

第一次实验报告目录

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的感想等

第二次实验报告目录

(1)预处理
(2)分词与词性标注实验
 逆向最大匹配算法
 HanLp分词和词性标注
 jieba精准模式分词和词性标注
 人工分词
 不同算法分词的性能比较
 jieba分词与人工分词性能比较
 HanLP分词与人工分词性能比较
(3)命名实体识别实验
 jieba分词两种算法的命名实体识别
 进一步错误分析
 附加题

第一次实验报告目录

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
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
         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]

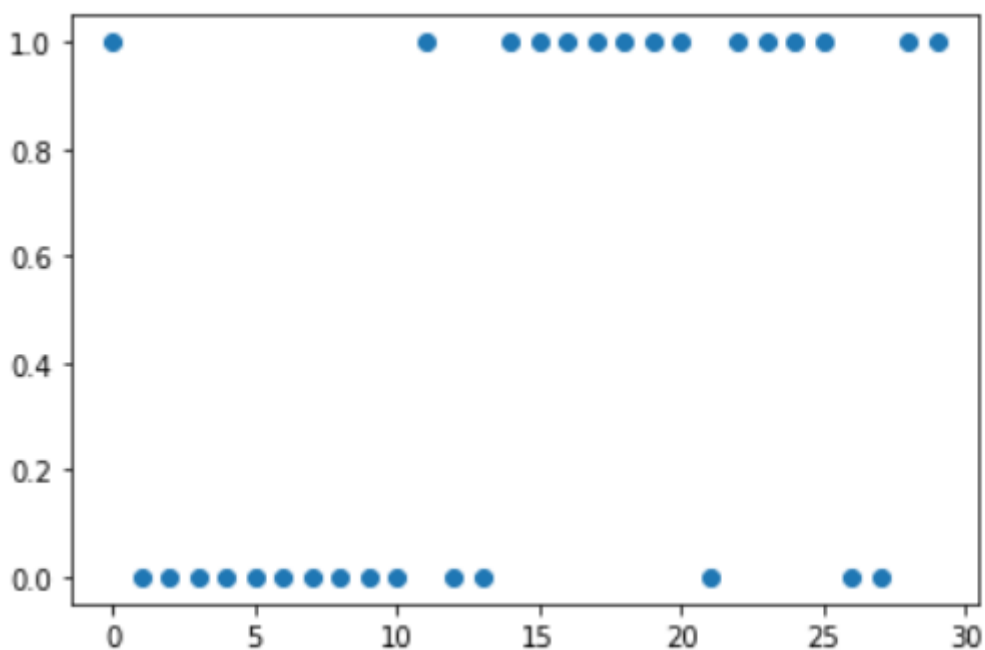
```

```

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]
```

```

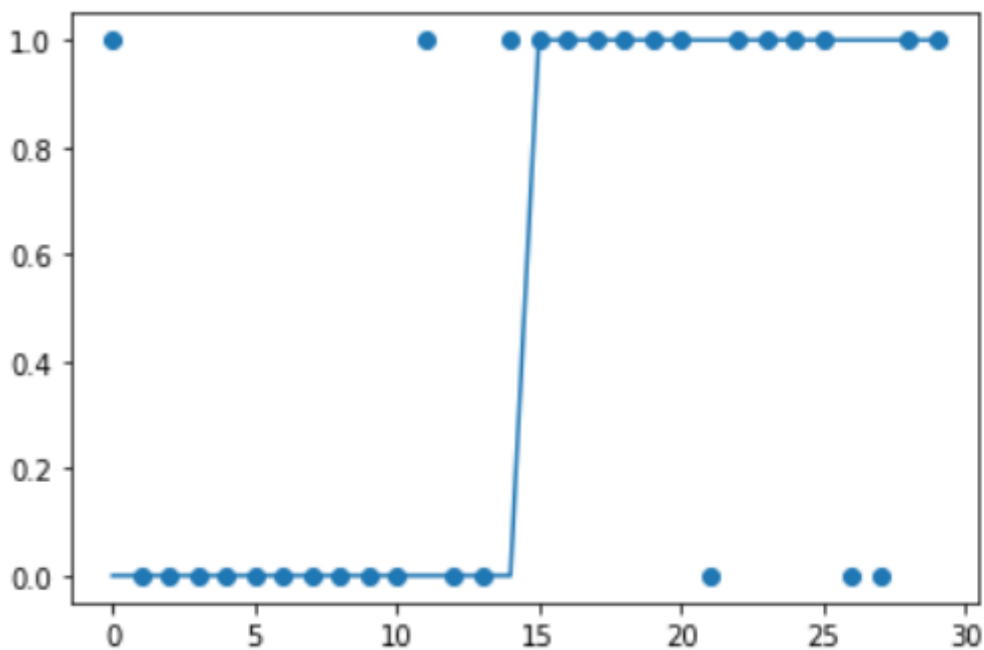
1 # 未知数据属于某一种类别的概率
2 print(reg.predict_proba([[5],[19]]))

```

```
[[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')
```

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'【】0-9、-、，、！~\*']', '', 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\\'+
      str(i) +'.txt' for i in range(151)]
8      # 获取训练集中所有邮件中的全部单词
9      for txtFile in txtFiles:
10         allWords.append(getWordsFromFile(txtFile))
11     # 获取并返回出现次数最多的前topN个单词
12     freq = Counter(chain(*allWords))
13     return [w[0] for w in freq.most_common(topN)]
14
15 # 全部训练集中出现次数最多的前600个单词
16 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]])

