

实验报告

实验报告

线性回归验证实验
岭回归算法验证实验
Lasso回归算法验证实验
弹性网络算法验证实验
回归算法来训练模型预测儿童身高
逻辑回归算法预测考试是否能够及格
朴素贝叶斯算法对中文邮件进行分类
决策树算法验证实验
判断学员的Python水平
支持向量机算法对手写数字图像进行分类
KNN算法判断交通工具
KMeans聚类算法压缩图像颜色
分层聚类算法验证实验
DBSCAN算法验证实验
协同过滤算法
使用关联规则分析演员关系
使用交叉验证与网格搜索，进行支持向量机分类模型的参数学习
对我国人工智能或机器学习应用研究的现状与趋势介绍，以及对自己将来职业规划感想等

线性回归验证实验

```
1 # 线性回归模型的应用
2 from sklearn import linear_model
3 regression=linear_model.LinearRegression() # 创建线性回归模型
4 x=[[3],[8]] # 观察值的x坐标，是一个二维的数组
5 y=[1,2] # 观察值的y坐标，是一个一维数组
6 regression.fit(X,y) # 拟合
```

LinearRegression()

```
1 regression.intercept_ # 截距
```

0.400000000000000013

```
1 regression.coef_ # 斜率，回归系数
2 # 反映x对y影响的大小
```

array([0.2])

- `np.random.randint(随机最小值, 随机最大值, [生成多少行, 多少列])`
- 使用以上生成10个随机数据，用来测试

```
1 # regression.predict([[6]])
2 unknown=np.random.randint(0,100,[10,1])
3 regression.predict(unknown)
```

- 结果截图：

```
In [4]: # 线性回归模型的应用
from sklearn import linear_model
import numpy as np
regression=linear_model.LinearRegression() # 创建线性回归模型
X=[[3],[8]] # 观察值的X坐标, 是一个二维的数组
y=[1,2] # 观察值的y坐标, 是一个一维数组
regression.fit(X,y) # 拟合
```

```
Out[4]: LinearRegression()
```

```
In [5]: regression.intercept_ # 截距
```

```
Out[5]: 0.40000000000000013
```

```
In [6]: regression.coef_ # 斜率, 回归系数
        # 反映x对y影响的大小
```

```
Out[6]: array([0.2])
```

```
In [7]: # regression.predict([[6]])
unknown=np.random.randint(0,100,[10,1])
regression.predict(unknown)
```

预测结果

```
Out[7]: array([ 4.4,  7.2,  3. , 12.2, 15.8,  7.8, 16. , 15.2,  2.4,  1.4])
```

岭回归算法验证实验

- 岭回归算法实验与线性回归算法的实验数据是一样的, 两者只有模型的不同, 但是在一定情况下两者可以转化。
- 岭回归通过在代价函数后面加上一个对参数的约束项来防止过拟合, 其中约束项系数 α 数值越大, 特征对结果的影响就越小。

约束项系数为10

```
1 from sklearn.linear_model import Ridge
2 import numpy as np
3 ridgeRegression=Ridge(alpha=10) # 创建岭回归模型, 设置约束项系数为10
4 X=[[3],[8]]
5 y=[1,2]
6 ridgeRegression.fit(X,y) # 拟合
```

```
Ridge(alpha=10)
```

```
1 unknown=np.random.randint(0,100,[10,1])
2 ridgeRegression.predict(unknown) # 预测
```

```
array([7. , 9.88888889, 7.77777778, 9.22222222, 7.66666667,
       8.44444444, 8. , 9.11111111, 9. , 9.77777778])
```

```
1 # 查看回归系数
2 ridgeRegression.coef_
```

```
array([0.11111111])
```

```
1 # 和截距
2 ridgeRegression.intercept_
```

0.8888888888888888

- 结果截图:

```
In [8]: from sklearn.linear_model import Ridge
import numpy as np
ridgeRegression=Ridge(alpha=10) # 创建岭回归模型, 设置约束项系数为10
X=[[3],[8]]
y=[1,2]
ridgeRegression.fit(X,y) # 拟合
```

Out[8]: Ridge(alpha=10)

```
In [9]: unknown=np.random.randint(0,100,[10,1])
ridgeRegression.predict(unknown) # 预测
```

Out[9]: array([7. , 9.88888889, 7.77777778, 9.22222222, 7.66666667,
8.44444444, 8. , 9.11111111, 9. , 9.77777778])

```
In [10]: # 查看回归系数
ridgeRegression.coef_
```

Out[10]: array([0.11111111])

```
In [11]: # 和截距
ridgeRegression.intercept_
```

Out[11]: 0.8888888888888888

约束项是1

```
1 from sklearn.linear_model import Ridge
2 ridgeRegression=Ridge(alpha=1.0)
3 x=[[3],[8]]
4 y=[1,2]
5 ridgeRegression.fit(X,y) # 拟合
```

Out[9]:Ridge()

```
1 ridgeRegression.predict([[6]]) # 预测
```

Out[10]:array([1.59259259])

```
1 # 查看回归系数
2 ridgeRegression.coef_
```

Out[11]:array([0.18518519])

```
1 # 和截距
2 ridgeRegression.intercept_
```

Out[12]:0.4814814814814816

- 结果截图:

```
In [9]: from sklearn.linear_model import Ridge
ridgeRegression=Ridge(alpha=1.0)
X=[[3],[8]]
y=[1,2]
ridgeRegression.fit(X,y) # 拟合
```

Out[9]: Ridge()

```
In [10]: ridgeRegression.predict([[6]]) # 预测
```

Out[10]: array([1.59259259])

```
In [11]: # 查看回归系数
ridgeRegression.coef_
```

Out[11]: array([0.18518519])

```
In [12]: # 和截距
ridgeRegression.intercept_
```

Out[12]: 0.4814814814814816

约束项是0

- 当约束项为0的时候，我们比较和线性回归验证函数的数据发现，这个时候的岭回归等价于线性回归

```
1 from sklearn.linear_model import Ridge
2 ridgeRegression=Ridge(alpha=0)
3 x=[[3],[8]]
4 y=[1,2]
5 ridgeRegression.fit(x,y) # 拟合
```

Out[13]:Ridge(alpha=0)

```
1 ridgeRegression.predict([[6]]) # 预测
```

Out[14]:array([1.6])

```
1 # 查看回归系数
2 ridgeRegression.coef_
```

Out[15]:array([0.2])

```
1 # 和截距
2 ridgeRegression.intercept_
```

Out[16]:0.40000000000000013

- 结果截图:

```
In [13]: from sklearn.linear_model import Ridge
         ridgeRegression=Ridge(alpha=0)
         X=[[3],[8]]
         y=[1,2]
         ridgeRegression.fit(X,y) # 拟合
```

Out[13]: Ridge(alpha=0)

```
In [14]: ridgeRegression.predict([[6]]) # 预测
```

Out[14]: array([1.6])

```
In [15]: # 查看回归系数
         ridgeRegression.coef_
```

Out[15]: array([0.2])

```
In [16]: # 和截距
         ridgeRegression.intercept_
```

Out[16]: 0.40000000000000013

指定约束项的范围

```
1 import numpy as np
2 from sklearn.linear_model import RidgeCV
3 X=[[3],[8]]
4 y=[1,2]
5 reg=RidgeCV(alphas=np.arange(0.001,10)) # 指定 $\alpha$ 参数的范围，不能是负数和0，每次递增为1
6 reg.fit(X,y) # 拟合
```

Out[17]:RidgeCV(alphas=array([1.000e-03, 1.001e+00, 2.001e+00, 3.001e+00, 4.001e+00, 5.001e+00, 6.001e+00, 7.001e+00, 8.001e+00, 9.001e+00]))

```
1 reg.alpha_ # 最佳数值，拟合之后这个值才可以使用
```

Out[18]:2.001

```
1 reg.predict([[6]]) # 预测
```

Out[19]:array([1.58620095])

- 结果截图:

```
In [17]: import numpy as np
from sklearn.linear_model import RidgeCV
X=[[3],[8]]
y=[1,2]
reg=RidgeCV(alphas=np.arange(0.001,10)) # 指定  $\alpha$  参数的范围, 不能是负数
reg.fit(X,y) # 拟合
```

```
Out[17]: RidgeCV(alphas=array([1.000e-03, 1.001e+00, 2.001e+00, 3.001e+00,
4.001e+00, 5.001e+00,
6.001e+00, 7.001e+00, 8.001e+00, 9.001e+00]))
```

```
In [18]: reg.alpha_ # 最佳数值, 拟合之后这个值才可以使用
```

```
Out[18]: 2.001
```

```
In [19]: reg.predict([[6]]) # 预测
```

```
Out[19]: array([1.58620095])
```

Lasso回归算法验证实验

惩罚系数是3.0

```
1 from sklearn.linear_model import Lasso
2 x=[[3],[8]]
3 y=[1,2]
4
5 reg=Lasso(alpha=3.0) # 惩罚系数是3.0
6 reg.fit(X,y) # 拟合
```

```
Out[20]:Lasso(alpha=3.0)
```

```
1 reg.coef_ # 查看系数
```

```
Out[21]:array([0.])
```

```
1 reg.intercept_ # 截距
```

```
Out[22]:1.5
```

```
1 reg.predict([[6]])
```

```
Out[23]:array([1.5])
```

- 结果截图:

```
In [20]: from sklearn.linear_model import Lasso
X=[[3],[8]]
y=[1,2]

reg=Lasso(alpha=3.0) # 惩罚系数是3.0
reg.fit(X,y) # 拟合
```

