
- SVC之人脸识别
- SVR 之california　housing

SVC之鸢尾花

概述

任务：采用支持向量机对鸢尾花进行类型鉴别。

目的：

文本数据的读取（两种方法：numpy 与pandas）

ＳＶＣ的应用

预测结果分析

数据可视化

数据分析

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
6.5,3.2,5.1,2.0,Iris-virginica
6.4,2.7,5.3,1.9,Iris-virginica
6.8,3.0,5.5,2.1,Iris-virginica
5.7,2.5,5.0,2.0,Iris-virginica
5.8,2.8,5.1,2.4,Iris-virginica
```

sepal_length, sepal_width,
petal_length, petal_width,
class

文本数据读取—numpy

```
path = 'iris/iris1.txt'
data = np.loadtxt(path, dtype=float, delimiter=',' , converters={4: Iris_label} )

def Iris_label(s):
    it = {b'Iris-setosa':0,b'Iris-versicolor':1,b'Iris-virginica':2}
    return it[s]
```

文本数据读取—Pandas

```
path = 'iris/iris.data'
data_0 = DataFrame(pd.read_csv(path))
data_0['class'] = data_0['class'].apply(Iris_label)
```

```
def Iris_label(s):
    it = {b'Iris-setosa':0,b'Iris-versicolor':1,b'Iris-virginica':2}
    return it[s]
```

划分数据与标签—numpy

Numpy读取数据划分：

x, y = **np.split**(data, (4,), axis=1)

train_data, test_data, train_label, test_label =

train_test_split (x, y, random_state = 1, train_size = 0.6,

test_size = 0.4)#sklearn.model_selection.

划分数据与标签—pandas

feature = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

label=['class']

x = data_0[feature]

x = np.array(x)#为模型准备数据，第一行不能有column title

y = data_0[label]

y = np.array(y)#同 x 处理原因

train_data, test_data, train_label, test_label = **train_test_split** (x, y, random_state = 1, train_size = 0.6, test_size = 0.4)#sklearn.model_selection.

训练ＳＶＣ分类器

```
classifier=SVC(C=2,  kernel='rbf',  gamma=10,
decision_function_shape='ovo')
# ovo：一对一；ovr:一对其余策略
#rbf为高斯核,又称"径向基"核
classifier.fit(train_data,  train_label.ravel())
#ravel函数在降维时默认是行序优先
```

计算svc分类器的准确率

```
print("训练集：",classifier.score(train_data,train_label))
print("测试集：",classifier.score(test_data,test_label))

#也可直接调用accuracy_score方法计算准确率
tra_label = classifier.predict(train_data)#训练集的预测标签
tes_label = classifier.predict(test_data)#测试集的预测标签
print("训练集：",accuracy_score(train_label, tra_label))
print("测试集：",accuracy_score(test_label, tes_label))
```

查看决策函数

print('train_decision_function:\n’, classifier.**decision_function**(train_data)) # (90,3)

print('predict_result:\n’, **classifier.predict**(train_data))

ovo

```
train_decision_function:
 [[ 1.00809313  1.01540937  0.151101  ]
 [-1.00777232 -1.00008021  0.63444159]
 [-0.99416388 -1.06070485 -0.681607  ]
 [ 1.0000563   1.00031087  0.05426675]
 [ 1.01709573  1.02782828  0.14377175]
 [-0.94393468 -1.02064785 -1.00008844]
 [ 0.99963476  1.00001849  0.15117146]
 [-0.96836014 -0.99980535  0.58258857]
 [-0.57544729 -1.05719931 -1.38921374]
 [-0.99957828 -1.07645757 -0.44947688]
 [-1.06076059 -1.12561405  1.00935269]
 [-0.61824976 -1.00025184 -0.99988863]
 [-1.04911984 -1.01738974  0.82720369]
 [ 1.0004868   0.99973226  0.15150631]
 [-0.99959974 -0.27106766  1.00008043]
 [-1.00037028 -1.0072383  -0.79454   ]
 [-0.94393468 -1.02064785 -1.00008844]
 [-1.16901646 -0.68521942  0.99969349]
 [-0.36188214 -1.05474156 -1.22681095]
 [-0.9998921  -0.7292288   1.00000149]
 [ 0.9997492   1.00028017  0.16527647]
 [ 1.10037424  1.1294864   0.11662767]
```

ovr