```

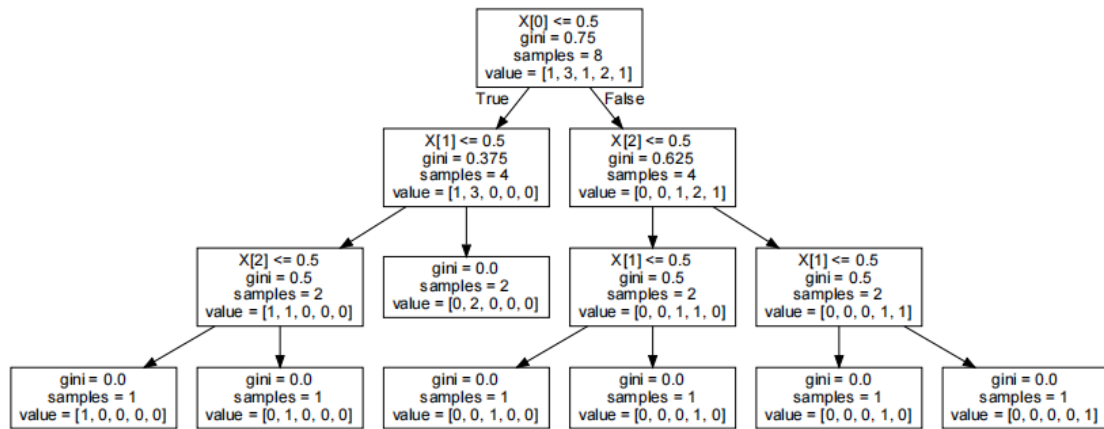
Out[11]: array([2])

```

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
46             fn.endswith('.jpg')]
47         # 按编号排序
48         digitsFile.sort(key=lambda fn: int(basename(fn)[-4]))
49         # digitsData用于存放读取的图片中数字信息
50         # 每个图片中所有像素值存放于digitsData中的一行数据
51         digitsData = []
52         for fn in digitsFile:
53             with Image.open(fn) as im:
54                 # getpixel()方法用来读取指定位置像素的颜色值
55                 data = [sum(im.getpixel((w,h)))/len(im.getpixel((w,h)))
56                     for w in range(width)
57                     for h in range(height)]
58                 digitsData.append(data)
59
60         # digitsLabel用于存放图片中数字的标准分类
61         with open(dstDir+'\\digits.txt') as fp:
62             digitsLabel = fp.readlines()
63         # 删除数字字符两侧的空白字符
64         digitsLabel = [label.strip() for label in digitsLabel]
65         return (digitsData, digitsLabel)
66
67     # 生成图片文件
68     generatedDigits(num=1000)
69     # 加载数据
70     data = loadDigits()
71     print('数据加载完成。')
72
73     # 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
74     X_train, X_test, y_train, y_test = train_test_split(data[0], data[1],
75         test_size=0.1)
76     # 创建并训练模型
77     svcClassifier = svm.SVC(kernel="linear", C=1000, gamma=0.001)
78     svcClassifier.fit(X_train, y_train)
79     print('模型训练完成。')
80
81     # 使用测试集对模型进行评分
82     score = svcClassifier.score(X_test, y_test)
83     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

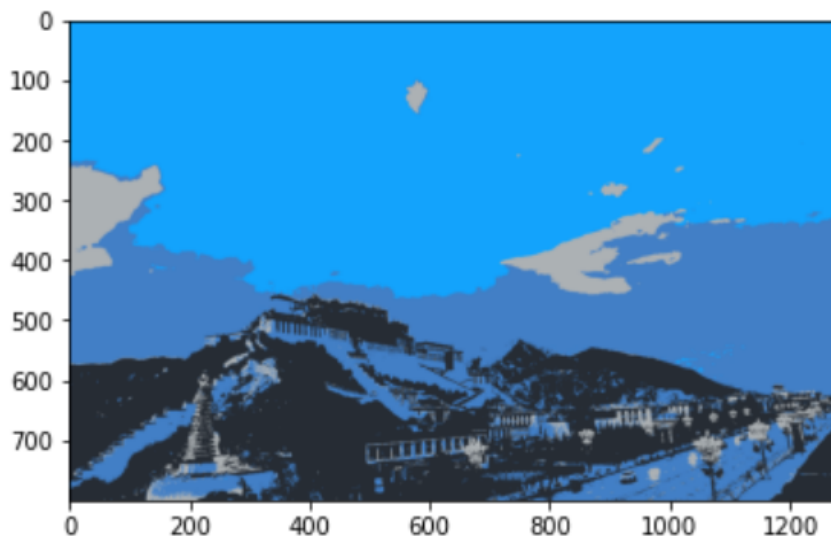
[931 487 764 741]:高铁
[46 194 77 975]:普通火车
[402 542 355 784]:高铁
[182 781 446 366]:飞机
[685 867 805 203]:飞机
[45 156 989 639]:高铁
[620 304 919 515]:高铁
[125 925 922 749]:高铁
[536 918 546 642]:飞机
[27 755 948 288]:飞机
[232 32 265 662]:普通火车
[975 243 921 72]:高铁
[869 941 104 96]:飞机
[827 449 849 867]:高铁
[334 49 147 106]: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

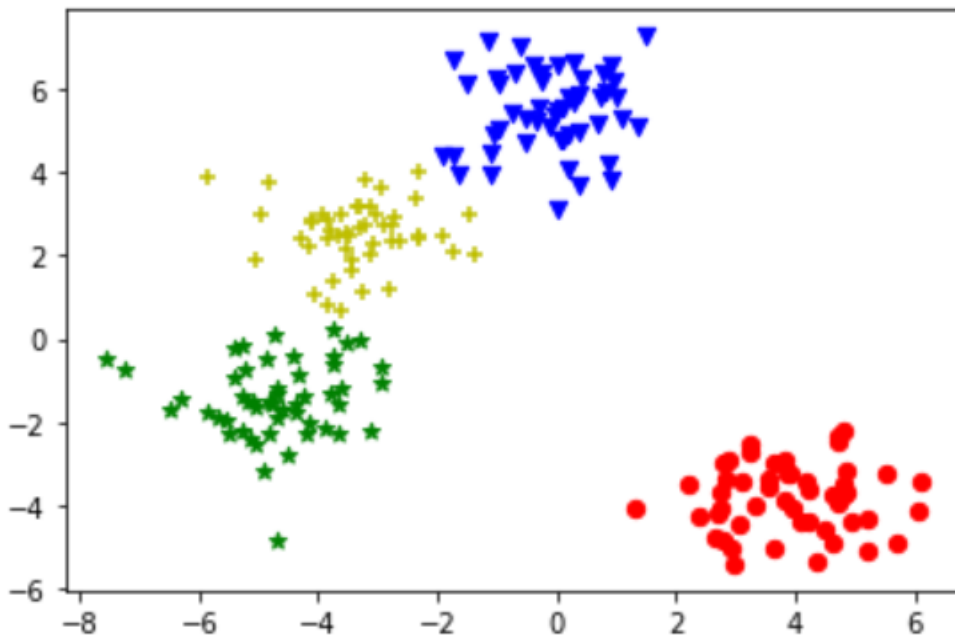
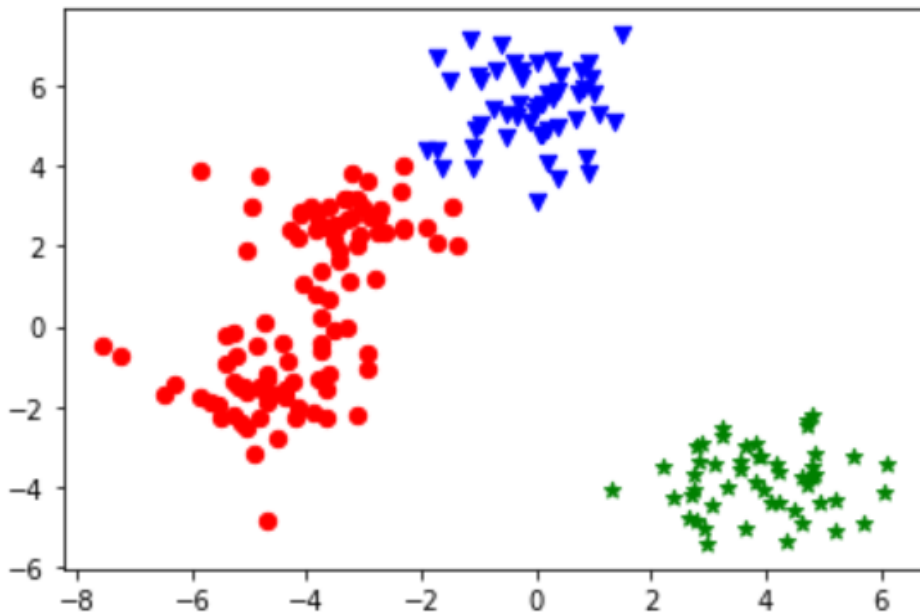
- 分层聚类又称为系统聚类算法或者系谱聚类算法，这个方法把所有样本都看做各自一类，定义类间距计算方式，选择距离最小的一对元素合并成一个新的类，重新计算各个类之间的距离并重复上面的步骤，直到所有原始元素划分为指定数量的类。
- 这个类计算的复杂度非常高，不适合大数据聚类问题
- 代码展示：

```
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]
```



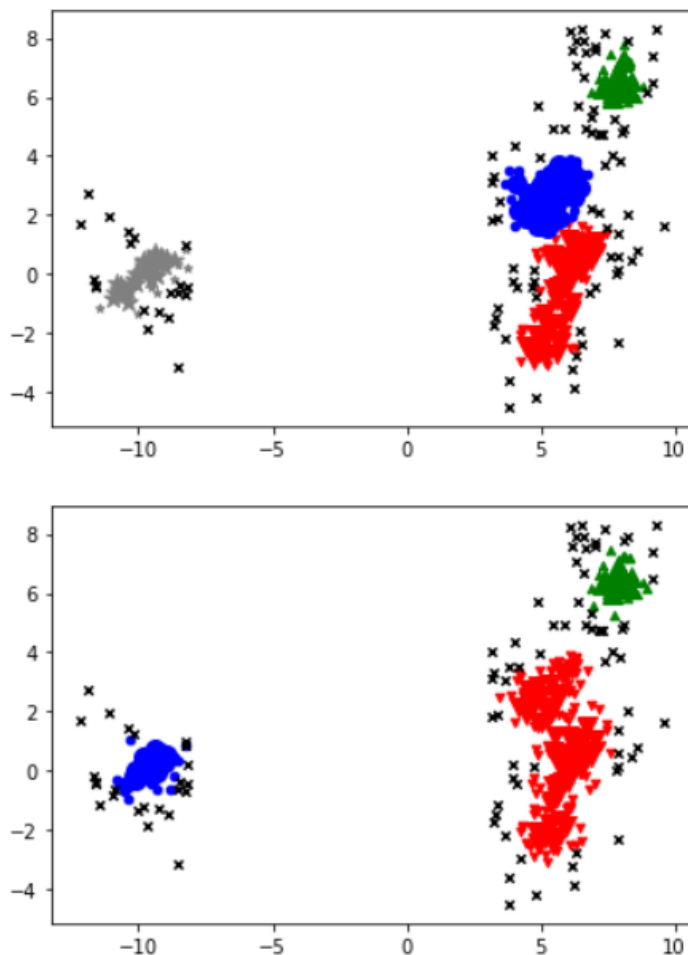
DBSCAN算法验证实验

- 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评分最高的进行推荐
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.xlsx').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

当前人工智能正在迅猛发展，主要表现在人工智能产业规模稳步上升，数字化经济不断发展，国家和地方政府顺应时代要求，出台了很多有利于人工智能发展的帮扶政策，在这样的大背景下，市场对人工智能行业的投资热情不断上涨，相关的技术也在不断的成熟。虽然我国的人工智能产业起步比较晚，但是顺应这历史的发展，也取得了不俗的成就。经过中国信息通信研究院的测算，我国2021的AI产业规模达到434亿美元，同比增长15%。其中，人工智能最被关注的点在于它促进了传统产业的转型升级。由此产生很多关于它的应用，比如“AI+交通”，“AI+医疗”，“AI+零售”等。以交通为例，在交通工具上，AI通过深度学习，学习交通规则、利用计算机视觉和人工大脑，快速识别路面情况并且做出正确的选择；在出行导航上，AI可以实时识别道路情况、掌握分析并计算最佳道路、反馈道路信息（如堵车、封路等），这不仅仅能合理利用交通资源，还能减少事故发生率。当然，诸如“AI+交通”这样的例子还有很多。但是不可避免的是，如今的AI产业虽然有所作为，但是成熟的应用还是十分欠缺的，但是正是这一不成熟，给人工智能产业带来无数的机遇。在未来，人工智能算法算力的深入、人工智能神经网络发展、人工智能伦理问题等，都将是需要解决而必须解决的问题。

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，以能力培养为基础，需要学会的知识，要学精，并且要树立终身学习的目标。

第二次实验报告目录

实验名称：

中文文本分词、词性标注与命名实体识别算法综合实验

实验内容如下：

1. 实验数据：200篇新闻
2. 实验步骤：

(1) 预处理：将标题和正文中的所有可能包含的html格式符号清除干净，并按照句号和问号进行分行显示（每一行为一句话）

(2) 分词与词性标注实验：

① 使用逆向最大匹配分词算法、[HanLP](#)以及jieba分词的精确分词模式对预处理之后的实验数据200篇新闻文本进行[分词与词性标注](#)以及jieba分词的精确分词模式对预处理之后的实验数据200篇新闻文本进行分词与词性标注；

② 并选取2篇新闻【按照学号来进行选取，比如学号1结尾的同学，选取文件名字以1结尾的两篇新闻文本；比如学号2结尾的同学，选取文件名字以2结尾的两篇新闻文本，等等】，人工标注分词结果，只需要标注分词，不需要标注词性；

③ 然后对不同算法的分词结果进行性能比较（正确分词的词数目/这2篇新闻的人工标注分词的词总数），并进行错误分析（将人工标注结果与分词错误结果列在一张表格中，从而对比出这几个分词算法各自或者共同的分词错误样例）。

(3) 命名实体识别实验：

① 用jieba分词工具，根据词性获取实体对象，从而完成命名实体识别（人名、地名、机构名）实验，【<https://blog.csdn.net/fgg1234567890/article/details/115315068>】

nr 人名 名词代码n和“人(ren)”的声母并在一起

ns 地名 名词代码n和处所词代码s并在一起

nt 机构团体“团”的声母为t，名词代码n和t并在一起

nz 其他专名“专”的声母的第1个字母为z，名词代码n和z并在一起

② 并人工标注分词实验中的人工标注的那2篇新闻的命名实体识别的结果，计算命名实体识别准确率（正确命名实体数目/这2篇新闻的人工标注命名实体总数），进一步做错误分析

③ 【附加题】用课件中HMM算法【可参考：<https://www.cnblogs.com/lokvahkoor/p/12781056.html>】来进行命名实体识别的模型训练与测试，【训练数据可以用：实验数据/第5章/1980_01rmb.txt，已上传在005-006-实验代码参考与数据集.zip】并对比出jieba与HMM模型各自或者共同的实体识别错误样例；

(1)预处理

要求：将标题和正文中的所有可能包含的html格式符号清除干净，并按照句号和问号进行分行显示（每一行为一句话）

1. 首先打开获取到文本的路径，因为文本的命名不是连续的，所以用的是 `os.listdir(path)` 来获取到路径的链表：