Out[20]: Lasso(alpha=3.0)

```
In [21]: reg.coef_ # 查看系数
```

Out[21]: array([0.])

```
In [22]: reg.intercept_ # 截距
```

Out[22]: 1.5

```
In [23]: reg.predict([[6]])
```

Out[23]: array([1.5])

弹性网络算法验证实验

alpha=1.0,l1_ratio=0.7

```
1 from sklearn.linear_model import ElasticNet
2 reg=ElasticNet(alpha=1.0,l1_ratio=0.7)
3 x=[[3],[8]]
4 y=[1,2]
5 # 拟合
6 reg.fit(X,y)
```

Out[1]:ElasticNet(l1_ratio=0.7)

```
1 # 预测
2 reg.predict([[6]])
```

Out[2]:array([1.54198473])

```
1 reg.coef_
```

Out[3]:array([0.08396947])

```
1 reg.intercept_
```

Out[4]:1.0381679389312977

- 结果截图:

```
In [1]: from sklearn.linear_model import ElasticNet
reg=ElasticNet(alpha=1.0,ll_ratio=0.7)
X=[[3],[8]]
y=[1,2]
reg.fit(X,y)
```

```
Out[1]: ElasticNet(ll_ratio=0.7)
```

```
In [2]: reg.predict([[6]])
```

```
Out[2]: array([1.54198473])
```

```
In [3]: reg.coef_
```

```
Out[3]: array([0.08396947])
```

```
In [4]: reg.intercept_
```

```
Out[4]: 1.0381679389312977
```

回归算法来训练模型预测儿童身高

- 首先我们先导入数据，并且使用sklearn中的模型来训练数据，这里值得注意的是，我们设定年龄在18岁之后身高便不随年龄增长而变高。

```
1 import copy
2 import numpy as np
3 from sklearn import linear_model
4
5 # 训练数据，每一行表示一个样本，包含的信息分别为：
6 # 儿童年龄,性别（0女1男）
7 # 父亲、母亲、祖父、祖母、外祖父、外祖母的身高
8 x = np.array([[1, 0, 180, 165, 175, 165, 170, 165],
9               [3, 0, 180, 165, 175, 165, 173, 165],
10              [4, 0, 180, 165, 175, 165, 170, 165],
11              [6, 0, 180, 165, 175, 165, 170, 165],
12              [8, 1, 180, 165, 175, 167, 170, 165],
13              [10, 0, 180, 166, 175, 165, 170, 165],
14              [11, 0, 180, 165, 175, 165, 170, 165],
15              [12, 0, 180, 165, 175, 165, 170, 165],
16              [13, 1, 180, 165, 175, 165, 170, 165],
17              [14, 0, 180, 165, 175, 165, 170, 165],
18              [17, 0, 170, 165, 175, 165, 170, 165]])
19
20 # 儿童身高，单位：cm
21 y = np.array([60, 90, 100, 110, 130, 140, 150, 164, 160, 163, 168])
22
23 # 创建线性回归模型
24 lr=linear_model.LinearRegression()
25 # 拟合
26 lr.fit(X,y)
```


Out[5]:LinearRegression()

- 输入测试数据

```
1 # 待测的未知数据，其中每个分量的含义和训练数据相同
2 x = np.array([[10, 0, 180, 165, 175, 165, 170, 165],
3               [17, 1, 173, 153, 175, 161, 170, 161],
4               [34, 0, 170, 165, 170, 165, 170, 165]])
5
6 for item in x:
7     # 为不改变原始数据，进行深复制，并假设超过18岁以后就不再长高了
8     # 对于18岁以后的年龄，返回18岁时的身高
9     item1 = copy.deepcopy(item)
10    if item1[0] > 18:
11        item1[0] = 18
12    print(item, ': ', lr.predict(item1.reshape(1,-1)))
```

[10 0 180 165 175 165 170 165]: [140.56153846]

[17 1 173 153 175 161 170 161]: [158.41]

[34 0 170 165 170 165 170 165]: [176.03076923]

- 我们的输出结果是测试数据：预测身高
- 结果截图

```
        [17, 0, 170, 165, 175, 165, 170, 165]])

# 儿童身高，单位：cm
y = np.array([60, 90, 100, 110, 130, 140, 150, 164, 160, 163, 168])

# 创建线性回归模型
lr=linear_model.LinearRegression()
# 拟合
lr.fit(X,y)
```

Out[5]: LinearRegression()

```
In [6]: # 待测的未知数据，其中每个分量的含义和训练数据相同
x = np.array([[10, 0, 180, 165, 175, 165, 170, 165],
               [17, 1, 173, 153, 175, 161, 170, 161],
               [34, 0, 170, 165, 170, 165, 170, 165]])

for item in x:
    # 为不改变原始数据，进行深复制，并假设超过18岁以后就不再长高了
    # 对于18岁以后的年龄，返回18岁时的身高
    item1 = copy.deepcopy(item)
    if item1[0] > 18:
        item1[0] = 18
    print(item, ': ', lr.predict(item1.reshape(1,-1)))

[ 10  0 180 165 175 165 170 165] : [140.56153846]
[ 17  1 173 153 175 161 170 161] : [158.41]
[ 34  0 170 165 170 165 170 165] : [176.03076923]
```

逻辑回归算法预测考试是否能够及格

随机预测

- 随机预测的特点在于使用了 `random` 函数，模拟数据的随机性

```

1 import numpy as np
2 from sklearn.linear_model import LogisticRegression
3 import matplotlib.pyplot as plt
4
5 # 构造测试数据
6 x=np.array([[i] for i in range(30)])
7 y=np.array([0]*15+[1]*15)
8
9 # 人为修改部分样本的值
10 y[np.random.randint(0,15,3)] = 1
11 y[np.random.randint(15,30,4)] = 0
12 print(y[:15])
13 print(y[15:])

```

```

[1 0 0 0 0 0 0 0 0 0 0 1 0 0 1]
[1 1 1 1 1 1 0 1 1 1 1 0 0 1 1]

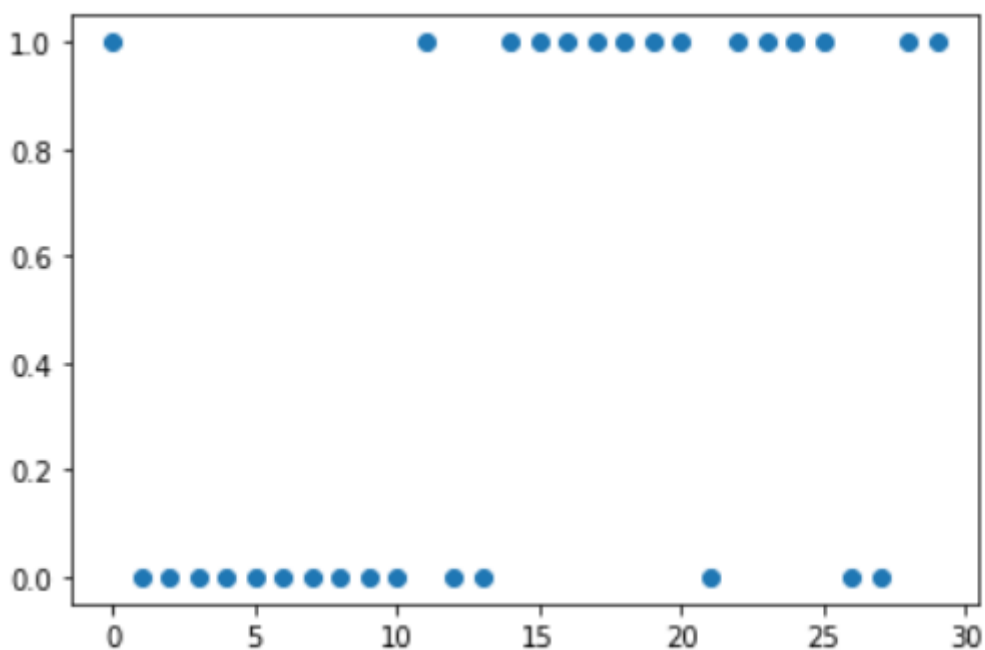
```

```

1 # 根据原始数据绘制散点图
2 plt.scatter(x,y)

```

Out[8]:<matplotlib.collections.PathCollection at 0x244ae0fd808>



```

1 # 创建并训练逻辑回归模型
2 reg=LogisticRegression('l2',C=3.0)
3 reg.fit(X,y)

```

Out[9]:LogisticRegression(C=3.0)

```

1 # 对未知数据进行预测
2 print(reg.predict([[5],[19]]))

