数据科学作业

第一题 numpy 创建数组，数组形状修改结果截图

```
import numpy
x=numpy.arange(20) #创建一个包含个 30 数字的数组 x
print('原始数组：')
print(x) #打印数组 x
print('原始数组的维度是：  {}'.format(x.ndim)) #输出数组的维度
print('原始数组的形状是：  {}'.format(x.shape)) #输出数组的形状
y=x.reshape(2,2,5) #改变数组的形状
print('修改后的数组:')
print(y)
print('修改后的数组的维度是：  {}'.format(y.ndim))
print('修改后的数组的形状是：  {}'.format(y.shape))
```

```
原始数组:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
原始数组的维度是：  1
原始数组的形状是：  (20,)
修改后的数组:
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]]
修改后的数组的维度是：  3
修改后的数组的形状是：  (2, 2, 5)
```

---

numpy.arange()函数用法

- 一个参数时，参数值为终点，起点取默认值 0，步长取默认值 1。
- 两个参数时，第一个参数为起点，第二个参数为终点，步长取默认值 1。
- 三个参数时，第一个参数为起点，第二个参数为终点，第三个参数为步长。其中步长支持小数。

reashpe()函数用法

- 改变数组的形状，并且原始数据不发生变化。但是，reshape()函数中的参数需要满足乘积等于数组中数据总数.
- 通过 reshape 生成的新数组和原始数组公用一个内存，也就是说，假如更改一个数组的元素，另一个数组也将发生改变
- 参考文章：https://numpy.org/doc/stable/reference/generated/numpy.reshape.html

第二题 输出糖尿病数据集所有变量值及其数组形状

Step1 先观察 datasets.load_diabetes()的所有数据

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
#导入数据集
from sklearn.metrics import mean_squared_error
diabetes = datasets.load_diabetes() #载入数据集
```

```
#输出显示整个diabetes数据
print(diabetes)
```

```
{'data': array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
         0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
        -0.06832974, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
         0.00286377, -0.02593034],
       ...,
       [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
        -0.04687948,  0.01549073],
       [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
         0.04452837, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
        -0.00421986,  0.00306441]]), 'target': array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
        68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
        87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
       259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
       128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
       150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
       200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
        42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
        83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
       104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
       173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
       107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
        60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
       197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
        59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
       237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
       143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
       142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
        77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
        78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
       154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
```

Step2 打印输出提示文档

```
#输出显示提示文档
print(diabetes.DESCR)
```

```
Diabetes dataset
================

Notes
-----

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

Data Set Characteristics:

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one year after baseline

  :Attributes:
    :Age:
    :Sex:
    :Body mass index:
    :Average blood pressure:
    :S1:
    :S2:
    :S3:
    :S4:
    :S5:
    :S6:

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times `n_samples` (i.e. the sum of squ
ares of each column totals 1).

Source URL:
http://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion),
407-499.
(http://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

Step3 输出糖尿病数据集所有变量值及其数组形状

```
print('数据集的所有变量值：')
print(diabetes.data) #输出数据集
print('数据集的所有变量值的数组形状为：{}'.format(diabetes.data.shape))
```

```
数据集的所有变量值：
[[ 0.03807591  0.05068012  0.06169621 ... -0.00259226  0.01990842
  -0.01764613]
 [-0.00188202 -0.04464164 -0.05147406 ... -0.03949338 -0.06832974
  -0.09220405]
 [ 0.08529891  0.05068012  0.04445121 ... -0.00259226  0.00286377
  -0.02593034]
 ...
 [ 0.04170844  0.05068012 -0.01590626 ... -0.01107952 -0.04687948
   0.01549073]
 [-0.04547248 -0.04464164  0.03906215 ...  0.02655962  0.04452837
  -0.02593034]
 [-0.04547248 -0.04464164 -0.0730303  ... -0.03949338 -0.00421986
   0.00306441]]
数据集的所有变量值的数组形状为：(442, 10)
```

通过分析说明文档知道共有十个基本变量：年龄，性别，身体指数，平均血压，和六个血清指标，他们包含在 data 中，数组形状为(442,10)

第三题 输出糖尿病数据所有样本真实标签及其数组形状

在第二题的说明文档上可知样本只是标签在 target 中直接打印输出即可

```
print('样本真实标签：')
print(diabetes.target) #输出真实标签
print('样本真实标签的数组形状为：{}'.format(diabetes.target.shape))
```