```

1 # 目录
2 path = "D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集\\第二次实验-分词实验数据"
3 # 获取目录下所有的文件的名字
4 dirs = os.listdir( path )

```

2. 获得到的路径，打开文本对文本去除html格式和空行，这里用的是字符串处理的 `replace()` 函数

```

1 line=line.replace('&nbsp','').replace('&lt;','').replace('&gt;','').replace('&amp','').replace('&quot','').replace('&copy','').replace('&reg','')

```

3. 去除html和空行之后，使用正则表达式对文本切割，如：

```

1 line_write=re.split(r'。|? ',line)

```

4. 切割文本后，装载到一个列表中，并把收集到的数据逐行写到新的文本中去

```

1 # 写数据
2 with open(aft_path,'w',encoding='gb18030') as txt:
3     print(f'打开{aft_path}文件')
4     for line in file_string:
5         txt.write(line+'\n')
6     print(f'打开{aft_path}文件')
7     txt.close()

```

5. 以上的全部代码总和：

```

1 import os
2 import re
3
4 def txt_extraction(bef_path,aft_path):
5     """提取文本，并把提取到的文本写到新的文件中"""
6     file_string=[]
7     # 打开文件
8     with open(bef_path,encoding='gb18030') as txt:
9         print(f'打开{bef_path}文件')
10        for line in txt:
11            # 去除段落和空格换行
12            line=line.strip()
13            # 去除空行
14            if line=='':
15                continue
16            # 查找html格式内容，并去除
17
18            line=line.replace('&nbsp','').replace('&lt;','').replace('&gt;','').replace('&amp','').replace('&quot','').replace('&copy','').replace('&reg','')
19
20            # 按照句号或者问号切割
21            line_write=re.split(r'。|? ',line)
22            for i in line_write:
23                # 获取到全部内容并保存
24                file_string.append(i)

```

```

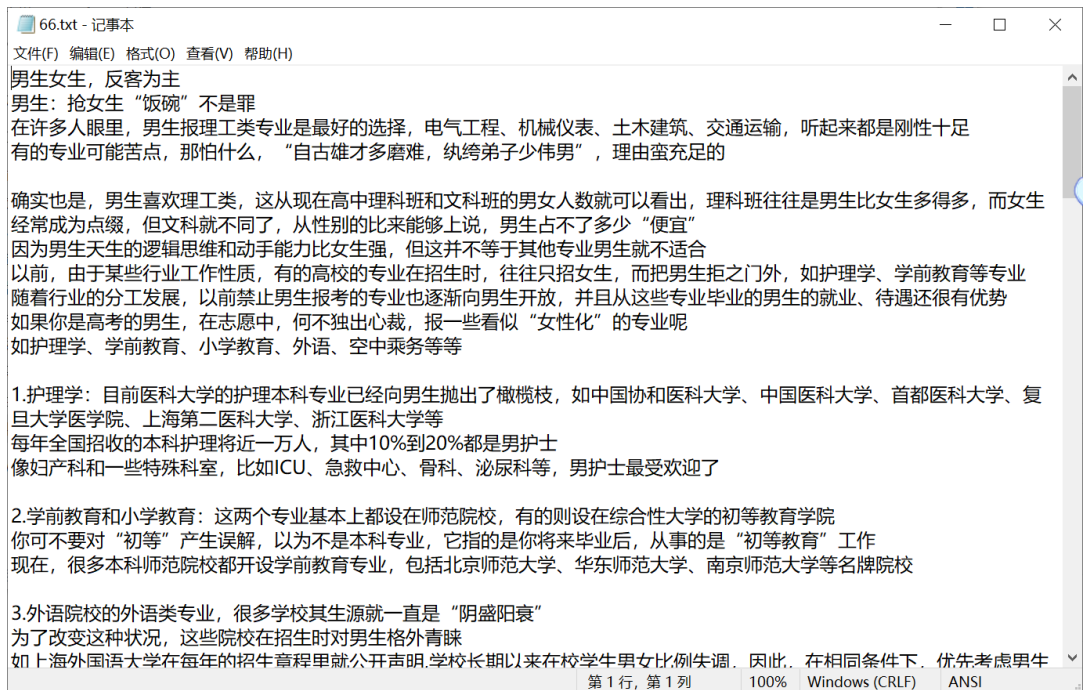
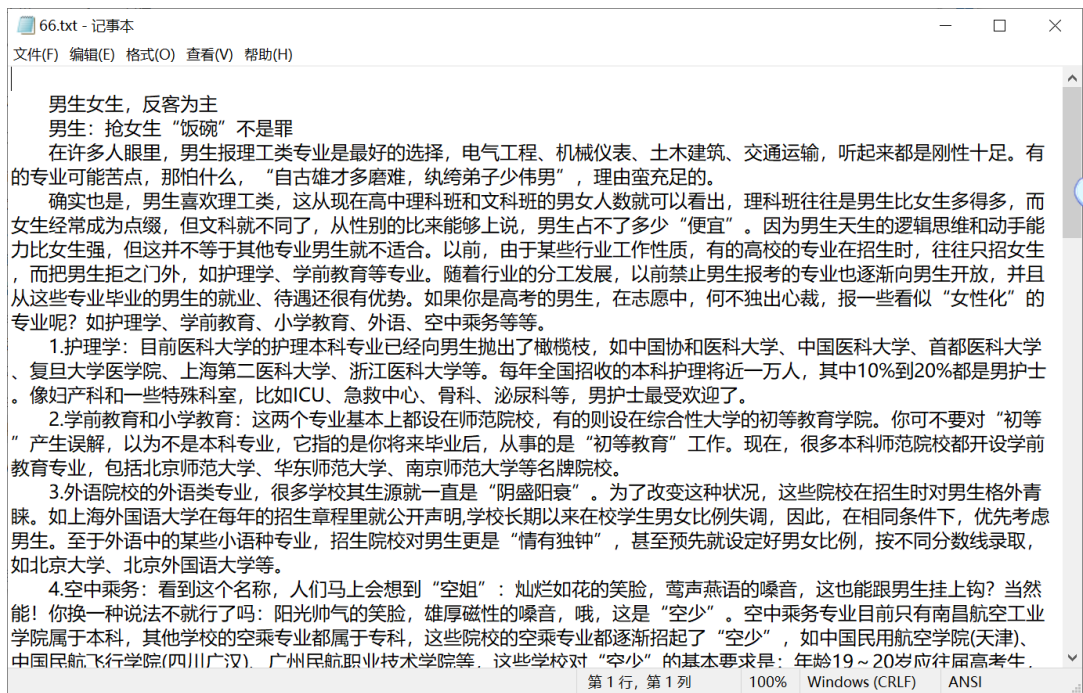
23 #         print(file_string)
24         print(f'关闭{bef_path}文件')
25         txt.close()
26
27 #         # 文件不存在，新建文件
28 #         if not os.path.exists(aft_path):
29 #             os.makedirs(aft_path)
30 #             print('目录创建成功')
31
32         # 写数据
33         with open(aft_path, 'w', encoding='gb18030') as txt:
34             print(f'打开{aft_path}文件')
35             for line in file_string:
36                 txt.write(line+'\n')
37             print(f'打开{aft_path}文件')
38             txt.close()
39
40
41 def main():
42     """主函数"""
43     # 目录
44     path = "D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考
与数据集\\第二次实验-分词实验数据"
45     # 获取目录下所有的文件的名字
46     dirs = os.listdir( path )
47
48     for file in dirs:
49         bef_path=path+'\\'+file
50         aft_path='D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验
代码参考与数据集\\分词实验预处理后数据\\'+file
51         txt_extraction(bef_path,aft_path)
52
53     main()

```

6. 输出结果：在文件处理的时候，我对于打开文件和关闭文件进行了输出路径的操作

打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\100.txt文件
 关闭D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\100.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\100.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\100.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\1004.txt文件
 关闭D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\1004.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\1004.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\1004.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\102.txt文件
 关闭D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\102.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\102.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\102.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\1026.txt文件
 关闭D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\1026.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\1026.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\1026.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\1038.txt文件
 关闭D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\第二次实验-分词实验数据\1038.txt文件
 打开D:\workspaces\AI\ip\人工智能\第二次实验数据\实验代码参考与数据集\分词实验预处理后数据\1038.txt文件

7. 文本处理前后对比



(2)分词与词性标注实验

要求：使用逆向最大匹配分词算法、[HanLP](#)以及jieba分词的精确分词模式对预处理之后的实验数据200篇新闻文本进行[分词与词性标注](#)；

逆向最大匹配算法

这里是参考实例中的逆向最大匹配算法修改后的代码，对于单独成词的数据，这里同样是收录在结果中的。

1. 逆向最大匹配分词算法对预处理后的实验数据100篇新闻文本进行分词
2. 第一步先加载预处理后的实验数据：`dirs = os.listdir(path)` 获取路径下的文件，以`encoding='gb18030'` 格式打开，代码如下：