```
train_decision_function:
 [[ 2.41356518  0.82484721 -0.23841239]
 [-0.41036662  2.33563709  1.07472954]
 [-0.41997583  1.06388064  2.35609519]
 [ 2.40883676  0.80669872 -0.21553548]
 [ 2.41794332  0.82150929 -0.23945262]
 [-0.40152306  0.98852325  2.41299982]
 [ 2.40869085  0.82659034 -0.23528119]
 [-0.40225535  2.31698423  1.08527112]
 [-0.33368171  0.83368171  2.5       ]
 [-0.42430199  1.1124302   2.31187179]
 [-0.44685313  2.42309153  1.02376159]
 [-0.33079075  0.92200032  2.40879043]
 [-0.422355    2.38348461  1.03887039]
 [ 2.40880649  0.82648464 -0.23529113]
 [-0.25970009  2.40869635  0.85100374]
 [-0.41031677  1.04206777  2.368249  ]
 [-0.40152306  0.98852325  2.41299982]
 [-0.37897032  2.4432428   0.93572752]
 [-0.28953077  0.82322511  2.46630566]
 [-0.35341906  2.40875982  0.94465924]
```

```
predict_result:
 [0. 1. 2. 0. 0. 2. 0. 1. 2. 2. 1. 2. 1. 0. 1. 2. 2. 1. 2. 1. 0. 0. 0. 2.
 0. 1. 2. 1. 0. 0. 1. 0. 2. 1. 2. 2. 1. 2. 2. 1. 0. 1. 0. 1. 1. 0. 1. 0.
 0. 2. 1. 1. 0. 0. 1. 0. 1. 0. 2. 1. 0. 2. 0. 1. 0. 1. 1. 0. 0. 1. 0. 1.
 1. 0. 2. 1. 1. 1. 1. 0. 0. 1. 1. 2. 1. 2. 2. 1. 2. 0.]
```

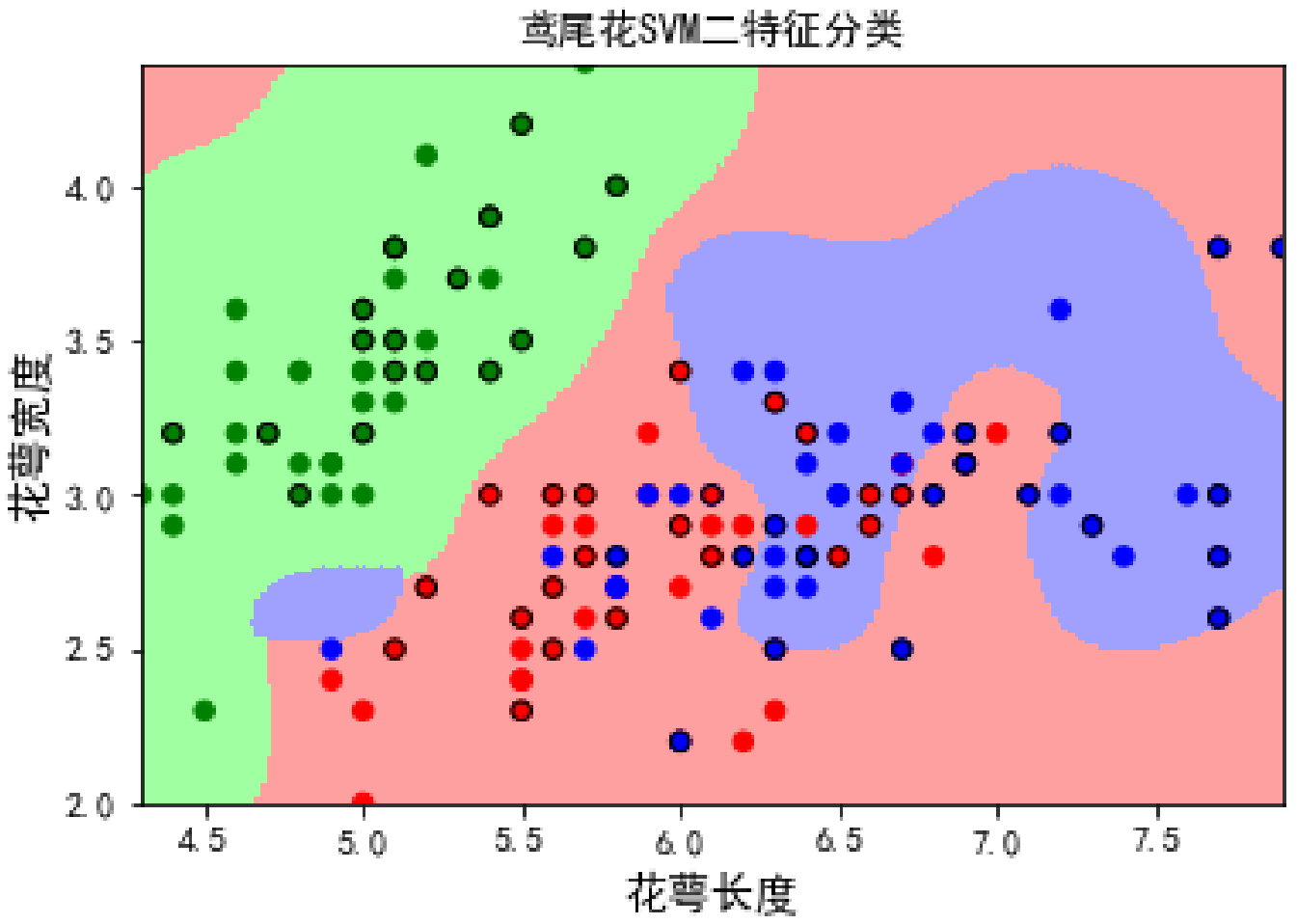
绘制图形