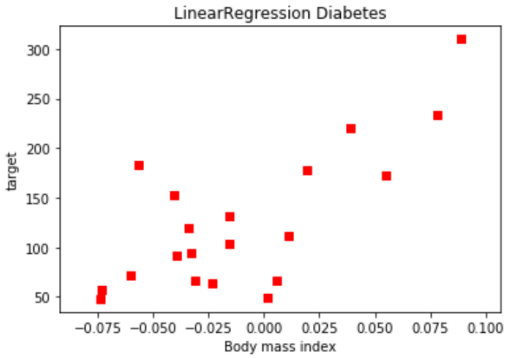
```
样本真实标签：
[151.  75. 141. 206. 135.  97. 138.  63. 110. 310. 101.  69. 179. 185.
 118. 171. 166. 144.  97. 168.  68.  49.  68. 245. 184. 202. 137.  85.
 131. 283. 129.  59. 341.  87.  65. 102. 265. 276. 252.  90. 100.  55.
  61.  92. 259.  53. 190. 142.  75. 142. 155. 225.  59. 104. 182. 128.
  52.  37. 170. 170.  61. 144.  52. 128.  71. 163. 150.  97. 160. 178.
  48. 270. 202. 111.  85.  42. 170. 200. 252. 113. 143.  51.  52. 210.
  65. 141.  55. 134.  42. 111.  98. 164.  48.  96.  90. 162. 150. 279.
  92.  83. 128. 102. 302. 198.  95.  53. 134. 144. 232.  81. 104.  59.
 246. 297. 258. 229. 275. 281. 179. 200. 200. 173. 180.  84. 121. 161.
  99. 109. 115. 268. 274. 158. 107.  83. 103. 272.  85. 280. 336. 281.
 118. 317. 235.  60. 174. 259. 178. 128.  96. 126. 288.  88. 292.  71.
 197. 186.  25.  84.  96. 195.  53. 217. 172. 131. 214.  59.  70. 220.
 268. 152.  47.  74. 295. 101. 151. 127. 237. 225.  81. 151. 107.  64.
 138. 185. 265. 101. 137. 143. 141.  79. 292. 178.  91. 116.  86. 122.
  72. 129. 142.  90. 158.  39. 196. 222. 277.  99. 196. 202. 155.  77.
 191.  70.  73.  49.  65. 263. 248. 296. 214. 185.  78.  93. 252. 150.
  77. 208.  77. 108. 160.  53. 220. 154. 259.  90. 246. 124.  67.  72.
 257. 262. 275. 177.  71.  47. 187. 125.  78.  51. 258. 215. 303. 243.
  91. 150. 310. 153. 346.  63.  89.  50.  39. 103. 308. 116. 145.  74.
  45. 115. 264.  87. 202. 127. 182. 241.  66.  94. 283.  64. 102. 200.
 265.  94. 230. 181. 156. 233.  60. 219.  80.  68. 332. 248.  84. 200.
  55.  85.  89.  31. 129.  83. 275.  65. 198. 236. 253. 124.  44. 172.
 114. 142. 109. 180. 144. 163. 147.  97. 220. 190. 109. 191. 122. 230.
 242. 248. 249. 192. 131. 237.  78. 135. 244. 199. 270. 164.  72.  96.
 306.  91. 214.  95. 216. 263. 178. 113. 200. 139. 139.  88. 148.  88.
 243.  71.  77. 109. 272.  60.  54. 221.  90. 311. 281. 182. 321.  58.
 262. 206. 233. 242. 123. 167.  63. 197.  71. 168. 140. 217. 121. 235.
 245.  40.  52. 104. 132.  88.  69. 219.  72. 201. 110.  51. 277.  63.
 118.  69. 273. 258.  43. 198. 242. 232. 175.  93. 168. 275. 293. 281.
  72. 140. 189. 181. 209. 136. 261. 113. 131. 174. 257.  55.  84.  42.
 146. 212. 233.  91. 111. 152. 120.  67. 310.  94. 183.  66. 173.  72.
  49.  64.  48. 178. 104. 132. 220.  57.]
样本真实标签的数组形状为：(442,)
```

第四题 输出测试数据散点图(学号尾号为 5 输出散点图为红色方形)

Step1 选择第三列 Body mass index 为主要特征进行分析
Step2 划分测试集和训练集（最后二十个数据为数据集）
Step3 给出测试数据的散点图

```
diabetes_X = diabetes.data[:, 2]  #第三列 Body mass index
print(diabetes_X.shape)
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
#绘制测试集的散点图
plt.title(u'LinearRegression Diabetes')   #标题
plt.xlabel(u'Body mass index')                #x轴坐标
plt.ylabel(u'target')          #y轴坐标
plt.scatter(diabetes_X_test, diabetes_y_test, color='red', marker='s')   #画散点图
plt.show()
```

```
(442,)
```



第五题

$diabetes\_X\_trainnp.array(diabetes\_X\_train).reshape(-1,1)$

句的意义？

```
diabetes_X_train = diabetes_X[:-20]
print("原数组形状","\n",diabetes_X_train)
print(diabetes_X_train.shape)
print("\n")
print(np.array(diabetes_X_train).reshape(-1,1).shape)
print("新数组形状","\n",np.array(diabetes_X_train).reshape(-1,1))
```

```
 -0.02991782 -0.0191397  -0.04069594  0.01535029 -0.02452876  0.00133873
  0.06924089 -0.06979687 -0.02991782 -0.046085    0.01858372  0.00133873
 -0.03099563 -0.00405033  0.01535029  0.02289497  0.04552903 -0.04500719
 -0.03315126  0.097264    0.05415152  0.12313149 -0.08057499  0.09295276
 -0.05039625 -0.01159501 -0.0277622   0.05846277  0.08540807 -0.00081689
  0.00672779  0.00888341  0.08001901  0.07139652 -0.02452876 -0.0547075
 -0.03638469  0.0164281 ]
(422,)


(422, 1)
新数组形状
 [[ 0.06169621]
 [-0.05147406]
 [ 0.04445121]
 [-0.01159501]
 [-0.03638469]
 [-0.04069594]
```

diabetes_X_train=np.array(diabetes_X_train).reshape(-1,1)将训练集的数据转化为列数为 1 的新数组，目的是和回归模型中 fit()函数的第一个输入矩阵参数的形状一致