```

1 def main():
2     path = "D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考
    与数据集\\分词实验预处理后数据\\"
3     dirs = os.listdir(path)
4     for file in dirs:
5         with open(path+file,encoding='gb18030') as txt:
6             text=txt.read()
7             tokenizer = IMM('D:\\workspaces\\AI\\ip\\人工智能\\第二次实
    验数据\\实验代码参考与数据集\\实验分词所用词典-dict.txt.big\\dict.txt.big')
8             print(file,end=':')
9             print(tokenizer.cut(text)[0:10])

```

3. 第二步是加载词典中的内容，去掉空行后，存储在 `set()` 中，并记录其中最长的词的长度，代码如下：

```

1 def __init__(self, dic_path):
2     """
3     获取到dic_path中的词典的内容，并存在dictionary元素集中
4     同步记录字典的最大行数，存储在maximum中
5     """
6     # 创建一个无序不重复元素
7     self.dictionary = set()
8     self.maximum = 0
9     #读取词典
10    with open(dic_path, 'r', encoding='utf8') as f:
11        for line in f:
12            # 切割并去掉空行
13            line = line.strip()
14            if not line:
15                continue
16            self.dictionary.add(line)
17            # 记录最大行数
18            if len(line) > self.maximum:
19                self.maximum = len(line)

```

4. 第三步剪切数据，首先从文本最后取 `maximum` 长度的数据，如果词典有这个数据，就剪切，否则减掉数据前面的一个字，重复上面操作，直到数据长度为1，如果还是匹配不到，就记为单个字，记录在结果中，否则就剪切。代码如下：

```

1 def cut(self, text):
2     """
3     按照字典中的数据对文本进行切割，如果字典中没有对应的数据就单独成字
4     """
5     result = []
6     # 获取到全部的text的内容，只要内容的长度不为0，就一直在搜寻
7     index = len(text)
8     while index > 0:
9         word = None
10        # 获取后面maximum数据，一直到匹配后才结束
11        size[maximum,maximum-1.....1]
12        for size in range(self.maximum, 0, -1):
13            if index - size < 0:
14                continue
15            # 获取到后面maximum的数据碎片
16            piece = text[(index - size):index]

```

```

16         # 字典可以找到数据, 进入if
17         if piece in self.dictionary:
18             word = piece
19             result.append(word)
20             index -= size
21             break
22         # 字典找不到数据, 说明是单独成字, 一样要合并收集
23         if len(piece)==1:
24             # 不要空行, 也不要标点符号
25             if piece==' ':
26                 continue
27             result.append(piece)
28         if word is None:
29             index -= 1
30         return result[::-1]

```

5. 关于输出, 因为直接打印出来的数据过分多, 所以在屏幕下打印每一份文本的前10个数据, 代码如下:

```

1 tokenizer = IMM('D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代
码参考与数据集\\实验分词所用词典-dict.txt.big\\dict.txt.big')
2 print(file, end=':')
3 print(tokenizer.cut(text)[0:10])

```

6. 全部代码如下:

```

1 import os
2 #逆向最大匹配
3 class IMM(object):
4     def __init__(self, dic_path):
5         """
6         获取到dic_path中的词典的内容, 并存在dictionary元素集中
7         同步记录字典的最大行数, 存储在maximum中
8         """
9         # 创建一个无序不重复元素
10        self.dictionary = set()
11        self.maximum = 0
12        #读取词典
13        with open(dic_path, 'r', encoding='utf8') as f:
14            for line in f:
15                # 切割并去掉空行
16                line = line.strip()
17                if not line:
18                    continue
19                self.dictionary.add(line)
20                # 记录最大行数
21                if len(line) > self.maximum:
22                    self.maximum = len(line)
23        def cut(self, text):
24            """
25            按照字典中的数据对文本进行切割, 如果字典中没有对应的数据就单独成字
26            """
27            result = []
28            # 获取到全部的text的内容, 只要内容的长度不为0, 就一直在搜寻
29            index = len(text)
30            while index > 0:

```

```

31         word = None
32         # 获取后面maximum数据，一直到匹配后才结束
size[maximum,maximum-1.....1]
33         for size in range(self.maximum, 0, -1):
34             if index - size < 0:
35                 continue
36             # 获取到后面maximum的数据碎片
37             piece = text[(index - size):index]
38             # 字典可以找到数据，进入if
39             if piece in self.dictionary:
40                 word = piece
41                 result.append(word)
42                 index -= size
43                 break
44             # 字典找不到数据，说明是单独成字，一样要合并收集
45             if len(piece)==1:
46                 # 不要空行，也不要标点符号
47                 if piece==' ':
48                     continue
49                 result.append(piece)
50             if word is None:
51                 index -= 1
52         return result[::-1]
53
54 def main():
55     path = "D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考
与数据集\\分词实验预处理后数据\\"
56     dirs = os.listdir(path)
57     for file in dirs:
58         with open(path+file,encoding='gb18030') as txt:
59             text=txt.read()
60             tokenizer = IMM('D:\\workspaces\\AI\\ip\\人工智能\\第二次实
验数据\\实验代码参考与数据集\\实验分词所用词典-dict.txt.big\\dict.txt.big')
61             print(file,end=':')
62             print(tokenizer.cut(text)[0:10])
63
64 main()

```

7. 输出截图：

这里因为文本编码问题一直都是报错，所以最后改成了 gb18030 格式的文本打开，但是这样的结果就是智能单个字的切割数据