```
#确定坐标轴范围
x1_min,x1_max=x[:,0].min(),x[:,0].max()#第0维特征的范围
x2_min,x2_max=x[:,1].min(),x[:,1].max()#第1维特征的范围
x1,x2=np.mgrid[x1_min:x1_max:200j,x2_min:x2_max:200j]#生成网络采样点
grid_test=np.stack((x1.flat,x2.flat),axis=1)#测试点
#指定默认字体
mpl.rcParams['font.sans-serif']=['SimHei']
```

```
#设置颜色
cm_light=mpl.colors.ListedColormap(['#A0FFA0','#FFA0A0','#A0A0FF'])
cm_dark=mpl.colors.ListedColormap(['g','r','b'])

grid_hat=classifier.predict(grid_test)#预测分类值
grid_hat=grid_hat.reshape(x1.shape)#使之与输入的形状相同

plt.pcolormesh(x1,x2,grid_hat,cmap=cm_light)# 预测值的显示
plt.scatter(x[:,0],x[:,1],c=y[:,0],s=30,cmap=cm_dark)# 样本
plt.scatter(test_data[:,0],test_data[:,1],c=test_label[:,0],s=30,edgecolors='k',zor
der=2,cmap=cm_dark)#圈中测试集样本点
plt.xlabel('花萼长度',fontsize=13)
plt.ylabel('花萼宽度',fontsize=13)
plt.xlim(x1_min,x1_max)
plt.ylim(x2_min,x2_max)
plt.title('鸢尾花SVM二特征分类')
plt.show()
```


鸢尾花SVM二特征分类
花萼宽度
花萼长度
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5

```
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.model_selection  import
train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

SVC之人脸识别

概述

任务：

- 采用支持向量机进行对sklearn自带人脸数据库fetcg_lfw_people进行人脸识别。

目的：

- 人脸数据的读取
- ＰＣＡ降维的应用
- GridSearchCV调参
- ＳＶＣ的应用
- 预测结果分析

数据集说明

fetcg_lfw_people：在 sklearn 中的封装的人脸识别数据集；

faces：字典类型，其中 “target_names” 为每一个人脸样本对应的真实的人的姓名；

数据集中共有 6370张人脸，每张人脸有 125*94 个特征；

lfw_people = fetch_lfw_people()

默认resize参数为0.5，故取出为2914 = 62*47个特征

其中 62 * 47 = 2914，表示每张人脸都是 62 X 47 像素的图片

具体参见官方文档：
https://sklearn.apachecn.org/docs/master/47.html?h=fetch_lfw_people

每个人各自有的图片数量不等，可以通过设置min_faces_per_person的值来确定。
lfw_people = fetch_lfw_people(min_faces_per_person=100, resize=0.4)
即获取的每个人图片不少于100张，并且将尺寸缩小到原来0.4，
即为50X37 = 1850

示例代码：

```
lfw_people = fetch_lfw_people()
print(lfw_people.keys())
# 输出：dict_keys(['data', 'images', 'target', 'target_names', 'DESCR'])

print(lfw_people.data.shape)

print(lfw_people.images.shape)
```

```
dict_keys(['data', 'images', 'target', 'target_names', 'DESCR'])
(6370, 2914)
(6370, 62, 47)
```

数据降维PCA

n_components= 150 #降维的参数，组成元素的数量，即保留下来的特征个数

pca = PCA(n_components=n_components,svd_solver='randomized',whiten=True).fit(X_train)

#使用进行数据模型降维,降成了150

X_train_pca = pca.transform(X_train)

X_test_pca = pca.transform(X_test)