第六题 线性回归回归系数计算

通过调用模型的 coef_方法可以直接得到线性回归的回归系数

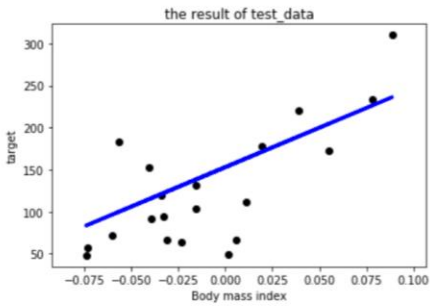
```
#回归训练及预测
diabetes_X_train=np.array(diabetes_X_train).reshape(-1,1)
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
print(regr)
#系数 残差平法和 方差得分
diabetes_X_test=np.array(diabetes_X_test).reshape(-1,1)
print ('Coefficients :\n', regr.coef_)
print ("Residual sum of square: %.2f" %np.mean((regr.predict(diabetes_X_test) - diabetes_y_test) ** 2))
print ("variance score: %.2f" % regr.score(diabetes_X_test, diabetes_y_test))
```

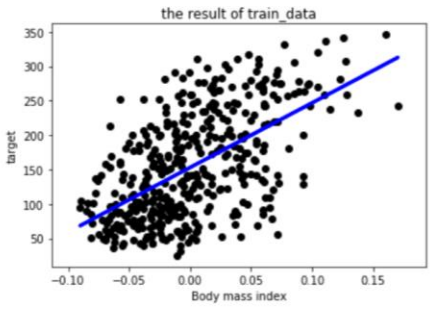
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
Coefficients :
 [938.23786125]
Residual sum of square: 2548.07
variance score: 0.47
```

第七题 线性回归的回归结果折线图及散点图展示

散点图采用 scatter 折线图采用 plot
以下分别展示了训练集和测试集的数据散点图和预测折线图

```
# Plot outputs
diabetes_y_pred1 = regr.predict(diabetes_X_test)
#绘制测试集的散点图
plt.title('the result of test_data')   #标题
plt.xlabel(u'Body mass index')                #x轴坐标
plt.ylabel(u'target')          #y轴坐标
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred1, color='blue', linewidth=3) #画折线图
plt.show()
#绘制训练集集的散点图
diabetes_y_pred2 = regr.predict(diabetes_X_train)
plt.title('the result of train_data')   #标题
plt.xlabel(u'Body mass index')                #x轴坐标
plt.ylabel(u'target')          #y轴坐标
plt.scatter(diabetes_X_train, diabetes_y_train,  color='black')
plt.plot(diabetes_X_train, diabetes_y_pred2, color='blue', linewidth=3) #画折线图
plt.show()
```





第八题 逻辑回归回归系数计算

Step1 导入模型和数据

```
from sklearn.linear_model import LogisticRegression  #导入逻辑回归模型
clf = LogisticRegression()
print(clf)#显示逻辑回归模型
'''
参数 penalty 表示惩罚项（ L1 、 L2 值可选。 L1 向量中各元素绝对值的和，作用是产生少量的特征，
而其他特征都是 0 ，常用于特征选择； L2 向量中各个元素平方之和再开根号，作用是选择较多的特征，使他们都趋近于 0 。）；
C 值的目标函数约束条件： s.t.||w||1<C ，默认值是 0 ， C 值越小，则正则化强度越大。
'''
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

LogisticRegression模型参数介绍参见

https://blog.csdn.net/jark_/article/details/78342644

```
from sklearn.datasets import load_iris   #导入数据集iris
iris = load_iris()  #载入数据集
print(iris.target_names)#显示0，1，2对应的鸢尾花的种类
print(iris.DESCR)#显示相关描述
```

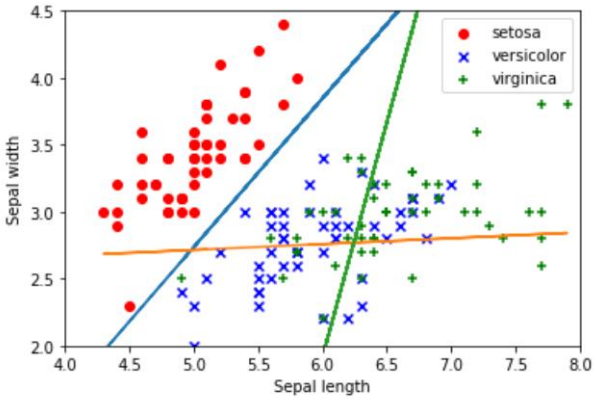
Step2 将数据导入模型进行训练得到回归系数和截距

```
import pandas as pd
data = iris.data[:,:2]#获取花卉两列数据集
label = iris.target
#初始化逻辑回归模型并进行训练， C=1e5 表示目标函数。
X=pd.DataFrame(data)
X.columns=["Sepal length","Sepal width"]
Y=pd.DataFrame(label)
Y.columns=["class"]
Y=Y.loc[:,"class"]
lr = LogisticRegression(C=1e5)
lr.fit(data,label)
print(lr.coef_)#回归系数
print(lr.intercept_)#b坐标
```

```
[[-30.61879527  27.54963779]
 [  0.14041199  -3.21392459]
 [  2.60373147  -0.74348327]]
[ 77.73711825   8.02399007 -14.19811218]
```