```

100.txt: ['来', '源', ':', '你', '来', '我', '网', ''], ['来', '源', ':', '你', '来', '我', '网', '']
1004.txt: ['来', '源', ':', '你', '来', '我', '网', ''], ['来', '源', ':', '你', '来', '我', '网', '']
102.txt: ['对', '于', '一', '个', '住', '在', '上', '海', '的', '考'], ['对', '于', '一', '个', '住', '在', '上', '海', '的', '考']
1026.txt: ['来', '源', ':', '厦', '门', '大', '学', '招', '生'], ['来', '源', ':', '厦', '门', '大', '学', '招', '生']
1038.txt: ['北', '京', '新', '航', '道', '学', '校', '校', '长', '胡'], ['北', '京', '新', '航', '道', '学', '校', '校', '长', '胡']
1039.txt: [';', '第', '一', '页', '\n', '珊', '瑚', '礁', '孕', '育'], [';', '第', '一', '页', '\n', '珊', '瑚', '礁', '孕', '育']
1057.txt: [';', '\n', '正', '确', '对', '待', '模', '拟', '考', '试'], [';', '\n', '正', '确', '对', '待', '模', '拟', '考', '试']
1060.txt: ['英', '语', '单', '词', '的', '词', '形', '变', '化', '主'], ['英', '语', '单', '词', '的', '词', '形', '变', '化', '主']
108.txt: ['来', '源', ':', '沈', '阳', '新', '东', '方', '学'], ['来', '源', ':', '沈', '阳', '新', '东', '方', '学']
1087.txt: [';', '第', '一', '页', '\n', '现', '代', '文', '阅', '读'], [';', '第', '一', '页', '\n', '现', '代', '文', '阅', '读']
1091.txt: ['北', '京', '大', '学', '光', '华', '管', '理', '学', '院'], ['北', '京', '大', '学', '光', '华', '管', '理', '学', '院']
1101.txt: [';', ';', ';', '话', '题', '主', '持', '\u3000', '本', '报'], [';', ';', ';', '话', '题', '主', '持', '\u3000', '本', '报']
1102.txt: [';', ';', ';', '来', '源', ':', '全', '品', '高'], [';', ';', ';', '来', '源', ':', '全', '品', '高']
1108.txt: [';', ';', ';', '来', '源', ':', '全', '品', '高'], [';', ';', ';', '来', '源', ':', '全', '品', '高']
1111.txt: [';', '第', '一', '页', '\n', '子', '把', '父', '当', '马'], [';', '第', '一', '页', '\n', '子', '把', '父', '当', '马']
1112.txt: ['各', '位', '领', '导', '、', '老', '师', '们', '、', '同'], ['各', '位', '领', '导', '、', '老', '师', '们', '、', '同']
1113.txt: ['1', '、', '具', '有', '人', '格', '魅', '力', '\n', '\n'], ['1', '、', '具', '有', '人', '格', '魅', '力', '\n', '\n']
1117.txt: ['来', '源', ':', '东', '南', '大', '学', '招', '生'], ['来', '源', ':', '东', '南', '大', '学', '招', '生']
1123.txt: ['来', '源', ':', '大', '连', '理', '工', '大', '学'], ['来', '源', ':', '大', '连', '理', '工', '大', '学']
1127.txt: ['来', '源', ':', '内', '蒙', '古', '大', '学', '招'], ['来', '源', ':', '内', '蒙', '古', '大', '学', '招']
1128.txt: ['来', '源', ':', '重', '庆', '大', '学', '招', '生'], ['来', '源', ':', '重', '庆', '大', '学', '招', '生']
1131.txt: ['来', '源', ':', '上', '海', '交', '通', '大', '学'], ['来', '源', ':', '上', '海', '交', '通', '大', '学']
1132.txt: ['来', '源', ':', '东', '华', '大', '学', '网', '站'], ['来', '源', ':', '东', '华', '大', '学', '网', '站']
1143.txt: ['来', '源', ':', '中', '国', '政', '法', '大', '学'], ['来', '源', ':', '中', '国', '政', '法', '大', '学']
1148.txt: ['来', '源', ':', '北', '京', '交', '通', '大', '学'], ['来', '源', ':', '北', '京', '交', '通', '大', '学']
1152.txt: ['来', '源', ':', '北', '京', '交', '通', '大', '学'], ['来', '源', ':', '北', '京', '交', '通', '大', '学']
1155.txt: ['来', '源', ':', '中', '国', '人', '民', '大', '学'], ['来', '源', ':', '中', '国', '人', '民', '大', '学']
116.txt: ['口', '译', '和', '笔', '译', '虽', '然', '同', '属', '翻'], ['口', '译', '和', '笔', '译', '虽', '然', '同', '属', '翻']
118.txt: ['随', '着', '5', '月', '1', '3', '日', '最', '后', '一'], ['随', '着', '5', '月', '1', '3', '日', '最', '后', '一']
119.txt: ['来', '源', ':', '沈', '阳', '新', '东', '方', '学'], ['来', '源', ':', '沈', '阳', '新', '东', '方', '学']
1197.txt: [';', '第', '一', '页', '\n', '一', '、', '社', '会', '公'], [';', '第', '一', '页', '\n', '一', '、', '社', '会', '公']
121.txt: ['第', '一', '招', '计', '划', '答', '题', '时', '间', '\n'], ['第', '一', '招', '计', '划', '答', '题', '时', '间', '\n']
1211.txt: ['一', '位', '读', '者', '给', '我', '们', '来', '电', '说'], ['一', '位', '读', '者', '给', '我', '们', '来', '电', '说']
1218.txt: ['中', '新', '网', '2', '0', '0', '6', '年', '4', '月'], ['中', '新', '网', '2', '0', '0', '6', '年', '4', '月']
1219.txt: ['教', '育', '高', '收', '费', '、', '乱', '收', '费', '已'], ['教', '育', '高', '收', '费', '、', '乱', '收', '费', '已']
1223.txt: ['据', '了', '解', '、', '近', '年', '来', '、', '新', '疆'], ['据', '了', '解', '、', '近', '年', '来', '、', '新', '疆']
1224.txt: ['主', '题', ':', '我', '是', '怎', '样', '以', '4', '1'], ['主', '题', ':', '我', '是', '怎', '样', '以', '4', '1']
1226.txt: ['年', '一', '过', '完', '、', '各', '地', '大', '大', '小'], ['年', '一', '过', '完', '、', '各', '地', '大', '大', '小']
1227.txt: ['中', '学', '历', '史', '课', '的', '内', '容', '、', '纵'], ['中', '学', '历', '史', '课', '的', '内', '容', '、', '纵']
1229.txt: ['2', '7', '、', '5', '5', '%', '心', '理', '反', '应'], ['2', '7', '、', '5', '5', '%', '心', '理', '反', '应']
1234.txt: ['在', '我', '国', '古', '代', '历', '史', '上', '有', '许'], ['在', '我', '国', '古', '代', '历', '史', '上', '有', '许']
1235.txt: [';', 'A', '\n', '来', '源', ':', '点', '点', '英'], [';', 'A', '\n', '来', '源', ':', '点', '点', '英']
1238.txt: ['将', '来', '打', '算', '从', '事', '法', '律', '职', '业'], ['将', '来', '打', '算', '从', '事', '法', '律', '职', '业']
1244.txt: ['2', '5', '岁', '的', '戴', '戈', '正', '忙', '着', '赶'], ['2', '5', '岁', '的', '戴', '戈', '正', '忙', '着', '赶']
1245.txt: [';', '（', '一', '）', '孩', '子', '需', '要', '尊', '重'], [';', '（', '一', '）', '孩', '子', '需', '要', '尊', '重']

```

HanLp分词和词性标注

1. 安装：注意的是，这个需要以来java的jdk环境，如果没有安装jdk会出现报错

```

1 | conda activate test
2 | pip install pyhanlp

```

2. 以命令行形式进行分词交互

```

1 | hanlp segment

```

结果截图：

```

(test) C:\Users\Administrator>hanlp segment
下载 https://file.hanks.com/hanlp/hanlp-1.8.3-release.zip 到 D:\Anaconda3\envs\test\lib\site-packages\pyhanlp\static\hanlp-1.8.3-release.zip
100% 1.8 MiB 1.8 MiB/s ETA: 0 s [=====]
下载 https://file.hanks.com/hanlp/data-for-1.7.5.zip 到 D:\Anaconda3\envs\test\lib\site-packages\pyhanlp\static\data-for-1.8.3.zip
100% 637.7 MiB 75.3 KiB/s ETA: 0 s [=====]
解压 data.zip...
中华人民共和国万岁。
中华人民共和国/ns 万岁/n 。 /w
a
a/nx

```

可以看到运行一切正常。

3. 在anaconda中使用HanLp分词和词性标注

```
1 import os
2 from pyhanlp import *
3
4
5 path="D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集\\分词实验预处理后数据\\"
6 dirs=os.listdir(path)
7 for file in dirs:
8     with open(path+file,encoding='ANSI') as txt:
9         text = txt.read()
10
11         print(file)
12         print(HanLP.segment(text)[0:10])
```

4. 结果截图：