建模

```
#C 是对错误部分的惩罚；gamma 合成点
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005,0.01, 0.1],}
#rbf处理图像较好，C和gamma组合，找出最好的一个组合
clf = GridSearchCV(SVC(kernel='rbf'), param_grid)
print( clf)
#建模
clf = clf.fit(X_train_pca, y_train)
print( clf.best_estimator_  )#最好的模型的信息
```

sklearn.model_selection.GridSearchCV

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

GridSearchCV，它**存在的意义就是自动调参**，只要把参数输进去，就能给出最优化的结果和参数。

GridSearchCV用于系统地遍历多种参数组合，通过交叉验证确定最佳效果参数。

但是**这个方法适合于小数据集**，一旦数据的量级上去了，很难得出结果。

数据预测

```
y_pred = clf.predict(X_test_pca)


#打印预测成绩报告
print( classification_report(y_test,y_pred,target_names=target_names))


#打印预测成绩混淆矩阵
print( confusion_matrix(y_test,y_pred,labels=range(n_classes)))
print( y_test)
print( y_pred)
```

	precision	recall	f1-score	support
Ariel Sharon	0.70	0.64	0.67	22
Colin Powell	0.80	0.86	0.83	56
Donald Rumsfeld	0.84	0.81	0.82	26
George W Bush	0.87	0.90	0.89	133
Gerhard Schroeder	0.81	0.76	0.79	29
Hugo Chavez	0.93	0.81	0.87	16
Tony Blair	0.79	0.75	0.77	40
micro avg	0.83	0.83	0.83	322
macro avg	0.82	0.79	0.80	322
weighted avg	0.83	0.83	0.83	322

support列为每个标签的出现次数
precision 为预测为该人，真正是该人的比率
recall 为 该人正确被预测的比率

F_1值

F_1值是精确度和召回率的调和平均值：

$$\frac{2}{F_1} = \frac{1}{P} + \frac{1}{R}$$

$$F_1 = 2\frac{P \times R}{P + R}$$

```
[[ 14   5   1   2   0   0   0]
 [  1  48   0   5   0   0   2]
 [  0   1  21   3   1   0   0]
 [  3   3   2 120   2   1   2]
 [  0   1   1   1  22   0   4]
 [  1   1   0   1   0  13   0]
 [  1   1   0   6   2   0  30]]
```

测试数据标签

```
[3 3 1 5 3 3 1 6 1 3 6 3 3 6 2 4 4 0 1 3 1 6 3 2 3 3 3 2 3 6 1 3 3 6
 3 3 6 1 1 3 3 3 0 3 1 1 1 5 3 3 3 4 3 5 1 1 6 3 3 3 3 3 1 4 1 1 3 3 3 1
 4 1 0 3 4 6 4 2 2 3 5 5 2 3 3 6 1 3 3 1 2 2 4 2 3 3 4 4 6 3 1 4 1 1 3 3
 3 3 3 3 3 3 1 3 3 3 2 0 3 3 3 6 3 2 3 3 3 0 1 4 3 3 6 1 6 2 0 3 2 1 2 0 5
 4 3 3 5 1 3 3 1 3 0 1 0 0 1 1 3 3 6 3 4 1 5 3 6 1 2 3 3 1 3 5 6 6 5 3 4 0
 3 3 6 1 3 4 4 3 0 5 3 5 6 4 3 1 6 0 3 3 4 3 0 3 6 6 3 3 6 4 1 3 3 4 4 3 6
 6 4 2 0 0 2 3 3 6 3 2 0 1 3 1 3 3 4 1 4 1 3 6 1 1 3 6 3 1 2 3 1 6 0 6 3 3
 1 2 0 4 4 6 3 3 0 1 3 3 5 3 3 3 3 2 6 2 3 3 3 3 6 2 0 6 3 0 3 1 6 1 1 3 3
 6 6 3 1 5 4 3 1 3 3 1 3 3 6 5 3 4 5 3 3 3 2 3 1 2 3]
```

```
[3 3 1 1 3 3 1 6 1 3 6 3 3 6 2 4 6 1 1 3 1 6 3 2 5 3 3 2 1 3 3 3 3 6 2 6 3
 3 3 6 0 1 3 6 2 1 3 1 1 1 5 3 3 3 4 3 5 1 1 1 3 0 3 0 3 1 4 1 1 3 3 3 3 1
 1 1 0 3 4 3 4 2 2 3 3 5 2 1 3 6 1 3 3 1 2 2 4 2 3 3 2 6 4 3 1 4 3 1 1 3 3
 3 3 3 3 3 3 1 4 3 3 3 0 3 3 3 6 3 3 3 3 3 0 1 4 3 3 6 1 6 2 0 3 1 1 4 1 5
 3 3 3 0 1 1 3 3 3 0 1 0 0 1 1 3 3 6 3 4 1 5 3 6 1 2 3 3 1 3 5 6 6 5 3 4 0
 3 2 6 1 3 4 4 3 3 5 3 5 6 6 3 3 6 1 3 3 4 3 1 3 6 6 3 3 6 4 1 3 3 4 4 3 0
 6 6 2 0 0 2 3 3 6 3 2 2 1 3 3 3 0 4 1 4 6 3 3 1 1 3 6 3 1 2 3 1 3 0 6 3 3
 1 2 3 4 4 6 3 3 0 1 3 3 5 6 3 3 3 2 4 3 3 3 3 3 6 2 0 3 3 0 3 6 6 1 1 3 3
 6 3 3 1 5 4 3 1 3 3 1 3 3 6 5 3 4 5 3 3 3 2 3 1 2 4]
```