```

```
[0 1]
```

```

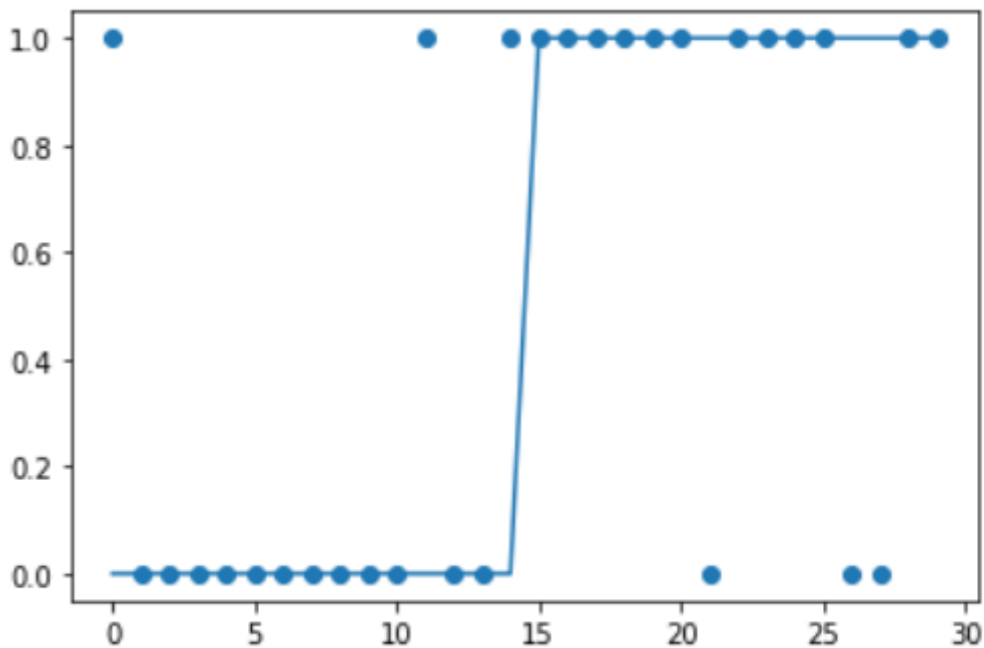
1 # 未知数据属于某一类别的概率
2 print(reg.predict_proba([[5],[19]]))

```

```
[[0.78130853 0.21869147]
 [0.35362437 0.64637563]]
```

- 把原始数据散点图和预测结果散点图合并一起画出对比

```
1 # 对原始观察点进行预测
2 yy=reg.predict(X)
3 # 根据原始数据绘制散点图
4 plt.scatter(X,y)
5 # 根据预测结果绘制折线图
6 plt.plot(X,yy)
7 plt.show()
```



使用逻辑回归模型算法预测考试是否能及格

```
1 from sklearn.linear_model import LogisticRegression
2
3 # 复习情况，格式为(时长,效率)，其中时长单位为小时
4 # 效率为[0,1]之间的小数，数值越大表示效率越高
5 X_train = [(0,0), (2,0.9), (3,0.4), (4,0.9), (5,0.4), (6,0.4), (6,0.8),
6             (6,0.7), (7,0.2), (7.5,0.8), (7,0.9), (8,0.1), (8,0.6), (8,0.8)]
7 # 0表示不及格，1表示及格
8 y_train = [0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]
9
10 # 创建并训练逻辑回归模型
11 reg=LogisticRegression()
12 reg.fit(X_train,y_train)
```

Out[13]:LogisticRegression()

```
1 # 测试模型
2 X_test = [(3,0.9), (8,0.5), (7,0.2), (4,0.5), (4,0.7)]
3 y_test = [0, 1, 0, 0, 1]
4 score = reg.score(X_test, y_test)
5
```

```
6 # 预测并输出预测结果
7 learning = [(8, 0.9)]
8 result = reg.predict_proba(learning)
9 msg = '''模型得分: {0}
10 复习时长为: {1[0]}, 效率为: {1[1]}
11 您不及格的概率为: {2[0]}
12 您及格的概率为: {2[1]}
13 综合判断, 您会: {3}'''.format(score, learning[0], result[0], '不及格' if
14 result[0][0]>0.5 else '及格')
15 print(msg)
```

综合判断，您会：及格

- **结果截图**

```
Out[13]: LogisticRegression()
```

```
In [14]: # 测试模型
X_test = [(3, 0.9), (8, 0.5), (7, 0.2), (4, 0.5), (4, 0.7)]
y_test = [0, 1, 0, 0, 1]
score = reg.score(X_test, y_test)

# 预测并输出预测结果
learning = [(8, 0.9)]
result = reg.predict_proba(learning)
msg = '''模型得分: {0}
复习时长为: {1[0]}, 效率为: {1[1]}
您不及格的概率为: {2[0]}
您及格的概率为: {2[1]}
综合判断, 您会: {3}'''.format(score, learning[0], result[0], '不及格' if result[0][0]>0.5 else '及格')
print(msg)
```

模型得分: 0.6
复习时长为: 8, 效率为: 0.9
您不及格的概率为: 0.18982398713996873
您及格的概率为: 0.8101760128600313
综合判断, 您会: 及格

朴素贝叶斯算法对中文邮件进行分类

程序逻辑:

1. 首先是获取邮箱中的所有词语，并且过滤掉所有的干扰字符和无效字符，并且把长度为1的词也过滤掉，得到有效的词语。

```

1  # 获取每一封邮件中的所有词语
2  def getWordsFromFile(txtFile):
3      words = []
4      # 所有存储邮件文本内容的记事本文件都使用UTF8编码
5      with open(txtFile, encoding='utf8') as fp:
6          for line in fp:
7              # 遍历每一行，删除两端的空白字符
8              line = line.strip()
9              # 过滤干扰字符或无效字符
10             line = sub(r'[\s0-9、。、！~\*]', '', line)
11             # 分词
12             line = cut(line)
13             # 过滤长度为1的词
14             line = filter(lambda word: len(word)>1, line)
15             # 把本行文本预处理得到的词语添加到words列表中

```

```

16         words.extend(line)
17     # 返回包含当前邮件文本中所有有效词语的列表
18     return words

```

2. 存放所有的单词,返回出现次数最多的前600个单词, 这里我使用的是绝对路径。

```

1  # 存放所有文件中的单词
2  # 每个元素是一个子列表, 其中存放一个文件中的所有单词
3  allWords = []
4  def getTopNWords(topN):
5      # 按文件编号顺序处理当前文件夹中所有记事本文件
6      # 训练集中共151封邮件内容, 0.txt到126.txt是垃圾邮件内容, 127.txt到150.txt为正常
      # 邮件内容
7      txtFiles = ['D:\\workspaces\\AI\\ip\\人工智能\\数据集\\贝叶斯中文邮件分类\\'+
8                  str(i) + '.txt' for i in range(151)]
9      # 获取训练集中所有邮件中的全部单词
10     for txtFile in txtFiles:
11         allWords.append(getWordsFromFile(txtFile))
12     # 获取并返回出现次数最多的前topN个单词
13     freq = Counter(chain(*allWords))
14     return [w[0] for w in freq.most_common(topN)]
15
16 # 全部训练集中出现次数最多的前600个单词
topWords = getTopNWords(600)

```

3. 获取特征向量, 计算前600个单词在每一个邮件中出现的频率,并且贴上标签

```

1  # 获取特征向量, 前600个单词的每个单词在每个邮件中出现的频率
2  vectors = []
3  for words in allWords:
4      temp = list(map(lambda x: words.count(x), topWords))
5      vectors.append(temp)
6
7  vectors = array(vectors)
8  # 训练集中每个邮件的标签, 1表示垃圾邮件, 0表示正常邮件
9  labels = array([1]*127 + [0]*24)

```

4. 训练模型并且预测测试数据内容

```

1  # 创建模型, 使用已知训练集进行训练
2  model = MultinomialNB()
3  model.fit(vectors, labels)
4
5  def predict(txtFile):
6      # 获取指定邮件文件内容, 返回分类结果
7      words = getWordsFromFile(txtFile)
8      currentVector = array(tuple(map(lambda x: words.count(x),
9                                      topWords)))
10     result = model.predict(currentVector.reshape(1, -1))[0]
11     print(model.predict_proba(currentVector.reshape(1, -1)))
12     return '垃圾邮件' if result==1 else '正常邮件'
13

```

```
14 # 151.txt至155.txt为测试邮件内容
15 for mail in ('D:\\workspaces\\AI\\ip\\\\人工智能\\数据集\\贝叶斯中文邮件分类\\'+
16             str(i) + '.txt' for i in range(151, 156)):
17     print(mail, predict(mail), sep=':')
```

全部代码展示

```

1 from re import sub
2 from os import listdir
3 from collections import Counter
4 from itertools import chain
5 from numpy import array
6 from jieba import cut
7 from sklearn.naive_bayes import MultinomialNB
8
9 # 获取每一封邮件中的所有词语
10 def getWordsFromFile(txtFile):
11     words = []
12     # 所有存储邮件文本内容的记事本文件都使用UTF8编码
13     with open(txtFile, encoding='utf8') as fp:
14         for line in fp:
15             # 遍历每一行，删除两端的空白字符
16             line = line.strip()
17             # 过滤干扰字符或无效字符
18             line = sub(r'[\.\!\~\*]', '', line)
19             # 分词
20             line = cut(line)
21             # 过滤长度为1的词
22             line = filter(lambda word: len(word)>1, line)
23             # 把本行文本预处理得到的词语添加到words列表中
24             words.extend(line)
25     # 返回包含当前邮件文本中所有有效词语的列表
26     return words
27
28 # 存放所有文件中的单词
29 # 每个元素是一个子列表，其中存放一个文件中的所有单词
30 allwords = []
31 def getTopNWords(topN):
32     # 按文件编号顺序处理当前文件夹中所有记事本文件
33     # 训练集中共151封邮件内容，0.txt到126.txt是垃圾邮件内容，127.txt到150.txt为正常
    邮件内容
34     txtFiles = ['D:\\workspaces\\AI\\ip\\人工智能\\数据集\\贝叶斯中文邮件分类\\'+
35 str(i) +'.txt' for i in range(151)]
36     # 获取训练集中所有邮件中的全部单词
37     for txtFile in txtFiles:
38         allwords.append(getWordsFromFile(txtFile))
39     # 获取并返回出现次数最多的前topN个单词
40     freq = Counter(chain(*allwords))
41     return [w[0] for w in freq.most_common(topN)]
42
43 # 全部训练集中出现次数最多的前600个单词
44 topwords = getTopNWords(600)
45
46 # 获取特征向量，前600个单词的每个单词在每个邮件中出现的频率
47 vectors = []