Step3 图形化展示散点图和折线图

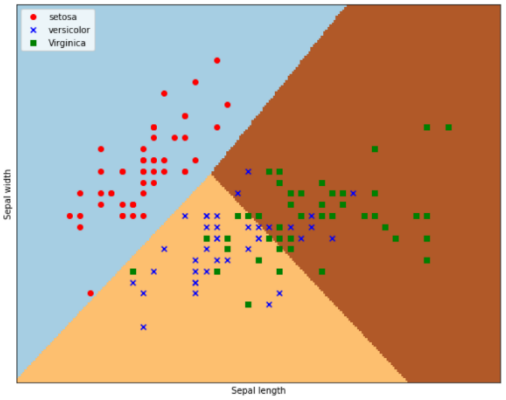
```
#print(lr.score(Y,lr.predict(X)))
fig=plt.figure()
a1=lr.coef_[0][0]
b1=lr.coef_[0][1]
c1=lr.intercept_[0]

a2=lr.coef_[1][0]
b2=lr.coef_[1][1]
c2=lr.intercept_[1]

a3=lr.coef_[2][0]
b3=lr.coef_[2][1]
c3=lr.intercept_[2]
x=X.loc[:,"Sepal length"]
y1=-(c1+a1*x)/b1
y2=-(c2+a2*x)/b2
y3=-(c3+a3*x)/b3
plt.plot(x,y1)
plt.plot(x,y2)
plt.plot(x,y3)
setosa=plt.scatter(X.loc[:,"Sepal length"][Y==0],X.loc[:,"Sepal width"][Y==0], color='red', marker='o', label='setosa')
versicolor=plt.scatter(X.loc[:,"Sepal length"][Y==1],X.loc[:,"Sepal width"][Y==1], color='blue', marker='x', label='versicolor')
virginica=plt.scatter(X.loc[:,"Sepal length"][Y==2], X.loc[:,"Sepal width"][Y==2], color='green', marker='+', label='virginica')
plt.legend((setosa,versicolor,virginica),  ('setosa','versicolor','virginica'))
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.xlim((4, 8))
plt.ylim((2, 4.5))
plt.show()
```



第九题 逻辑回归回归散点图展示

Step1 获取的鸢尾花两列数据，对应为花萼长度和花萼宽度，每个点的坐标就是(x,y).
用 meshgrid 函数生成两个网格矩阵
Step2 调用 predict 函数进行预测
Step3 调用 pcolormesh()函数将 xx、yy 两个网格矩阵和对应的预测结果 Z 绘制在图片上

```
#meshgrid函数生成两个网格矩阵
h = .02
x_min, x_max = data[:, 0].min() - .5, data[:, 0].max() + .5
y_min, y_max = data[:, 1].min() - .5, data[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
'''
获取的鸢尾花两列数据，对应为花萼长度和花萼宽度，每个点的坐标就是 (x,y) 。
先取 X 二维数组的第一列（长度）的最小值、最大值和步长 h （设置为 0.02 ）生成数组，
再取 X 二维数组的第二列（宽度）的最小值、最大值和步长 h 生成数组，
最后用 meshgrid 函数生成两个网格矩阵 xx 和 yy
'''
#调用ravel()函数将xx和yy的两个矩阵转变成一维数组，由于两个矩阵大小相等，因此两个一维数组大小也相等。
#np.c_[xx.ravel(), yy.ravel()]是获取矩阵，即：
Z = lr.predict(np.c_[xx.ravel(), yy.ravel()])
'''
总结：上述操作是把第一列花萼长度数据按h取等分作为行，并复制多行得到xx网格矩阵；再把第二列花萼宽度数据按h取等分，
作为列，并复制多列得到yy网格矩阵；最后将xx和yy矩阵都变成两个一维数组，调用np.c_[]函数组合成一个二维数组进行预测。
'''
Z = Z.reshape(xx.shape)#调用reshape()函数修改形状，将其Z转换为两个特征（长度和宽度），则39501个数据转换为171*231的矩阵
#注意39501个数据由xx, yy的矩阵个数生成
plt.figure(2, figsize=(10,8))
#调用pcolormesh()函数将xx、yy两个网格矩阵和对应的预测结果Z绘制在图片上，可以发现输出为三个颜色区块，分布表示分类的三类区域。
#cmap=plt.cm.Paired表示绘图样式选择Paired主题
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

#绘制散点图
plt.scatter(data[:50,0], data[:50,1], color='red',marker='o', label='setosa')
plt.scatter(data[50:100,0], data[50:100,1], color='blue', marker='x', label='versicolor')
plt.scatter(data[100:,0], data[100:,1], color='green', marker='s', label='Virginica')

plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
plt.legend(loc=2)
plt.show()
```