预测值

SVR california housing

概述

任务：

- 根据给出的california 房价信息，进行房价预测。

目的：

- 数据集数据读取
- 线性支持向量回归模型应用
- RandomizedSearchCV调参
- 预测结果分析

数据说明

7.3.7. California Housing dataset

Data Set Characteristics:

Number of Instances:	20640
Number of Attributes:	8 numeric, predictive attributes and the target
Attribute Information:	• MedInc median income in block • HouseAge median house age in block • AveRooms average number of rooms • AveBedrms average number of bedrooms • Population block population • AveOccup average house occupancy • Latitude house block latitude • Longitude house block longitude

数据获取与观察

```
housing = fetch_california_housing()#获取数据
X = housing['data']#得到数据
y = housing['target']#得到标签
print(housing.feature_names)#显示特征名称
print(housing.DESCR)
print(X.shape)
print(X[0:5])
print(y.shape)
print(y[0:5])
```

```
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/

The target variable is the median house value for California districts.
```

housing.feature_names

housing.DESCR

```
(20640, 8)
[[ 8.32520000e+00  4.10000000e+01  6.98412698e+00  1.02380952e+00
   3.22000000e+02  2.55555556e+00  3.78800000e+01 -1.22230000e+02]
 [ 8.30140000e+00  2.10000000e+01  6.23813708e+00  9.71880492e-01
   2.40100000e+03  2.10984183e+00  3.78600000e+01 -1.22220000e+02]
 [ 7.25740000e+00  5.20000000e+01  8.28813559e+00  1.07344633e+00
   4.96000000e+02  2.80225989e+00  3.78500000e+01 -1.22240000e+02]
 [ 5.64310000e+00  5.20000000e+01  5.81735160e+00  1.07305936e+00
   5.58000000e+02  2.54794521e+00  3.78500000e+01 -1.22250000e+02]
 [ 3.84620000e+00  5.20000000e+01  6.28185328e+00  1.08108108e+00
   5.65000000e+02  2.18146718e+00  3.78500000e+01 -1.22250000e+02]]
(20640,)
[4.526 3.585 3.521 3.413 3.422]
```

X[0:5]

y[0:5]

数据划分与归一化

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)//均值归一化
X_test_scaled = scaler.transform(X_test)
```

StandardScaler类中transform和fit_transform方法有什么区别？

fit_transform方法是fit和transform的结合，fit_transform(X_train) 意思是找出X_train的μ和σ，并应用在X_train上。这时对于X_test，我们就可以直接使用transform方法。因为此时StandardScaler已经保存了X_train的μ和σ 。

建模与分析一 LinearSVR

```
lin_svr = LinearSVR(random_state=42)
lin_svr.fit(X_train_scaled, y_train)


y_pred = lin_svr.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print(mse)
print(np.sqrt(mse))
```

建模与分析一 RandomizedSearchCV

```
param_distributions = {'gamma': reciprocal(0.001, 0.1), 'C': uniform(1, 10)}

rnd_search_cv = RandomizedSearchCV(SVR(), param_distributions, n_iter=10,
verbose=2, cv=3, random_state=42)

rnd_search_cv.fit(X_train_scaled, y_train)

y_pred = rnd_search_cv.best_estimator_.predict(X_train_scaled)

mse = mean_squared_error(y_train, y_pred)

print(np.sqrt(mse))  # 0.572752477078535 6

y_pred = rnd_search_cv.best_estimator_.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)

print(np.sqrt(mse))  # 0.592916838552874
```

小结

- **本次课主要示例了三个支持向量机应用：**
  - ✓鸢尾花
  - ✓人脸识别
  - ✓california　housing房价预测
- **需要掌握技巧：**
  - ✓数据集的了解
  - ✓数据处理
  - ✓建模
  - ✓结果分析
  - ✓可视化