```

100.txt
[【/w, 来源/n, : /w, 济南/ns, 新东方学校/nt, 】, /w, /w, 【/w, 作者/nnt, : /w】
1004.txt
[【/w, 来源/n, : /w, 你/rr, 来/vf, 我/rr, 网/n, 】, /w, /w, 【/w】
102.txt
[对于/p, 一个/mq, 住在/v, 上海/ns, 的/udel, 考生/n, 来说/uls, , /w, 今年/t, 的/udel]
1026.txt
[[/w, 来源/n, : /w, 厦门大学/ntu, 招生办/nis, ]/w,
/w, 2006/m, 年/qt, 普通/a]
1038.txt
[北京/ns, 新航道/nz, 学校/nis, 校长/nnt, 胡敏/nr, 教授/nnt,
/w, 曾几何时/dl, , /w, “/w]
1039.txt
[:/w, 第一页/nz,
/w, 珊瑚礁/n, 孕育/v, 了/u!e, 非同寻常/vl, 的/udel, 生物/n, 多样性/n]
1057.txt
[:/w,
/w, 正确对待/nz, 模拟考试/n, 的/udel, 成绩/n,
/w, 横向/n, 复习/vn, 与/cc]
1060.txt
[英语单词/nz, 的/udel, 词形变化/nz, 主要/b, 是/vshi, 增加/v, 前/f, 后缀/n, , /w, 即/v]
108.txt
[【/w, 来源/n, : /w, 沈阳/ns, 新东方学校/nt, 】, /w, /w, 【/w, 作者/nnt, : /w】
1087.txt
[:/w, 第一页/nz,
/w, 现代文/n, 阅读/v, 和/cc, 作文/n, 的/udel, 复习/vn, 是/vshi]
1091.txt
[北京大学光华管理学院/nt, 副院长/nnt, , /w, 教授/nnt, , /w, 著名/a, 经济学家/nnt, 张维/nr, 迎/v,
/w]
1101.txt
[:::/w, /w, 话题/n, 主持/v, /w, 本报记者/nz, /w, 于建坤/nr,
/w, :::/w]
1102.txt
[:::/w, /w, [ /w, 来源/n, : /w, 全品/nr, 高考网/nz, ]/w,
/w, 2005/m]
1108.txt
[:::/w, /w, [ /w, 来源/n, : /w, 全品/nr, 高考网/nz, ;]/w,
/w, :::/w]
1111.txt
[:/w, 第一页/nz,
/w, 子/ng, 把/pba, 父/ng, 当/p, 马/n, , /w, 父/ng]
1112.txt
[各位/rr, 领导/n, 、 /w, 老师/nnt, 们/k, 、 /w, 同学们/nz, : /w,
/w, 大家/rr]
1113.txt
[l/m, 、 /w, 具有/v, 人格魅力/nz,
/w, “/w, 人格/n, “/w, 一/m, 词/n]
1117.txt
[[/w, 来源/n, : /w, 东南大学/ntu, 招生/vn, 办公室/nis, ]/w,
/w, 根据/p, 《/w]
1123.txt
[[/w, 来源/n, : /w, 大连理工大学/ntu, 招生网/nz, ]/w,
/w, 第一/mq, 章/q, /w]
1127.txt
[[/w, 来源/n, : /w, 内蒙古大学/ntu, 招生网/nz, ]/w,
/w, 为/p, 继续/v, 实施/v]
1128.txt
[[/w, 来源/n, : /w, 重庆大学/ntu, 招生网/nz, ]/w,
/w, 一/m, 、 /w, 总则/n]
1131.txt
[[/w, 来源/n, : /w, 上海交通大学/ntu, 网站/n, ]/w,
/w, 根据/p, 教育部/nt, 颁发/v]
1132.txt
[[/w, 来源/n, : /w, 东华大学/ntu, 网站/n, ]/w,
/w, 根据/p, 教育部/nt, 《/w]
1143.txt
[[/w, 来源/n, : /w, 中国政法大学/ntu, 本科/n, 招生/vn, 信息网/n, ]/w,
/w, 第一/mq]
1148.txt
[[/w, 来源/n, : /w, 北京交通大学/ntu, 招生/vn, 资讯网/nz, ]/w,
/w, 第一/mq, 章/q]
1152.txt
[[/w, 来源/n, : /w, 北京交通大学/ntu, 招生/vn, 资讯网/nz, ]/w,
/w, 北京交通大学/ntu, 招生/vn]
1155.txt
[[/w, 来源/n, : /w, 中国人民大学/ntu, 网站/n, ]/w,
/w, 第一/mq, 章/q, /w]
116.txt
[口译/v, 和/cc, 笔译/n, 虽然/c, 同属/n, 翻译/v, 工作/vn, , /w, 但/c, 两者/rzv]
118.txt
[【/w, 来源/n, : /w, 你/rr, 来/vf, 我/rr, 网/n, 】, /w, /w, 【/w】

```

jieba精准模式分词和词性标注

1. jiaba常用的模式有三种：

- 精确模式，试图将句子最精确地切开，适合文本分析；
- 全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；

- 搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。
- 一般默认是精确模式，`cut_all=False` 是精确模式，`cut_all=True` 是全模式，`cut_for_search` 是搜索引擎模式。
- `jieba.cut` 方法接受三个输入参数：需要分词的字符串；`cut_all` 参数用来控制是否采用全模式；HMM 参数用来控制是否使用 HMM 模型。
- `jieba.cut_for_search` 方法接受两个参数：需要分词的字符串；是否使用 HMM 模型。该方法适合用于搜索引擎构建倒排索引的分词，粒度比较细。

2. 用 `lcut` 生成 list:

`jieba.cut` 以及 `jieba.cut_for_search` 返回的结构都是一个可迭代的 Generator，可以使用 for 循环来获得分词后得到的每一个词语（Unicode）。`jieba.lcut` 对 `cut` 的结果做了封装，l 代表 list，即返回的结果是一个 list 集合。同样的，用 `jieba.lcut_for_search` 也直接返回 list 集合。

3. 获取词性

`jieba` 可以很方便地获取中文词性，通过 `jieba.posseg` 模块实现词性标注。

4. 代码:

```
1 import os
2 # 引入词性标注接口
3 import jieba.posseg as psg
4
5
6 path="D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集\\分词实验预处理后数据\\"
7 dirs=os.listdir(path)
8 for file in dirs:
9     with open(path+file,encoding='ANSI') as txt:
10         text = txt.read()
11         # 精准分词并获取词性,这里输出是文件名和切割后的数据以及词性,因为篇幅问题,这里只展示0~10的词语
12         print(file)
13         print([(x.word,x.flag) for x in psg.lcut(text)[0:10]])
```

5. 输出截图

100.txt
[('【', 'x'), ('来源', 'n'), (':', 'x'), ('济南', 'ns'), ('新东方学校', 'nt'), ('】', 'x'), (' ', 'x'), ('【', 'x'), ('作者', 'n'), (':', 'x')]
1004.txt
[('【', 'x'), ('来源', 'n'), (':', 'x'), ('你', 'r'), ('来', 'v'), ('我', 'r'), ('网', 's'), ('】', 'x'), (' ', 'x'), ('【', 'x')]
102.txt
[('对于', 'p'), ('一个', 'm'), ('住', 'v'), ('在', 'p'), ('上海', 'ns'), ('的', 'uj'), ('考生', 'v'), ('来说', 'u'), (' ', 'x'), ('今年', 't')]
1026.txt
[('【', 'x'), ('来源', 'n'), (':', 'x'), ('厦门大学', 'nt'), ('招生办', 'nt'), (']', 'x'), ('\n', 'x'), ('2006', 'm'), ('年', 'm'), ('普
通', 'nz')]
1038.txt
[('北京', 'ns'), ('新', 'a'), ('航道', 'n'), ('学校', 'n'), ('校长', 'n'), ('胡敏', 'nr'), ('教授', 'n'), ('\n', 'x'), ('曾几何', 'nr'), (' ', 'x')]
1039.txt
[(':', 'x'), ('第一页', 'm'), ('\n', 'x'), ('珊瑚礁', 'n'), ('孕育', 'v'), ('了', 'ul'), ('非同寻常', 'i'), ('的', 'uj'), ('生物', 'n'), ('多样性', 'l')]
1057.txt
[(':', 'x'), ('\n', 'x'), ('正确对待', 'n'), ('模拟考试', 'n'), ('的', 'uj'), ('成绩', 'n'), ('\n', 'x'), ('横向', 'v'), ('复习', 'v'), ('与', 'p')]
1060.txt
[('英语单词', 'n'), ('的', 'uj'), ('词形变化', 'n'), ('主要', 'b'), ('是', 'v'), ('增加', 'v'), ('前', 'f'), ('后缀', 'n'), (' ', 'x'), ('即', 'v')]
108.txt
[('【', 'x'), ('来源', 'n'), (':', 'x'), ('沈阳', 'ns'), ('新东方学校', 'nt'), ('】', 'x'), (' ', 'x'), ('【', 'x'), ('作者', 'n'), (':', 'x')]
1087.txt
[(':', 'x'), ('第一页', 'm'), ('\n', 'x'), ('现代文', 'n'), ('阅读', 'v'), ('和', 'c'), ('作文', 'n'), ('的', 'uj'), ('复习', 'v'), ('是', 'v')]
1091.txt
[('北京大学光华管理学院', 'nt'), ('副', 'b'), ('院长', 'n'), (' ', 'x'), ('教授', 'n'), (' ', 'x'), ('著名', 'a'), ('经济学家', 'n'), ('张维迎', 'nr'), ('\n', 'x')]
1101.txt
[(':', 'x'), (':', 'x'), (':', 'x'), (' ', 'x'), ('话题', 'n'), ('主持', 'v'), ('\u3000', 'x'), ('本报记者', 'n'), ('\u3000', 'x'), ('于建坤', 'nrfg')]
1102.txt
[(':', 'x'), (':', 'x'), (':', 'x'), (' ', 'x'), ('[', 'x'), ('来源', 'n'), (':', 'x'), ('全品', 'n'), ('高考网', 'nz'), (']', 'x')]
1108.txt
[(':', 'x'), (':', 'x'), (':', 'x'), (' ', 'x'), ('[', 'x'), ('来源', 'n'), (':', 'x'), ('全品', 'n'), ('高考网', 'nz'), (':', 'x')]
1111.txt
[(':', 'x'), ('第一页', 'm'), ('\n', 'x'), ('子', 'ng'), ('把', 'p'), ('父', 'ng'), ('当', 'p'), ('马', 'nr'), (' ', 'x'), ('父', 'n')]
1112.txt
[('各位', 'r'), ('领导', 'n'), (' ', 'x'), ('老师', 'n'), ('们', 'k'), (' ', 'x'), ('同学们', 'n'), (':', 'x'), ('\n', 'x'), ('大家', 'n')]
1113.txt
[(':', 'x'), (' ', 'x'), ('具有', 'v'), ('人格魅力', 'i'), ('\n', 'x'), ('\n', 'x'), ('"', 'x'), ('人格', 'n'), ('"', 'x'), ('一', 'm')]
1117.txt
[('[', 'x'), ('来源', 'n'), (':', 'x'), ('东南大学', 'nt'), ('招生', 'v'), ('办公室', 'n'), (']', 'x'), ('\n', 'x'), ('根据', 'p'), ('《', 'x')]
1123.txt
[('[', 'x'), ('来源', 'n'), (':', 'x'), ('大连理工大学', 'nt'), ('招生网', 'nt'), (']', 'x'), ('\n', 'x'), ('第一章', 'm'), (' ', 'x'), ('总则', 'n')]
1127.txt
[('[', 'x'), ('来源', 'n'), (':', 'x'), ('内蒙古大学', 'nt'), ('招生网', 'nt'), (']', 'x'), ('\n', 'x'), ('为', 'p'), ('继续', 'v'), ('实施', 'v')]
1128.txt
[('[', 'x'), ('来源', 'n'), (':', 'x'), ('重庆大学', 'nt'), ('招生网', 'nt'), (']', 'x'), ('\n', 'x'), ('一', 'm'), (' ', 'x'), ('总则', 'n')]
1131.txt
[('[', 'x'), ('来源', 'n'), (':', 'x'), ('上海交通大学', 'nt'), ('网站', 'n'), (']', 'x'), ('\n', 'x'), ('根据', 'p'), ('教育部', 'n'), ('颁发', 'v')]
1132.txt
[('[', 'x'), ('来源', 'n'), (':', 'x'), ('东华大学', 'nt'), ('网站', 'n'), (']', 'x'), ('\n', 'x'), ('根据', 'p'), ('教育部', 'n'), ('《', 'x')]
1143.txt

人工分词

要求：并选取2篇新闻【按照学号来进行选取，比如学号1结尾的同学，选取文件名字以1结尾的两篇新闻文本；比如学号2结尾的同学，选取文件名字以2结尾的两篇新闻文本，等等】，人工标注分词结果，只需要标注分词，不需要标注词性；

说明：学号是1号，这里选取的是121.txt和171.txt两篇文章，因为人工分词的工作量有点大，所以这里使用snowlp分词工具来代替，具体实现逻辑是：对一整个文本数据进行按行切割，切割后的数据按照每一行一个列表的形式存储在result中，方便后面的使用，具体实现代码：