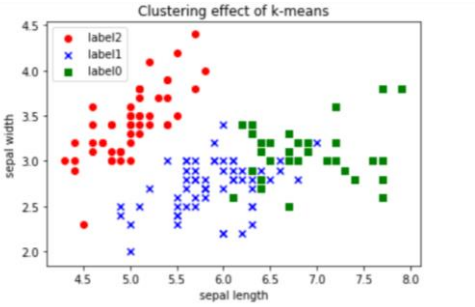
其中散点图代表原来的数据特征及标签，画布的三个区块为预测生成的标签

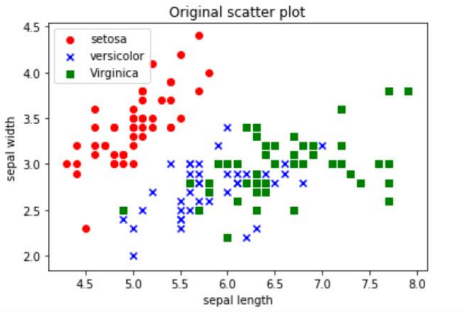
第十题 对鸢尾花数据进行 K-means 聚类，绘制聚类中心为 3 的聚类结果图

算法介绍：k-means 算法是一种基于划分的聚类算法，它以 k 为参数，把 n 个数据对象分成 k 个簇，使簇内具有较高的相似度，而簇间的相似度较低。首先，随机地选择 k 个数据对象，每个数据对象代表一个簇中心，即选择 k 个初始中心；对剩余的每个对象，根据其与各簇中心的相似度（距离），将它赋给与其最相似的簇中心对应的簇；然后重新计算每个簇中所有对象的平均值，作为新的簇中心。

以下采用数据的所有（4 个）特征进行聚类得到结果如下

```
from sklearn.cluster import KMeans #导入K-means聚类模型
from sklearn.datasets import load_iris   #导入数据集iris
iris = load_iris()
X = iris.data[:,:]

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.title('Clustering effect of k-means')   #标题
plt.scatter(x0[:, 0], x0[:, 1], c = "red", marker='o', label='label2')
plt.scatter(x1[:, 0], x1[:, 1], c = "blue", marker='x', label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c = "green", marker='s', label='label0')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)#标签位置
plt.show()

#绘制散点图
plt.scatter(data[:50,0], data[:50,1], color='red',marker='o', label='setosa')
plt.scatter(data[50:100,0], data[50:100,1], color='blue', marker='x', label='versicolor')
plt.scatter(data[100:,0], data[100:,1], color='green', marker='s', label='Virginica')
plt.title('Original scatter plot')   #标题
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)#标签位置
plt.show()
```


Clustering effect of k-means
label2
label1
label0
sepal width
sepal length
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


Original scatter plot
setosa
versicolor
Virginica
sepal width
sepal length
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

拓展创新实验

一 通过均方差评估下不同参数下回归模型的能力

评价模型回归的不同指标
第一，是否预测到了正确或者接近正确的数值--MSE 均方误差(求得结果与标签最大最小值对比)
一种是使用 sklearn 专用的模型评估模块 metrics 里的类 mean_squared_error
另一种是调用交叉验证的类 cross_val_score 并使用里面的 scoring 参数来设置为：neg_mean_squared_error 使用均方误差。
第二，是否拟合到了足够的信息--R 方(误差越小则分子分母表达式返回的结果越小，则 r2 越接近 1)
第一种是直接从 metrics 中导入 r2_score，输入预测值和真实值后打分。
第二种是直接从线性回归 LinearRegression 的接口 score 来进行调用。
第三种是在交叉验证中，输入"r2"来调用。

（1）分析糖尿病的影响因素的线性回归模型

```
print('Mean squared error: %.2f' % mean_squared_error(diabetes_y_test,diabetes_y_pred1))
```

```
Mean squared error: 2548.07
```

（2）分析鸢尾花分类的逻辑回归模型

均方误差分析

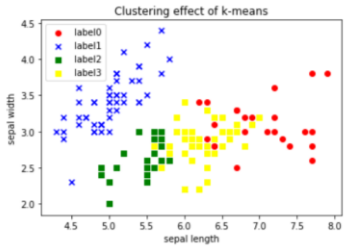
```
iris = load_iris() #载入数据集
DD = iris.data
X = [x[0] for x in DD]
Y = [x[1] for x in DD] #获取花卉两列数据集
from sklearn.linear_model import LogisticRegression #导入逻辑回归模型
clf = LogisticRegression()
x=np.array((X,Y)).T
y=iris.target
clf.fit(x,y)
X_test=X[-20:]
Y_test=Y[-20:]
x_test=np.array((X_test,Y_test)).T
y_test=y[-20:]
y_pred=clf.predict(x_test)
from sklearn.metrics import mean_squared_error
print('Mean squared error: %.2f' % mean_squared_error(y_test,y_pred))
```

```
Mean squared error: 0.20
```