```

```

47 for words in allWords:
48     temp = list(map(lambda x: words.count(x), topwords))
49     vectors.append(temp)
50
51 vectors = array(vectors)
52 # 训练集中每个邮件的标签，1表示垃圾邮件，0表示正常邮件
53 labels = array([1]*127 + [0]*24)
54
55 # 创建模型，使用已知训练集进行训练
56 model = MultinomialNB()
57 model.fit(vectors, labels)
58
59 def predict(txtFile):
60     # 获取指定邮件文件内容，返回分类结果
61     words = getWordsFromFile(txtFile)
62     currentVector = array(tuple(map(lambda x: words.count(x),
63                                     topwords)))
64     result = model.predict(currentVector.reshape(1, -1))[0]
65     print(model.predict_proba(currentVector.reshape(1, -1)))
66     return '垃圾邮件' if result==1 else '正常邮件'
67
68 # 151.txt至155.txt为测试邮件内容
69 for mail in ('D:\\workspaces\\AI\\ip\\人工智能\\数据集\\贝叶斯中文邮件分类\\'+
70             str(i) +'.txt' for i in range(151, 156)):
71     print(mail, predict(mail), sep=':')

```

结果截图

```

def predict(txtFile):
    # 获取指定邮件文件内容，返回分类结果
    words = getWordsFromFile(txtFile)
    currentVector = array(tuple(map(lambda x: words.count(x),
                                    topWords)))
    result = model.predict(currentVector.reshape(1, -1))[0]
    print(model.predict_proba(currentVector.reshape(1, -1)))
    return '垃圾邮件' if result==1 else '正常邮件'

# 151.txt至155.txt为测试邮件内容
for mail in ('D:\\workspaces\\AI\\ip\\人工智能\\数据集\\贝叶斯中文邮件分类\\'+ str(i) +'.txt' for i in range(151, 156)):
    print(mail, predict(mail), sep=':')

[[0.00531716 0.99468284]]
D:\workspaces\AI\ip\人工智能\数据集\贝叶斯中文邮件分类\151.txt:垃圾邮件
[[9.86125127e-13 1.00000000e+00]]
D:\workspaces\AI\ip\人工智能\数据集\贝叶斯中文邮件分类\152.txt:垃圾邮件
[[0.1589404 0.8410596]]
D:\workspaces\AI\ip\人工智能\数据集\贝叶斯中文邮件分类\153.txt:垃圾邮件
[[3.13377251e-04 9.99686623e-01]]
D:\workspaces\AI\ip\人工智能\数据集\贝叶斯中文邮件分类\154.txt:垃圾邮件
[[0.88673294 0.11326706]]
D:\workspaces\AI\ip\人工智能\数据集\贝叶斯中文邮件分类\155.txt:正常邮件

```

报错出现：

1. 编辑路径的时候出现报错：EOL while scanning string literal，发现其实是自己的路径使用的是单行线，网上查之后改为单行线就运行成功了。
2. 导包错误：No module named 'jieba'，直接打开jupyter notebook环境运行 `pip install jieba` 即可

决策树算法验证实验

```

1 import numpy as np
2 from sklearn import tree
3 # 数据
4 x = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],[1, 0, 0], [1, 0,
5 1], [1, 1, 0], [1, 1, 1]])
6 y=[0,1,1,1,2,3,3,4]
7 # 创建决策树分类器
8 clf=tree.DecisionTreeClassifier()
9 clf.fit(X,y)

```

Out[10]:DecisionTreeClassifier()

```

1 # 预测
2 clf.predict([[1,0,0]])

```

Out[11]:array([2])

```

1 import graphviz
2 # 导出决策树
3 dot_data=tree.export_graphviz(clf,out_file=None)
4 # 创建图形
5 graph= graphviz.Source(dot_data)
6 # 输出pdf文件
7 graph.render('result')

```

Out[4]:'result.pdf'

结果截图

```

In [10]: import numpy as np
         from sklearn import tree
         # 数据
         X = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])
         y=[0,1,1,1,2,3,3,4]
         # 创建决策树分类器
         clf=tree.DecisionTreeClassifier()
         clf.fit(X,y)

```

Out[10]: DecisionTreeClassifier()

```

In [11]: # 预测
         clf.predict([[1,0,0]])

```

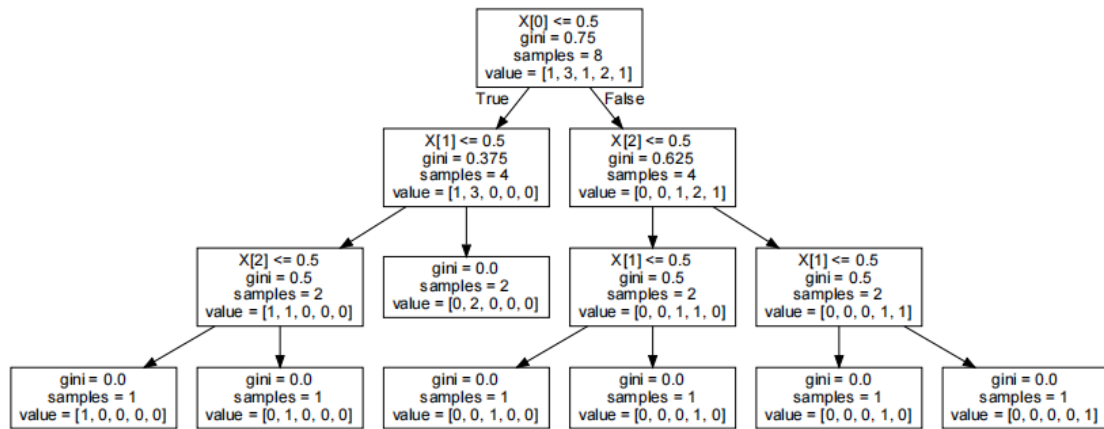
Out[11]: array([2])

```

In [12]: import graphviz
         # 导出决策树
         dot_data=tree.export_graphviz(clf,out_file=None)
         # 创建图形
         graph= graphviz.Source(dot_data)
         # 输出pdf文件
         graph.render('result')

```

Out[12]: 'result.pdf'



报错信息

failed to execute ['dot', '-Tsvg'], make sure the Graphviz executables are on your systems 原因是 graphviz 模块没安装，但是我们直接 `pip install graphviz` 也是会同样报错的，因为这样下载系统环境变量是没有这个模块的。解决方案是直接到 graphviz 官网中去下载安装，并且配置好系统环境变量，最后加载到 python 环境就可以了。

判断学员的Python水平

程序思路

通过一个问卷调查来输入数据，检验所学的课本以及学习的程度来检验学员的python水平。

1. 训练模型

```

1  questions = ('《Python程序设计基础（第2版）》',
2              '《Python程序设计基础与应用》',
3              '《Python程序设计（第2版）》',
4              '《大数据的Python基础》',
5              '《Python程序设计开发宝典》',
6              '《Python可以这样学》',
7              '《中学生可以这样学Python》',
8              '《Python编程基础与案例集锦（中学版）》',
9              '《玩转Python轻松过二级》',
10             '微信公众号“Python小屋”的免费资料',)
11  # 每个样本的数据含义：
12  # 0没看过，1很多看不懂，2大部分可以看懂，3没压力
13  answers = [[3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
14             [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
15             [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
16             [0, 0, 0, 0, 0, 0, 2, 2, 0, 1],
17             [0, 0, 0, 0, 3, 3, 0, 0, 0, 3],
18
19             [3, 3, 0, 3, 0, 0, 0, 0, 3, 1],
20             [3, 0, 3, 0, 3, 0, 0, 3, 3, 2],
21             [0, 0, 3, 0, 3, 3, 0, 0, 0, 3],
22             [2, 2, 0, 2, 0, 0, 0, 0, 0, 1],
23             [0, 2, 1, 3, 1, 1, 0, 0, 2, 1]
24         ]
25
26  labels = ['超级高手', '门外汉', '初级选手', '初级选手', '高级选手',
27           '中级选手', '高级选手', '超级高手', '初级选手', '初级选手']
28

```

```
29 clf = tree.DecisionTreeClassifier().fit(answers, labels) # 训练
```

2. 设计调查问卷并且分类

```
1 yourAnswer = []
2 # 显示调查问卷，并接收用户输入
3 for question in questions:
4     print('=====\n你看过董付国老师的', question, '吗? ')
5     # 确保输入有效
6     while True:
7         print('没看过输入0，很多看不懂输入1，'
8               '大部分可以看懂输入2，没压力输入3')
9         try:
10            answer = int(input('请输入: '))
11            assert 0<=answer<=3
12            break
13        except:
14            print('输入无效，请重新输入。')
15            pass
16    yourAnswer.append(answer)
17
18 print(clf.predict(np.array(yourAnswer).reshape(1,-1))) # 分类
```