```
1 # SnowNLP小用-用来表示人工切分数据
2 from snowlp import SnowNLP
3 def man_make_result(path):
4     """
5     模拟人工分词
6     """
7     result=[]
8
9
10    with open(path,encoding='ansi') as txt:
11        # 整篇文章
12        text=txt.read()
```

```

13         text_line=text.split()
14         for line in text_line:
15             l=SnowNLP(line)
16             result.append(l.words)
17         #         print(l.words)
18     return result
19
20 # man_make()

```

不同算法分词的性能比较

要求：然后对不同算法的分词结果进行性能比较（正确分词的词数目/这篇新闻的人工标注分词的词总数），并进行错误分析（将人工标注结果与分词错误结果列在一张表格中，从而对比出这几个分词算法各自或者共同的分词错误样例）。

jieba分词与人工分词性能比较

说明：这里输出的顺序是按照每一行来输出，首先输出文件名和人工分词的一行，接着输出jieba分词的一行，如果两行相同输出yes，不同输出相对于人工分词jieba不同的数量以及jieba不同的具体内容（方便比较），最后所有行都已经输出完毕，直接输出jieba相对人工分词的错误率。实现如下：

1. 首先是获取到文件，因为这里只需要两个文件，所以这里定义了一个列表，把两个文件固定住，通过遍历列表的元素就可以获得对应路径；再者一个重要的点就是获取到上面的人工分词的结果集 `result` (这里直接调用上面的函数就可以了)

`result=man_make_result(path+file)`), 作为下面调用函数的一个参数

`jieba_result(result,path+file)` ,其主函数实现如下：

```

1  def main_jieba():
2      files=['121.txt','171.txt']
3      path="D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与
      数据集\\人工分词和分析\\"
4
5      for file in files:
6          print(file)
7          # 人工分词
8          result=man_make_result(path+file)
9          # jieba分词
10         jieba_result(result,path+file)

```

2. jieba分词的划分第一步就是先对文章进行切割，并且按照行的形式对切割后的结果进行保存，并且统计每一行的切割的数目 `sum=sum+len(write_str)` 实现如下：

```

1  # 获取一整篇文章，并且切割成一行，存储成list
2  text=txt.read()
3  text_line=text.split()
4
5  # 对一行进行jieba的精准切割，切割后的分词存储在write_str
6  for line in text_line:
7      temp=psg.lcut(line)
8      write_str=[]
9      # 一整行全部保存在write_str中
10     for i in temp:
11         write_str.append(i.word)
12         print('人工:',result[index])

```

```

13         print('jieba:',write_str)
14         sum=sum+len(write_str)

```

3. jieba分词划分的第二步是把上面得到的每一行的结果 `write_str` 人工分词的对应该行的结果 `result[index]`，判断相同就输出yes，不同就记录不同的数据到 `temp` 列表中，并且统计一行中的不同数据的数量 `j`，最后得到j的累加 `dif_sum`，具体实现如下：

```

1  # 比较一行与人工分词的区别
2  if write_str==result[index]:
3      print('yes')
4      else:
5          j=0
6          temp=[]
7          for i in write_str:
8              if i not in result[index]:
9                  j=j+1
10                 temp.append(i)
11                 print('jieba不同的数量:',j,' jieba不同的数据是: ',temp)
12                 dif_sum=dif_sum+j

```

4. 输出打印错误率

```

1  # 输出错误率
2  print(jieba_path[-7:-1], '的jieba错误率是', dif_sum/sum)

```

5. 全部的实现代码

```

1  import os
2  import jieba.posseg as psg
3
4  # jieba分词后结果
5  def jieba_result(result,jieba_path):
6      """
7      jieba分词后结果
8      index:表示人工分词result的每一行的标号
9      sum:表示jieba分词的分词总数，是每一行切割后列表的长度的和
10     dif_sum:表示jieba和人工的划分的不同的数量的和，在本代码中，是j的和
11     """
12     index=0
13     sum=0
14     dif_sum=0
15     with open(jieba_path,encoding='ansi') as txt:
16         # 获取一整篇文章，并且切割成一行，存储成list
17         text=txt.read()
18         text_line=text.split()
19
20         # 对一行进行jieba的精准切割，切割后的分词存储在write_str
21         for line in text_line:
22             temp=psg.lcut(line)
23             write_str=[]
24             # 一整行全部保存在write_str中
25             for i in temp:
26                 write_str.append(i.word)
27             print('人工:',result[index])
28             print('jieba:',write_str)

```

```

29         sum=sum+len(write_str)
30
31         # 比较一行与人工分词的区别
32         if write_str==result[index]:
33             print('yes')
34         else:
35             j=0
36             temp=[]
37             for i in write_str:
38                 if i not in result[index]:
39                     j=j+1
40                     temp.append(i)
41             print('jieba不同的数量:',j,' jieba不同的数据是: ',temp)
42             dif_sum=dif_sum+j
43             index+=1
44         # 输出错误率
45         print(jieba_path[-7:-1], '的jieba错误率是', dif_sum/sum)
46
47     def main_jieba():
48         files=['121.txt', '171.txt']
49         path="D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集\\人工分词和分析\\"
50
51         for file in files:
52             print(file)
53             # 人工分词
54             result=man_make_result(path+file)
55             # jieba分词
56             jieba_result(result,path+file)
57
58     main_jieba()

```

6. 结果截图：截图太长，只截图一部分

这里我们可以看到，相对于人工分词，121.txt 结巴分词的错误率是 0.1750，171.txt 的错误率是 0.1133

121.txt
 人工: ['第一', '招']
 jieba: ['第一招']
 jieba不同的数量: 1 jieba不同的数据是: ['第一招']
 人工: ['计划', '答', '题', '时间']
 jieba: ['计划', '答题', '时间'] **成对出现**
 jieba不同的数量: 1 jieba不同的数据是: ['答题']
 人工: ['计划', '答', '题', '时间', '保持', '稳定', '的', '答题', '速度']
 jieba: ['计划', '答题', '时间', '保持', '稳定', '的', '答题', '速度']
 jieba不同的数量: 1 jieba不同的数据是: ['保持稳定']
 人工: ['选择', '题', '考试', '通常', '要求', '在', '短', '时间', '内', '作答', '考试', '开始', '时', '你', '应该', '看', '一', '看', '试题', '的分量', '并', '目', '对', '每', '道', '题', '应', '占用', '的', '时间', '迅速', '作出', '估计', '也许', '你', '会', '发现', '每', '道', '题', '选择', '题', '允许', '作答', '的', '时间', '不', '到', '一', '分钟']
 jieba: ['选择题', '考试', '通常', '要求', '在', '短时间', '内', '作答', '考试', '开始', '时', '你', '应该', '看一看', '试题', '的分量', '并', '目', '对', '每', '道题', '应', '占用', '的', '时间', '迅速', '作出', '估计', '也许', '你', '会', '发现', '每', '道', '题', '选择题', '允许', '作答', '的', '时间', '不到', '一分钟']
 jieba不同的数量: 7 jieba不同的数据是: ['选择题', '短时间', '看一看', '道题', '选择题', '不到', '一分钟']
 人工: ['在', '某些', '情况', '下', '这', '似乎', '不大', '可能', '但', '你', '不必', '担心', '有', '不少', '问题', '可能', '只', '需', '几', '秒钟', '就', '可', '作出', '选择', '这样', '你', '就', '有', '足够', '时', '同', '去', '考虑', '相对', '的', '问题', '将', '会', '迎刃而解']
 jieba不同的数量: 1 jieba不同的数据是: ['一条']
 人工: ['考试', '就', '不', '会', '觉得', '太', '难', '了']
 jieba: ['考试', '就', '不会', '觉得', '太', '难', '了']
 jieba不同的数量: 1 jieba不同的数据是: ['不会']
 人工: ['C', '来源', '中华', '会计', '网校']
 jieba: ['C', '来源', '中华', '会计', '网校']
 yes
 121.txt 的jieba错误率是 0.17503331852509996 **121.txt的容错**
 171.txt
 人工: [',', '第一', '页']
 jieba: [',', '第一页']
 jieba不同的数量: 1 jieba不同的数据是: ['第一页'] **不同的输出数据**
 人工: [',']
 jieba: [',']
 yes
 人工: ['不良', '的', '遗传', '因子', '对', '孩子', '的', '智力', '发展', '可能', '是', '致命', '的', '但', '对', '人格', '和', '行为', '方面', '的', '影响', '却', '较', '小']
 jieba: ['不良', '的', '遗传', '因子', '对', '孩子', '的', '智力', '发展', '可能', '是', '致命', '的', '但', '对', '人格', '和', '行为', '方面', '的', '影响', '却', '较', '小'] **相同的输出数据**