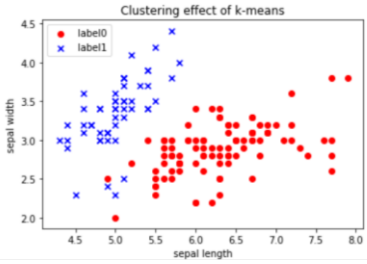
二 不同聚类数量对于结果的区别

对鸢尾花数据进行 K-means 聚类，分别绘制聚类中心为 2、4 的聚类结果图

```
from sklearn.cluster import KMeans #导入K-means聚类模型
from sklearn.datasets import load_iris   #导入数据集iris
iris = load_iris()
X = iris.data[:] ##表示我们只取特征空间中的后两个维度

estimator = KMeans(n_clusters=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]
x3 = X[label_pred == 3]

plt.title('Clustering effect of k-means')   #标题
plt.scatter(x0[:, 0], x0[:, 1], c = "red", marker='o', label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c = "blue", marker='x', label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c = "green", marker='s', label='label2')
plt.scatter(x3[:, 0], x3[:, 1], c = "yellow", marker='s', label='label3')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)#标签位置
plt.show()
```




```
from sklearn.cluster import KMeans #导入K-means聚类模型
from sklearn.datasets import load_iris   #导入数据集iris
iris = load_iris()
X = iris.data[:] ##表示我们只取特征空间中的后两个维度

estimator = KMeans(n_clusters=2)#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]


plt.title('Clustering effect of k-means')   #标题
plt.scatter(x0[:, 0], x0[:, 1], c = "red", marker='o', label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c = "blue", marker='x', label='label1')

plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)#标签位置
plt.show()
```

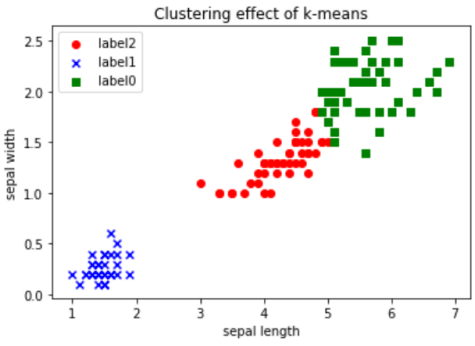


因为 K-means 聚类算法是产生指定类数的聚类结果，错误估计 k 会使算法不能揭示真正的聚类结构。在鸢尾花的分类中，最好的分类类别为 3，这是指定的，因此 K-means 聚类算法需要对分类数量进行明确，否则分类效果会受到影响。

三 使用不同的特征进行特征聚类会有什么结果

如下为选择 petal length in cm 和 petal width in cm 进行聚类的效果
可以看到聚类效果不是很好，这说明选择不同的特征进行聚类所实现的聚类效果是完全不同的，选择正确 的特征会使得聚类更加明确且简洁。

```
from sklearn.cluster import KMeans #导入K-means聚类模型
from sklearn.datasets import load_iris   #导入数据集iris
iris = load_iris()
X = iris.data[:,2:]

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.title('Clustering effect of k-means')   #标题
plt.scatter(x0[:, 0], x0[:, 1], c = "red", marker='o', label='label2')
plt.scatter(x1[:, 0], x1[:, 1], c = "blue", marker='x', label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c = "green", marker='s', label='label0')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)#标签位置
plt.show()
```



四 进行 2007 年前中国男足在亚洲水平的聚类实验

Step1 数据读入
Step2 建立模型并进行训练（抽取日本、巴林和泰国的值作为三个簇的种子）
Step3 输出结果 从聚类结果可以看出中国在亚洲属于第三梯队的水平

```
from matplotlib import pyplot
import scipy as sp
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt
from sklearn.cluster   import KMeans
from scipy import sparse
import pandas as pd
# 数据读入
data = np.loadtxt(r'C:\Users\27675\Desktop\football.txt')
print (data)
X_train = data[:,:]
#pyplot.imshow(X_train, interpolation='nearest')
print(X_train[1], X_train[12] ,X_train[9])
init = np.vstack([ X_train[1], X_train[12] ,X_train[9]])
kmeans = KMeans(n_clusters=3,init=init)
kmeans.fit(X_train)
y_pre = kmeans.predict(X_train)
center = kmeans.cluster_centers_ #获取中心点
print('三组聚类中心分别为：')
print(center)
print(pd.concat((pd.DataFrame(X_train,index=list(['中国','日本','韩国
```

输出结果为

```
三组聚类中心分别为：
[[0.15    0.075   0.16   ]
 [0.528   0.744   0.412  ]
 [1.      0.94    0.40625]]
        2006 世界杯  2010 世界杯  2007 亚洲杯  类别
中国          1.00       1.00       0.50   2
日本          0.30       0.00       0.19   0
韩国          0.00       0.15       0.13   0
伊朗          0.24       0.76       0.25   1
沙特          0.30       0.76       0.06   1
伊拉克         1.00       1.00       0.00   2
卡塔尔         1.00       0.76       0.50   2
阿联酋         1.00       0.76       0.50   2
乌兹别克斯坦       0.70       0.76       0.25   1
泰国          1.00       1.00       0.50   2
越南          1.00       1.00       0.25   2
阿曼          1.00       1.00       0.50   2
巴林          0.70       0.76       0.50   1
朝鲜          0.70       0.68       1.00   1
印尼          1.00       1.00       0.50   2
```

五 对于 matplotliib 的使用--Matplotlib 是 Python 中类似 MATLAB 的绘图工具

说明：ipynb 文件有该部分运行结果

（1）必备知识

通常情况下，我们可以将一副 Matplotlib 图像分成三层结构：

第一层是底层的容器层，主要包括 Canvas、Figure、Axes；

第二层是辅助显示层，主要包括 Axis、Spines、Tick、Grid、Legend、Title 等，该层可通过 set_axis_off()或 set_frame_on(False)等方法设置不显示；