3. 完整代码

```
1 from sklearn import tree
2 import numpy as np
3 # s数据
4 questions = ('《Python程序设计基础（第2版）》',
5              '《Python程序设计基础与应用》',
6              '《Python程序设计（第2版）》',
7              '《大数据的Python基础》',
8              '《Python程序设计开发宝典》',
9              '《Python可以这样学》',
10             '《中学生可以这样学Python》',
11             '《Python编程基础与案例集锦（中学版）》',
12             '《玩转Python轻松过二级》',
13             '微信公众号“Python小屋”的免费资料',)
14 # 每个样本的数据含义：
15 # 0没看过，1很多看不懂，2大部分可以看懂，3没压力
16 answers = [[3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
17            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
18            [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
19            [0, 0, 0, 0, 0, 0, 2, 2, 0, 1],
20            [0, 0, 0, 0, 3, 3, 0, 0, 0, 3],
21
22            [3, 3, 0, 3, 0, 0, 0, 0, 3, 1],
23            [3, 0, 3, 0, 3, 0, 0, 3, 3, 2],
24            [0, 0, 3, 0, 3, 3, 0, 0, 0, 3],
25            [2, 2, 0, 2, 0, 0, 0, 0, 0, 1],
26            [0, 2, 1, 3, 1, 1, 0, 0, 2, 1]
27        ]
28
29 labels = ['超级高手', '门外汉', '初级选手', '初级选手', '高级选手',
30           '中级选手', '高级选手', '超级高手', '初级选手', '初级选手']
31
```

```

32 clf = tree.DecisionTreeClassifier().fit(answers, labels) # 训练
33
34 yourAnswer = []
35 # 显示调查问卷，并接收用户输入
36 for question in questions:
37     print('=====\n你看过董付国老师的', question, '吗? ')
38     # 确保输入有效
39     while True:
40         print('没看过输入0，很多看不懂输入1，'
41               '大部分可以看懂输入2，没压力输入3')
42         try:
43             answer = int(input('请输入: '))
44             assert 0<=answer<=3
45             break
46         except:
47             print('输入无效，请重新输入。')
48             pass
49     yourAnswer.append(answer)
50
51 print(clf.predict(np.array(yourAnswer).reshape(1,-1))) # 分类

```

结果截图

你看过董付国老师的《Python程序设计基础（第2版）》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：3

你看过董付国老师的《Python程序设计基础与应用》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：3

你看过董付国老师的《Python程序设计（第2版）》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：0

你看过董付国老师的《大数据的Python基础》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：6
输入无效，请重新输入。
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：0

无效数据

你看过董付国老师的《Python程序设计开发宝典》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：0

你看过董付国老师的《Python可以这样学》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：3

你看过董付国老师的《中学生可以这样学Python》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：0

你看过董付国老师的《Python编程基础与案例集锦（中学版）》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：3

你看过董付国老师的《玩转Python轻松过二级》吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：2

你看过董付国老师的 微信公众号“Python小屋”的免费资料 吗？
没看过输入0，很多看不懂输入1，大部分可以看懂输入2，没压力输入3
请输入：0
['门外汉']

支持向量机算法对手写数字图像进行分类

程序思路

1. 随程序生成1000张图片，并且保存在 D:\workspaces\AI\ip\人工智能\数据集
\\datasets 上，这些图片的数字生成是随机的 `digit = choice(digits)`，最后把写到的数
据都放在 `digits.txt` 文件中。

```

1 # 生成图片
2 def generateDigits(dstDir='D:\\workspaces\\AI\\ip\\人工智能\\数据集\\datasets', num=1000):
3     # 生成num个包含数字的图片文件存放于datasets子目录
4     if not isdir(dstDir):
5         mkdir(dstDir)
6
7     # digits.txt用来存储每个图片对应的数字
8     with open(dstDir+'\\digits.txt', 'w') as fp:
9         for i in range(num):
10            # 随机选择一个数字, 生成对应的彩色图像文件
11            digit = choice(digits)
12            im = Image.new('RGB', (width,height), (255,255,255))
13            imDraw = ImageDraw.Draw(im)
14            font = truetype('c:\\windows\\fonts\\TIMESBD.TTF', fontSize)
15            # 写入黑色数字
16            imDraw.text((0,0), digit, font=font, fill=(0,0,0))
17            # 加入随机干扰
18            for j in range(int(noiseRate*width*height)):
19                w, h = randrange(1, width-1), randrange(height)
20                # 水平交换两个相邻像素的颜色
21                c1 = im.getpixel((w,h))
22                c2 = im.getpixel((w+1,h))
23                imDraw.point((w,h), fill=c2)
24                imDraw.point((w+1,h), fill=c1)
25            im.save(dstDir+'\\'+str(i)+'.jpg')
26            fp.write(digit+'\n')

```



2. 加载图片

```

1 def loadDigits(dstDir='D:\\workspaces\\AI\\ip\\人工智能\\数据集\\datasets'):
2     # 获取所有图像文件名
3     digitsFile = [dstDir+'\\'+fn for fn in listdir(dstDir) if
4                    fn.endswith('.jpg')]
5     # 按编号排序
6     digitsFile.sort(key=lambda fn: int(basename(fn)[-4]))
7     # digitsData用于存放读取的图片中数字信息
8     # 每个图片中所有像素值存放于digitsData中的一行数据

```

```

8     digitsData = []
9     for fn in digitsFile:
10         with Image.open(fn) as im:
11             # getpixel()方法用来读取指定位置像素的颜色值
12             data = [sum(im.getpixel((w,h)))/len(im.getpixel((w,h)))
13                     for w in range(width)
14                     for h in range(height)]
15             digitsData.append(data)
16
17         # digitsLabel用于存放图片中数字的标准分类
18         with open(dstDir+'\\digits.txt') as fp:
19             digitsLabel = fp.readlines()
20         # 删除数字字符两侧的空白字符
21         digitsLabel = [label.strip() for label in digitsLabel]
22         return (digitsData, digitsLabel)

```

3. 全部代码展示

```

1  from os import mkdir, listdir
2  from os.path import isdir, basename
3  from random import choice, randrange
4  from string import digits
5  from PIL import Image, ImageDraw
6  from PIL.ImageFont import truetype
7  from sklearn import svm
8  from sklearn.model_selection import train_test_split
9
10 # 图像尺寸、图片中的数字字体大小、噪点比例
11 width, height = 30, 60
12 fontSize = 40
13 noiseRate = 8
14
15 # 生成图片
16 def generateDigits(dstDir='D:\\workspaces\\AI\\ip\\人工智能\\数据集\\datasets', num=1000):
17     # 生成num个包含数字的图片文件存放于datasets子目录
18     if not isdir(dstDir):
19         mkdir(dstDir)
20
21     # digits.txt用来存储每个图片对应的数字
22     with open(dstDir+'\\digits.txt', 'w') as fp:
23         for i in range(num):
24             # 随机选择一个数字，生成对应的彩色图像文件
25             digit = choice(digits)
26             im = Image.new('RGB', (width,height), (255,255,255))
27             imDraw = ImageDraw.Draw(im)
28             font = truetype('c:\\windows\\fonts\\TIMESBD.TTF', fontSize)
29             # 写入黑色数字
30             imDraw.text((0,0), digit, font=font, fill=(0,0,0))
31             # 加入随机干扰
32             for j in range(int(noiseRate*width*height)):
33                 w, h = randrange(1, width-1), randrange(height)
34                 # 水平交换两个相邻像素的颜色
35                 c1 = im.getpixel((w,h))
36                 c2 = im.getpixel((w+1,h))
37                 imDraw.point((w,h), fill=c2)
38                 imDraw.point((w+1,h), fill=c1)

```

```

39         im.save(dstDir+'\\'+str(i)+'.jpg')
40         fp.write(digit+'\n')
41
42     # 加载图片
43     def loadDigits(dstDir='D:\\workspaces\\AI\\ip\\人工智能\\数据集\\datasets'):
44         # 获取所有图像文件名
45         digitsFile = [dstDir+'\\'+fn for fn in listdir(dstDir) if
fn.endswith('.jpg')]
46         # 按编号排序
47         digitsFile.sort(key=lambda fn: int(basename(fn)[-4]))
48         # digitsData用于存放读取的图片中数字信息
49         # 每个图片中所有像素值存放于digitsData中的一行数据
50         digitsData = []
51         for fn in digitsFile:
52             with Image.open(fn) as im:
53                 # getpixel()方法用来读取指定位置像素的颜色值
54                 data = [sum(im.getpixel((w,h)))/len(im.getpixel((w,h)))
55                         for w in range(width)
56                         for h in range(height)]
57                 digitsData.append(data)
58
59         # digitsLabel用于存放图片中数字的标准分类
60         with open(dstDir+'\\digits.txt') as fp:
61             digitsLabel = fp.readlines()
62         # 删除数字字符两侧的空白字符
63         digitsLabel = [label.strip() for label in digitsLabel]
64         return (digitsData, digitsLabel)
65
66     # 生成图片文件
67     generateDigits(num=1000)
68     # 加载数据
69     data = loadDigits()
70     print('数据加载完成。')
71
72     # 随机划分训练集和测试集，其中参数test_size用来指定测试集大小
73     X_train, X_test, y_train, y_test = train_test_split(data[0], data[1],
test_size=0.1)
74     # 创建并训练模型
75     svcClassifier = svm.SVC(kernel="linear", C=1000, gamma=0.001)
76     svcClassifier.fit(X_train, y_train)
77     print('模型训练完成。')
78
79     # 使用测试集对模型进行评分
80     score = svcClassifier.score(X_test, y_test)
81     print('模型测试得分: ', score)