第三层为图像层，即通过 plot、contour、scatter 等方法绘制的图像。

（2）基础知识

1. 在任何绘图之前，我们需要一个 Figure 对象，可以理解成我们需要一张画板才能开始绘图。

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (2,2))
```

figsize 用于设定尺寸

2. 在拥有 Figure 对象之后，在作画前我们还需要轴，没有轴的话就没有绘图基准，所以需要添加 Axes

```
ax = fig.add_subplot(111)
```

对于上面的 fig.add_subplot(111)就是添加 Axes 的，参数的解释的在画板的第 1 行第 1 列的第一个位置生成一个 Axes 对象来准备作画。也可以通过 fig.add_subplot(2, 2, 1)的方式生成 Axes，前面两个参数确定了面板的划分，例如 2， 2 会将整个面板划分成 2 * 2 的方格，第三个参数取值范围是[1, 2*2]表示第几个 Axes。

```
ax.set(xlim=[0.5, 4.5], ylim=[-2, 8], title='An Example Axes',
ylabel='Y-Axis', xlabel='X-Axis') 用于设置坐标尺度标题坐标轴名称
```

注明：Multiple Axes 可以直接生成网格画布，可以一次性生成所有的 Axes

```
fig, axes = plt.subplots(nrows=2, ncols=2)
axes[0,0].set(title='Upper Left')
axes[0,1].set(title='Upper Right')
axes[1,0].set(title='Lower Left')
axes[1,1].set(title='Lower Right')
```

（3）基本绘图 2D

折线图

线：plot()函数画出一系列的点，并且用线将它们连接起来。

```
x = np.linspace(0, np.pi)
y_sin = np.sin(x)
y_cos = np.cos(x)
ax1.plot(x, y_sin)
ax2.plot(x, y_sin, 'go--', linewidth=2, markersize=12)
ax3.plot(x, y_cos, color='red', marker='+', linestyle='dashed')
```

* 通过关键字参数的方式绘图

举例：

```
1 ax.plot('x', 'mean', color='black', data=data_obj)
2 填充两条线之间的颜色
x = np.linspace(0, 10, 200)
data_obj = {'x': x,
            'y1': 2 * x + 1,
            'y2': 3 * x + 1.2,
            'mean': 0.5 * x * np.cos(2*x) + 2.5 * x + 1.1}
ax.fill_between('x', 'y1', 'y2', color='yellow', data=data_obj)
```

散点图

只画点，但是不用线连接起来。

```
x = np.arange(10)
y = np.random.randn(10)
plt.scatter(x, y, color='red', marker='+')
plt.show()
```

条形图

matplotlib 库的 axiss 模块中的 Axes.barh()函数用于制作水平条形图。

Axes.barh(self, y, width, height=0.8, left=None, *, align='center', **kwargs)

在水平或者垂直方向上画线

```
axes[0].axhline(0, color='gray', linewidth=2)
axes[1].axvline(0, color='gray', linewidth=2)
```

绘制条形图例子：

```
np.random.seed(1)
x = np.arange(5)
y = np.random.randn(5)

fig, axes = plt.subplots(ncols=2, figsize=plt.figaspect(1./2))
vert_bars = axes[0].bar(x, y, color='lightblue', align='center')#竖直
horiz_bars = axes[1].barh(x, y, color='lightblue', align='center')#水平
plt.show()
```

直方图

直方图用于统计数据出现的次数或者频率，有多种参数可以调整

Axes.hist(self, x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)用于绘制直方图

直方图举例：

```
np.random.seed(19680801)
n_bins = 10
x = np.random.randn(1000, 3)
```

```
fig, axes = plt.subplots(nrows=2, ncols=2)
ax0, ax1, ax2, ax3 = axes.flatten()
colors = ['red', 'tan', 'lime']

ax0.hist(x, n_bins, density=True, histtype='bar', color=colors,
label=colors)
ax0.legend(prop={'size': 10})
ax0.set_title('bars with legend')

ax1.hist(x, n_bins, density=True, histtype='barstacked')
ax1.set_title('stacked bar')

ax2.hist(x, histtype='barstacked', rwidth=0.9)

ax3.hist(x[:, 0], rwidth=0.9)
ax3.set_title('different sample sizes')

fig.tight_layout()
plt.show()
```

参数中 density 控制 Y 轴是概率还是数量。histtype 控制着直方图的样式，默认是 ‘bar’，对于多个条形时就相邻的方式呈现‘barstacked’ 就是叠在一起 rwidth 控制着宽度，这样可以空出一些间隙

饼图

pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1, counterclock=True, wedgeprops=None, textprops=None, center=0, 0, frame=False, rotatelabels=False, *, normalize=None, data=None) 用于绘制饼图

例子：

```
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, (ax1, ax2) = plt.subplots(2)
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True)
ax1.axis('equal')
ax2.pie(sizes, autopct='%1.2f%%', shadow=True, startangle=90,
explode=explode,
    pctdistance=1.12)
ax2.axis('equal')
ax2.legend(labels=labels, loc='upper right')

plt.show()
```

饼图自动根据数据的百分比画饼.labels 是各个块的标签。autopct=%1.1f%%表

示格式化百分比精确输出，explode，突出某些块，不同的值突出的效果不一样。
pctdistance=1.12 百分比距离圆心的距离，默认是 0.6.

箱形图

boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None, widths=None, patch_artist=None, bootstrap=None, usermedians=None, conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None, showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None, meanprops=None, capprops=None, whiskerprops=None, manage_ticks=True, autorange=False, zorder=None, *, data=None)用于绘制箱线图

举例：

```
fig, (ax1, ax2) = plt.subplots(2)
data=np.random.randn(10, 3)
print(data)
data2=np.random.randn(15, 3)
ax1.boxplot(data)
ax2.boxplot(data2, vert=False) #控制方向
```

vert=False 代表水平方向，否则为竖直方向

等高线（轮廓图）

需要描绘边界的时候，就会用到轮廓图，机器学习用的决策边界也常用轮廓图来绘画

.contourf(*args, data=None, **kwargs) 用于绘制等高线

例子：

```
fig, (ax1, ax2) = plt.subplots(2)
x = np.arange(-5, 5, 0.1)
y = np.arange(-5, 5, 0.1)
xx, yy = np.meshgrid(x, y, sparse=True)
z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)
ax1.contourf(x, y, z)
ax2.contour(x, y, z)
```

上面画了两个一样的轮廓图，contourf 会填充轮廓线之间的颜色。数据 x，y，z 通常是具有相同形状的二维矩阵。x，y 可以为一维向量，但是必需有 z.shape = (y.n, x.n) ，这里 y.n 和 x.n 分别表示 x、y 的长度。Z 通常表示的是距离 X-Y 平面的距离，传入 X、Y 则是控制了绘制等高线的范围。

（4）布局、图例说明、边界

区间上下限

当绘画完成后，会发现 X、Y 轴的区间是会自动调整的，并不是跟我们传入的 X、Y 轴数据中的最值相同。为了调整区间我们使用下面的方式(任意选择一个)

```
* ax.set_xlim([xmin, xmax])    #设置 X 轴的区间
* ax.set_ylim([ymin, ymax])    #Y 轴区间
* ax.axis([xmin, xmax, ymin, ymax])    #X、Y 轴区间
* ax.set_ylim(bottom=-10)      #Y 轴下限
* ax.set_xlim(right=25)        #X 轴上限
```

图例说明

如果我们在一个 Axes 上做多次绘画，就需要添加图例说明
使用方法：
在绘图末尾添加标签：

```
ax.plot([1, 2, 3, 4], [10, 20, 25, 30], label='Philadelphia')
ax.set(ylabel='Temperature (deg C)', xlabel='Time', title='A tale of two cities')
```

然后使用 ax.legend()显示图例说明，对于 legend 还是传入参数，控制图例说明显示的位置

区间分段

默认情况下，绘图结束之后，Axes 会自动的控制区间的分段
例子：

```
data = [('apples', 2), ('oranges', 3), ('peaches', 1)]
fruit, value = zip(*data)

fig, (ax1, ax2) = plt.subplots(2)
x = np.arange(len(fruit))
ax1.bar(x, value, align='center', color='gray')
ax2.bar(x, value, align='center', color='gray')

ax2.set(xticks=x, xticklabels=fruit)

#ax.tick_params(axis='y', direction='inout', length=10) #修改 ticks 的方向以及长度
plt.show()
```

上面不仅修改了 X 轴的区间段，并且修改了显示的信息为文本。

布局

当我们绘画多个子图时，就会有一些美观的问题存在，例如子图之间的间隔，子图与画板的外边间距以及子图的内边距，下面说明这个问题：

```
fig, axes = plt.subplots(2, 2, figsize=(9, 9))
fig.subplots_adjust(wspace=0.5, hspace=0.3,
                    left=0.125, right=0.9,
                    top=0.9,    bottom=0.1)

#fig.tight_layout() #自动调整布局，使标题之间不重叠
plt.show()
```

通过 fig.subplots_adjust()我们修改了子图水平之间的间隔 wspace=0.5，垂直方向上的间距 hspace=0.3，左边距 left=0.125 等等，这里数值都是百分比的。以 [0, 1] 为区间，选择 left、right、bottom、top 注意 top 和 right 是 0.9 表示上、右边距为百分之 10。不确定如果调整的时候，fig.tight_layout()是一个很好的选择。之前说到了内边距，内边距是子图的，也就是 Axes 对象，所以这样使用 ax.margins(x=0.1, y=0.1)，当值传入一个值时，表示同时修改水平和垂直方向的内边距。

轴相关

改变边界的位置，去掉四周的边框：

```
fig, ax = plt.subplots()
ax.plot([-2, 2, 3, 4], [-10, 20, 25, 5])
ax.spines['top'].set_visible(False)      #顶边界不可见
ax.xaxis.set_ticks_position('bottom')  # ticks 的位置为下方，分上下的。
ax.spines['right'].set_visible(False)   #右边界不可见
ax.yaxis.set_ticks_position('left')

###### "outward"
###### 移动左、下边界离 Axes 10 个距离
#ax.spines['bottom'].set_position(('outward', 10))
#ax.spines['left'].set_position(('outward', 10))

###### "data"
###### 移动左、下边界到 (0, 0) 处相交
ax.spines['bottom'].set_position(('data', 0))
ax.spines['left'].set_position(('data', 0))

###### "axes"
###### 移动边界，按 Axes 的百分比位置
#ax.spines['bottom'].set_position(('axes', 0.75))
#ax.spines['left'].set_position(('axes', 0.3))

plt.show()
```