```

结果截图

```

svcClassifier = svm.SVC(kernel="linear", C=1000, gamma=0.001)
svcClassifier.fit(X_train, y_train)
print(' 模型训练完成。')

# 使用测试集对模型进行评分
score = svcClassifier.score(X_test, y_test)
print(' 模型测试得分: ', score)

```

数据加载完成。

模型训练完成。

模型测试得分: 1.0

KNN算法判断交通工具

- 判断交通工具的标准在于时速，这里使用 `unknown=np.random.randint(0,1000,size=(20,4))` 随机生成20个测试数据，数据范围是0-1000，以此来测试这个模型。

```

1  from sklearn.neighbors import KNeighborsClassifier
2  import numpy as np
3
4  # x中存储交通工具的参数
5  # 总长度（米）、时速（km/h）、重量（吨）、座位数量
6  x = [[96, 85, 120, 400],          # 普通火车
7        [144, 92, 200, 600],
8        [240, 87, 350, 1000],
9        [360, 90, 495, 1300],
10       [384, 91, 530, 1405],
11       [240, 360, 490, 800],      # 高铁
12       [360, 380, 750, 1200],
13       [290, 380, 480, 960],
14       [120, 320, 160, 400],
15       [384, 340, 520, 1280],
16       [33.4, 918, 77, 180],      # 飞机
17       [33.6, 1120, 170.5, 185],
18       [39.5, 785, 230, 240],
19       [33.84, 940, 150, 195],
20       [44.5, 920, 275, 275],
21       [75.3, 1050, 575, 490]]
22 # y中存储类别，0表示普通火车，1表示高铁，2表示飞机
23 y = [0]*5+[1]*5+[2]*6
24 # labels中存储对应的交通工具名称
25 labels = ('普通火车', '高铁', '飞机')
26
27 # 创建并训练模型
28 knn = KNeighborsClassifier(n_neighbors=3, weights='distance')
29 knn.fit(x, y)
30
31 # 对未知样本进行分类，随机生成数据
32 # unknown = [[300, 79, 320, 900], [36.7, 800, 190, 220]]
33 unknown=np.random.randint(0,1000,size=(20,4))
34 result = knn.predict(unknown)
35 for para, index in zip(unknown, result):
36     print(para, labels[index], sep=':')
37

```

结果截图

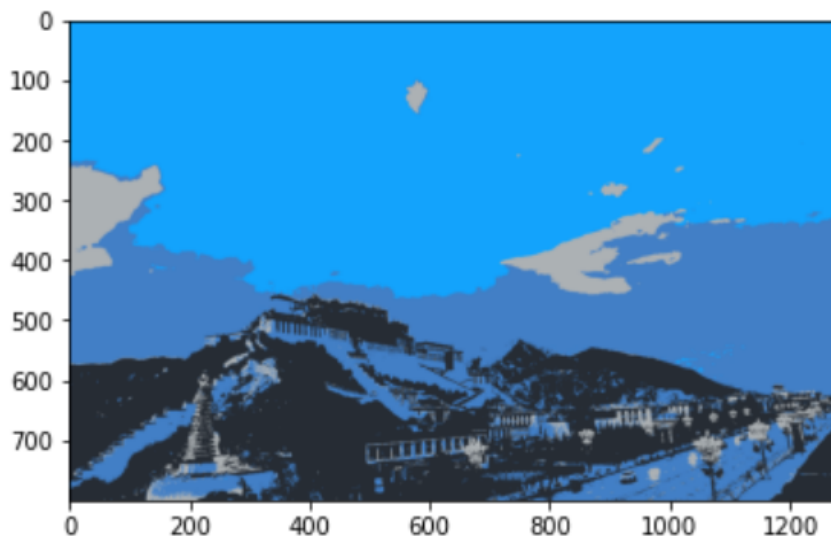
[931 487 764 741]:高铁
[46 194 77 975]:普通火车
[402 542 355 784]:高铁
[182 781 446 366]:飞机
[685 867 805 203]:飞机
[45 156 989 639]:高铁
[620 304 919 515]:高铁
[125 925 922 749]:高铁
[536 918 546 642]:飞机
[27 755 948 288]:飞机
[232 32 265 662]:普通火车
[975 243 921 72]:高铁
[869 941 104 96]:飞机
[827 449 849 867]:高铁
[334 49 147 106]:普通火车
[745 231 874 305]:高铁
[640 850 192 704]:高铁
[412 371 183 582]:高铁
[279 746 715 241]:飞机
[944 312 894 662]:高铁




KMeans聚类算法压缩图像颜色

- KMeans是一种无监督学习算法，在本实验中，选取四个样本空间为初始中心，不断比较附近的像素，更新聚类中心的像素值，直到最后无法更新，程序停止。
- 这里使用的是绝对路径 D:\\workspaces\\AI\\ip\\人工智能\\数据集\\...
- 代码展示：

```
1 import numpy as np
2 from sklearn.cluster import KMeans
3 from PIL import Image
4 import matplotlib.pyplot as plt
5
6 # 打开并读取原始图像中像素颜色值，转换为三维数组
7 imOrigin = Image.open('D:\\workspaces\\AI\\ip\\人工智能\\数据集\\颜色压缩测试图像.jpg')
8 dataOrigin = np.array(imOrigin)
9 # 然后再转换为二维数组，-1表示自动计算该维度的大小
10 data = dataOrigin.reshape(-1,3)
11
12 # 使用KMeans算法把所有像素的颜色值划分为4类
13 kmeansPredictor = KMeans(n_clusters=4)
14 kmeansPredictor.fit(data)
15
16 # 使用每个像素所属类的中心值替换该像素的颜色
17 # temp中存放每个数据所属类的标签
18 temp = kmeansPredictor.labels_
19 dataNew = kmeansPredictor.cluster_centers_[temp]
20 dataNew.shape = dataOrigin.shape
21 dataNew = np.uint8(dataNew)
22 plt.imshow(dataNew)
23 plt.imsave('D:\\workspaces\\AI\\ip\\人工智能\\数据集\\结果图像.jpg', dataNew)
24 plt.show()
```


结果展示



 结果图像.jpg	2022/3/18 21:52
 商场一楼手机信号强度.txt	2019/4/10 8:24
 颜色压缩测试图像.jpg	2019/6/1 14:01

分层聚类算法验证实验

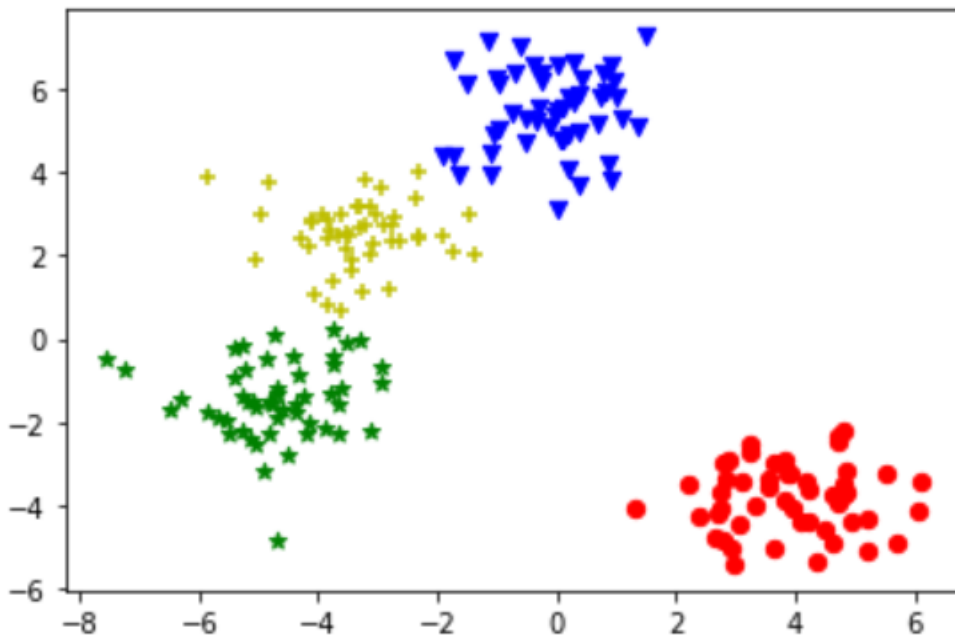
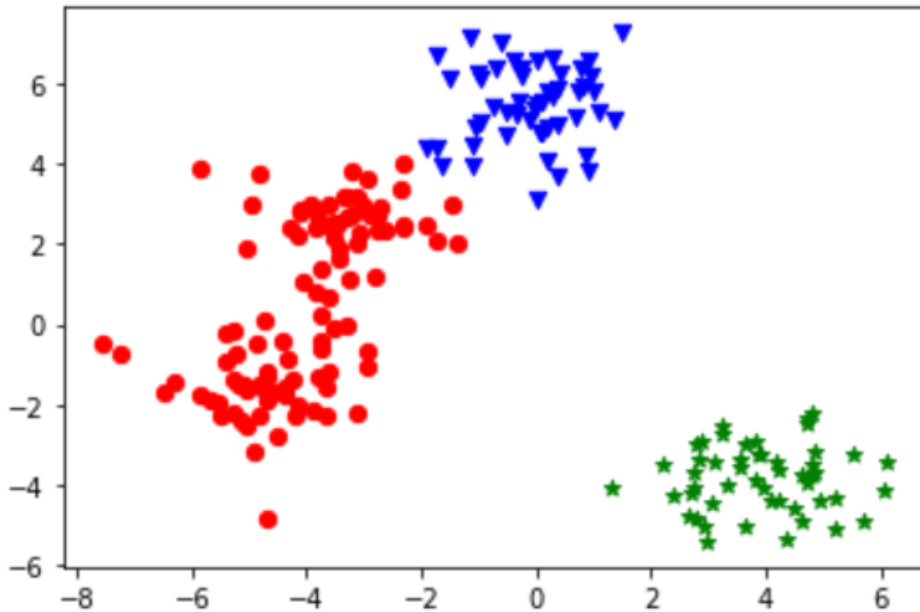
- 分层聚类又称为系统聚类算法或者系谱聚类算法，这个方法把所有样本都看做各自一类，定义类间距计算方式，选择距离最小的一对元素合并成一个新的类，重新计算各个类之间的距离并重复上面的步骤，直到所有原始元素划分为指定数量的类。
- 这个类计算的复杂度非常高，不适合大数据聚类问题
- 代码展示：

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import make_blobs
4 from sklearn.cluster import AgglomerativeClustering
5
6 def AgglomerativeTest(n_clusters):
7     assert 1 <= n_clusters <= 4
8     predictResult = AgglomerativeClustering(n_clusters=n_clusters,
9     affinity='euclidean',
10
11     linkage='ward').fit_predict(data)
12     # 定义绘制散点图时使用的颜色和散点符号
13     colors = 'rgby'
14     markers = 'o*v+'
15     # 依次使用不同的颜色和符号绘制每个类的散点图
16     for i in range(n_clusters):
17         subData = data[predictResult==i]
18         plt.scatter(subData[:,0], subData[:,1], c=colors[i],
19         marker=markers[i], s=40)
20     plt.show()
```

```
20 # 生成随机数据，200个点，分成4类，返回样本及标签
21 data, labels = make_blobs(n_samples=200, centers=4)
22 print(data)
23 AgglomerativeTest(3)
24 AgglomerativeTest(4)
```

结果截图

```
[-5.21331796e+00 -7.18973858e-01]
[-4.87682644e+00 -4.74622923e-01]
[-2.53651547e-01  6.20463143e+00]
[-2.61965427e+00  2.37917105e+00]
```



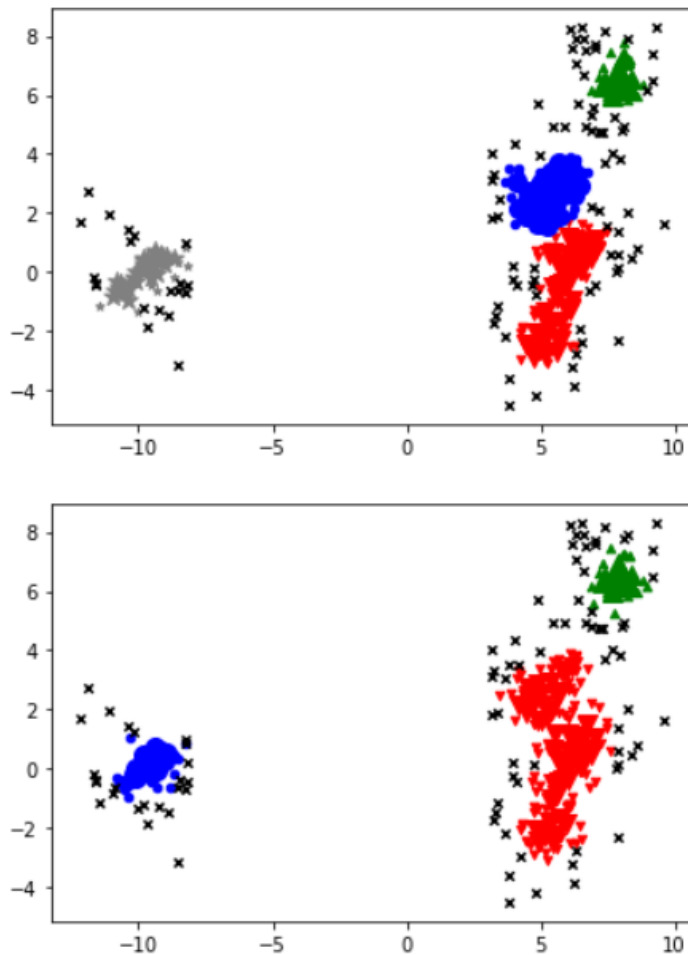
DBSCAN算法验证实验

- 属于密度聚类算法，把类定义为密度相连对象的最大集合，把类定义为密度相连对象的最大集合，通过在样本空间中不断搜索高密度的核心样本并扩展得到最大集合完成聚类，能够在带有噪点的样本空间中发现任意形状的聚类并排除噪点。

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.cluster import DBSCAN
4 from sklearn.datasets import make_blobs
5
6 def DBSCANtest(data, eps=0.6, min_samples=8):
7     # 聚类
8     db = DBSCAN(eps=eps, min_samples=min_samples).fit(data)
9
10    # 聚类标签（数组，表示每个样本所属聚类）和所有聚类的数量
11    # 标签-1对应的样本表示噪点
12    clusterLabels = db.labels_
13    uniqueClusterLabels = set(clusterLabels)
14    # 标记核心对象对应下标为True
15    coreSamplesMask = np.zeros_like(db.labels_, dtype=bool)
16    coreSamplesMask[db.core_sample_indices_] = True
17
18    # 绘制聚类结果
19    colors = ['red', 'green', 'blue', 'gray', '#88ff66',
20              '#ff00ff', '#ffff00', '#8888ff', 'black',]
21    markers = ['v', '^', 'o', '*', 'h', 'd', 'D', '>', 'x']
22    for label in uniqueClusterLabels:
23        # 使用最后一种颜色和符号绘制噪声样本
24        # clusterIndex是个True/False数组
25        # 其中True表示对应样本为cluster类
26        clusterIndex = (clusterLabels==label)
27
28        # 绘制核心对象
29        coreSamples = data[clusterIndex&coreSamplesMask]
30        plt.scatter(coreSamples[:, 0], coreSamples[:, 1],
31                    c=colors[label], marker=markers[label], s=100)
32
33        # 绘制非核心对象
34        nonCoreSamples = data[clusterIndex & ~coreSamplesMask]
35        plt.scatter(nonCoreSamples[:, 0], nonCoreSamples[:, 1],
36                    c=colors[label], marker=markers[label], s=20)
37    plt.show()
38
39    data, labels = make_blobs(n_samples=300, centers=5)
40    DBSCANtest(data)
41    DBSCANtest(data, 0.8, 15)
42

```



协同过滤算法

- 这个算法适合于商品推荐、商品捆绑。根据用户之间或商品之间的相似性进行精准推荐，可以分为基于用户的协同过滤算法和基于商品的协同过滤算法。

```

1  from random import randrange
2
3  # 模拟历史电影打分数据，共10个用户，每个用户打分的电影数量不等
4  data = {'user'+str(i):{'film'+str(randrange(1, 15)):randrange(1, 6) for j in
5  range(randrange(3, 10))}
6  for i in range(10)}
7  # 寻求推荐的用户对电影打分的数据
8  user = {'film'+str(randrange(1, 15)):randrange(1,6) for i in range(5)}
9
10 # 最相似的用户及其对电影打分情况
11 # 两个最相似的用户共同打分的电影最多，同时所有电影打分差值的平方和最小
12 rule = lambda item:(-len(item[1].keys()&user),
13 sum(((item[1].get(film)-user.get(film))**2 for film in
14 user.keys()&item[1].keys()))
15 similarUser, films = min(data.items(), key=rule)
16 # 输出信息以便验证，每行数据有3列
17 # 分别为该用户与当前用户共同打分的电影数量、打分差的平方和、该用户打分数据
18 print('known data'.center(50, '='))
19 for item in data.items():
20     print(len(item[1].keys()&user.keys()),
21           sum(((item[1].get(film)-user.get(film))**2 for film in
22 user.keys()&item[1].keys()))),
23           item, sep=':')
24 print('current user'.center(50, '='), user, sep='\n')

```

```

22 print('most similar user and his films'.center(50, '='))
23 print(similarUser, films, sep=':')
24 print('recommended film'.center(50, '='))
25 # 在当前用户没看过的电影中选择打分最高的进行推荐
26 print(max(films.keys()-user.keys(), key=lambda film: films[film]))
27

```

结果截图

```

=====known data=====
2:4:('user0', {'film10': 4, 'film6': 1, 'film2': 2, 'film5': 2, 'film3': 5, 'film9': 4})
4:12:('user1', {'film6': 2, 'film2': 1, 'film9': 3, 'film4': 3, 'film7': 4, 'film3': 4})
0:0:('user2', {'film10': 1, 'film14': 5, 'film2': 4, 'film1': 1, 'film5': 3, 'film12': 2})
1:4:('user3', {'film8': 4, 'film4': 4, 'film1': 2, 'film13': 1, 'film6': 3, 'film2': 5, 'film14': 5, 'film10': 4})
1:0:('user4', {'film10': 1, 'film14': 1, 'film12': 3, 'film3': 5, 'film13': 5})
2:10:('user5', {'film11': 1, 'film6': 2, 'film1': 2, 'film9': 5, 'film7': 2, 'film12': 2, 'film13': 2, 'film8': 5})
3:18:('user6', {'film2': 1, 'film4': 2, 'film5': 2, 'film11': 2, 'film10': 2, 'film12': 5, 'film7': 4, 'film9': 5})
1:4:('user7', {'film2': 3, 'film5': 4, 'film9': 4, 'film8': 1, 'film12': 1})
0:0:('user8', {'film12': 3, 'film14': 4, 'film8': 4, 'film2': 2, 'film1': 5, 'film10': 5})
1:9:('user9', {'film6': 4, 'film5': 2, 'film10': 4, 'film13': 2, 'film7': 4, 'film2': 2})
=====current user=====
{'film3': 5, 'film7': 1, 'film9': 2, 'film4': 2}
=====most similar user and his films=====
user1: {'film6': 2, 'film2': 1, 'film9': 3, 'film4': 3, 'film7': 4, 'film3': 4}
=====recommended film=====
film6

```

使用关联规则分析演员关系

```

1  from itertools import chain, combinations
2  from openpyxl import load_workbook
3
4  def loadDataSet():
5      '''加载数据，返回包含若干集合的列表'''
6      # 返回的数据格式为 [{1, 3, 4}, {2, 3, 5}, {1, 2, 3, 5}, {2, 5}]
7      result = []
8      # xlsx文件中有3列，分别为电影名称、导演名称、演员清单
9      # 同一个电影的多个主演演员使用逗号分隔
10     ws = load_workbook('D:\\workspaces\\AI\\ip\\人工智能\\数据集\\电影导演演员.xls').worksheets[0]
11     for index, row in enumerate(ws.rows):
12         # 跳过第一行表头
13         if index==0:
14             continue
15         result.append(set(row[2].value.split(', ')))
16     return result
17
18 def createC1(dataSet):
19     '''dataSet为包含集合的列表，每个集合表示一个项集
20     返回包含若干元组的列表，
21     每个元组为只包含一个物品的项集，所有项集不重复'''
22     return sorted(map(lambda i:(i,), set(chain(*dataSet))))
23
24 def scanD(dataSet, Ck, Lk, minSupport):
25     '''dataSet为包含集合的列表，每个集合表示一个项集
26     ck为候选项集列表，每个元素为元组
27     minSupport为最小支持度阈值
28     返回Ck中支持度大于等于minSupport的那些项集'''
29     # 数据集总数量
30     total = len(dataSet)
31     supportData = {}
32     for candidate in Ck:
33         # 加速，k-频繁项集的所有k-1子集都应该是频繁项集

```

```

34         if Lk and (not all(map(lambda item: item in Lk,
35                                 combinations(candidate,
36                                             len(candidate)-1))))):
37             continue
38         # 遍历每个候选项集，统计该项集在所有数据集中出现的次数
39         # 这里隐含了一个技巧：True在内部存储为1
40         set_candidate = set(candidate)
41         frequencies = sum(map(lambda item: set_candidate<=item,
42                               dataSet))
43         # 计算支持度
44         t = frequencies/total
45         # 大于等于最小支持度，保留该项集及其支持度
46         if t >= minSupport:
47             supportData[candidate] = t
48     return supportData
49
50 def aprioriGen(Lk, k):
51     '''根据k项集生成k+1项集'''
52     result = []
53     for index, item1 in enumerate(Lk):
54         for item2 in Lk[index+1:]:
55             # 只合并前k-2项相同的项集，避免生成重复项集
56             # 例如，(1,3)和(2,5)不会合并，
57             # (2,3)和(2,5)会合并为(2,3,5)，
58             # (2,3)和(3,5)不会合并，
59             # (2,3)、(2,5)、(3,5)只能得到一个项集(2,3,5)
60             if sorted(item1[:k-2]) == sorted(item2[:k-2]):
61                 result.append(tuple(set(item1)|set(item2)))
62     return result
63
64 def apriori(dataSet, minSupport=0.5):
65     '''根据给定数据集dataSet，
66         返回所有支持度>=minSupport的频繁项集'''
67     C1 = createC1(dataSet)
68     supportData = scanD(dataSet, C1, None, minSupport)
69     k = 2
70     while True:
71         # 获取满足最小支持度的k项集
72         Lk = [key for key in supportData if len(key)==k-1]
73         # 合并生成k+1项集
74         Ck = aprioriGen(Lk, k)
75         # 筛选满足最小支持度的k+1项集
76         supK = scanD(dataSet, Ck, Lk, minSupport)
77         # 无法再生成包含更多项的项集，算法结束
78         if not supK:
79             break
80         supportData.update(supK)
81         k = k+1
82     return supportData
83
84 def findRules(supportData, minConfidence=0.5):
85     '''查找满足最小置信度的关联规则'''
86     # 对频繁项集按长度降序排列
87     supportDataL = sorted(supportData.items(),
88                           key=lambda item: len(item[0]),
89                           reverse=True)
90     rules = []
91     for index, pre in enumerate(supportDataL):

```

```

92         for aft in supportDataL[index+1:]:
93             # 只查找k-1项集到k项集的关联规则
94             if len(aft[0]) < len(pre[0])-1:
95                 break
96             # 当前项集aft[0]是pre[0]的子集
97             # 且aft[0]==>pre[0]的置信度大于等于最小置信度阈值
98             if set(aft[0])<set(pre[0]) and\
99                 pre[1]/aft[1]>=minConfidence:
100                 rules.append([pre[0],aft[0]])
101         return rules
102
103     # 加载数据
104     dataSet = loadDataSet()
105     # 获取所有支持度大于0.2的项集
106     supportData = apriori(dataSet, 0.2)
107     # 在所有频繁项集中查找并输出关系较好的演员二人组合
108     bestPair = [item for item in supportData if len(item)==2]
109     print(bestPair)
110
111     # 查找支持度大于0.6的强关联规则
112     for item in findRules(supportData, 0.6):
113         pre, aft = map(set, item)
114         print(aft, pre-aft, sep='==>')
115

```

结果截图：

```

(['演员3', '演员1'), ('演员4', '演员1'), ('演员3', '演员4'), ('演员3', '演员5'), ('演员4', '演员9')]
{'演员4', '演员1'}==>{'演员3'}
{'演员1'}==>{'演员3'}
{'演员1'}==>{'演员4'}
{'演员3'}==>{'演员4'}
{'演员4'}==>{'演员3'}
{'演员5'}==>{'演员3'}
{'演员9'}==>{'演员4'}

```

使用交叉验证与网格搜索，进行支持向量机分类模型的学习

- 交叉验证会反复对数据集进行划分，并使用不同的划分对模型进行评分，可以更好地评估模型的泛化质量。

```

1  from time import time
2  from os import listdir
3  from os.path import basename
4  from PIL import Image
5  from sklearn import svm
6  from sklearn.model_selection import cross_val_score,\
7      ShuffleSplit, LeaveOneOut
8
9  # 图像尺寸
10 width, height = 30, 60
11
12 def loadDigits(dstDir='D:\\workspaces\\AI\\ip\\人工智能\\数据集\\datasets'):
13     # 获取所有图像文件名
14     digitsFile = [dstDir+'\\'+fn for fn in listdir(dstDir)
15                   if fn.endswith('.jpg')]
16     # 按编号排序

```

```

17     digitsFile.sort(key=lambda fn: int(basename(fn)[-4]))
18     # digitsData用于存放读取的图片中数字信息
19     # 每个图片中所有像素值存放于digitsData中的一行数据
20     digitsData = []
21     for fn in digitsFile:
22         with Image.open(fn) as im:
23             data = [sum(im.getpixel((w,h)))/len(im.getpixel((w,h)))
24                     for w in range(width)
25                     for h in range(height)]
26             digitsData.append(data)
27     # digitsLabel用于存放图片中数字的标准分类
28     with open(dstDir+'\\digits.txt') as fp:
29         digitsLabel = fp.readlines()
30     digitsLabel = [label.strip() for label in digitsLabel]
31     return (digitsData, digitsLabel)
32
33 # 加载数据
34 data = loadDigits()
35 print('数据加载完成。')
36
37 # 创建模型
38 svcClassifier = svm.SVC(kernel="linear", C=1000, gamma=0.001)
39
40 # 交叉验证
41 start = time()
42 scores = cross_val_score(svcClassifier, data[0], data[1], cv=8)
43 print('交叉验证（k折叠）得分情况: \n', scores)
44 print('平均分: \n', scores.mean())
45 print('用时（秒）: ', time()-start)
46 print('='*20)
47
48 start = time()
49 scores = cross_val_score(svcClassifier, data[0], data[1],
50                           cv=ShuffleSplit(test_size=0.3,
51                                             train_size=0.7,
52                                             n_splits=10))
53 print('交叉验证（随机拆分）得分情况: \n', scores)
54 print('平均分: \n', scores.mean())
55 print('用时（秒）: ', time()-start)
56 print('='*20)
57
58 start = time()
59 scores = cross_val_score(svcClassifier, data[0], data[1],
60                           cv=LeaveOneOut())
61 print('交叉验证（逐个测试）得分情况: \n', scores)
62 print('平均分: \n', scores.mean())
63 print('用时（秒）: ', time()-start)

```

结果展示

对于处于这样一个机遇与挑战并存的时代的当代大学生，在就业方面难免会有相应的压力，毕竟需要大多数的体力活都被或者将被机器和算法替代，因此作为当代大学生，更应该在大学期间，以能力培育为基础，需要学会的知识，要学精，并且要树立终身学习的目标。