```
jieba不同的数量: 4 jieba不同的数据是: ['一种', '家庭教育', '各不相同', '更好']
人工: ['这', '就', '需要', '家长', '主动', '去', '学习', '和', '了解', '摸索', '出', '最佳', 'yes
人工: ['如果', '仅', '凭', '自己', '的', '想法', '率', '性', '而', '为', '弄', '得', '不好', '的', '问题', '行为']
jieba: ['如果', '仅凭', '自己', '的', '想法', '率性', '而', '为', '弄', '得', '不好', '就', '问题', '行为']
jieba不同的数量: 2 jieba不同的数据是: ['仅凭', '率性']
人工: ['而', '一旦', '问题', '行为', '产生', '了', '矫正', '起来', '是', '非常', '的', '困难', '终身', '携带', '']
jieba: ['而', '一旦', '问题', '行为', '产生', '了', '矫正', '起来', '是', '非常', '的', '困难', '终身', '携带', '']
jieba不同的数量: 1 jieba不同的数据是: ['有']
人工: ['来源', '搜', '狐', '母婴', '社区', '2;']
jieba: ['来源', '搜狐', '母婴', '社区', '2;']
jieba不同的数量: 3 jieba不同的数据是: ['搜狐', '2', '']
171.tx 的jieba错误率是 0.11326757874642061
```

HanLP分词与人工分词性能比较

说明：这里输出的顺序是按照每一行来输出，首先输出文件名和人工分词的一行，接着输出HanPL分词的一行，如果两行相同输出yes，不同输出相对于人工分词jieba不同的数量以及jieba不同的具体内容，最后所有行都已经输出完毕，直接输出HanPL相对人工分词的错误率，并且在实现上，和上面jieba分词的算法逻辑是一样的，所以这里我也不重复逻辑了，直接放全部的代码：

```

1 import os
2 from pyhanlp import *
3 # 去掉词性
4 HanLP.Config.ShowTermNature = False
5
6 # HanPL分词错误比较
7 def hanPL_result(result,hanlp_path):
8     """
9     HanPL分词错误比较（与人工）
10    index:表示人工分词result的每一行的标号
11    sum:表示jieba分词的分词总数，是每一行切割后列表的长度的和
12    dif_sum:表示jieba和人工的划分的不同的数量的和，在本代码中，是j的和
13    """
14    index=0
15    sum=0
16    dif_sum=0
17    with open(hanlp_path,encoding='ansi') as txt:
18        # 获取一整篇文章，并且切割成一行，存储成list
19        text=txt.read()
20        text_line=text.split()
21
22        # 对一行进行jieba的精准切割，切割后的分词存储在write_str
23        for line in text_line:
24            temp=HanLP.segment(line)
25            write_str=[]
26            for i in temp:
27                write_str.append(i.word)
28
29            print('人工:',result[index])
30            print('HanPL:',write_str)
31            sum=sum+len(write_str)
32
33            if write_str==result[index]:
34                print('yes')
35            else:
36                j=0

```

```

37         temp=[]
38         for i in write_str:
39             if i not in result[index]:
40                 j=j+1
41                 temp.append(i)
42         print('HanPL不同的数量:',j,' HanPL不同的数据是: ',temp)
43         dif_sum=dif_sum+j
44         # index一定要最后才加1, 不然会出现顺序不一样
45         index+=1
46     # 输出错误率
47     print(hanpl_path[-7:-1], '的HanPL错误率是', dif_sum/sum)
48
49 def main_hanLP():
50     #     files=['121.txt', '171.txt']
51     files=['121.txt']
52     path="D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集
\\人工分词和分析\\"
53
54     for file in files:
55         print(file)
56         # 人工分词
57         result=man_make_result(path+file)
58         # HanPL分词
59         hanPL_result(result,path+file)
60
61 main_hanLP()

```

• 结果截图

这里我们可以看到, 相对于人工分词, 121.txt HanLP分词的错误率是 0.1885, 171.txt 的错误率是 0.1208

121.txt

人工: ['第一', '招']
HanPL: ['第一', '招']

yes

人工: ['计划', '答', '题', '时间']
HanPL: ['计划', '答题', '时间']

HanPL不同的数量: 1 HanPL不同的数据是: ['答题']

人工: ['计划', '答', '题', '时间', '保持', '稳定', '的', '答题', '速度']
HanPL: ['计划', '答题', '时间', '保持', '稳定', '的', '答题', '速度']

HanPL不同的数量: 1 HanPL不同的数据是: ['保持稳定']

人工: ['选择', '题', '通常', '要求', '在', '短', '时间', '内', '作答', '考试', '开始', '时', '你', '应该', '看', '一', '看', '试题', '的', '分量', '并', '目', '对', '每', '道', '题', '应', '占用', '的', '时间', '迅速', '作出', '估计', '也许', '你', '会', '发现', '每', '道', '题', '选择', '题', '允许', '作答', '的', '时间', '不', '到', '一', '分钟']
HanPL: ['选择题', '考试', '通常', '要求', '在', '短时间', '内', '作答', '考试', '开始', '时', '你', '应该', '看一看', '试题', '的', '分量', '并', '目', '对', '每', '道', '题', '应', '占用', '的', '时间', '迅速', '作出', '估计', '也许', '你', '会', '发现', '每', '道', '题', '选择题', '允许', '作答', '的', '时间', '不到', '一', '分钟']

HanPL不同的数量: 2 HanPL不同的数据是: ['第', '十五']

人工: ['把握', '考试', '重点']
HanPL: ['把握', '考试', '重点']

yes

人工: ['任何', '课程', '考试', '都', '有', '大纲', '也', '都', '有', '一', '条', '主线', '都', '有', '一定', '的', '重点', '如果', '我们', '能够', '把握', '这', '条', '主线', '这些', '重点', '也就是说', '抓住', '课程', '的', '精髓', '许多', '的', '问题', '将', '会', '迎刃而解']
HanPL: ['任何', '课程', '考试', '都', '有', '大纲', '也', '都', '有', '一', '条', '主线', '都', '有', '一定', '的', '重点', '如果', '我们', '能够', '把握', '这', '条', '主线', '这些', '重点', '也就是说', '抓住', '课程', '的', '精髓', '许多', '的', '问题', '将', '会', '迎刃而解']

yes

人工: ['考试', '就', '不', '会', '觉得', '太', '难', '了']
HanPL: ['考试', '就', '不会', '觉得', '太难', '了']

HanPL不同的数量: 2 HanPL不同的数据是: ['不会', '太难']

人工: ['来源', '中华', '会计', '网校']
HanPL: ['来源', '中华', '会计', '网校']

yes

121.txt 的HanPL错误率是 0.18853893263342084

人工: ['这', '就', '需要', '家长', '主动', '去', '学习', '和', '了解', ' ', ' ', '摸索', '出', '最佳', '的', '的', '教养', '方法']
HanPL: ['这', '就', '需要', '家长', '主动', '去', '学习', '和', '了解', ' ', ' ', '摸索', '出', '最佳', '的', '的', '教养', '方法']
yes
人工: ['如果', '仅凭', '自己', '的', '想法', '率', '性', '而', '为', ' ', ' ', '弄得', '不好', '就', '会', ' ', ' ', ' ', '造就', ' ', ' ', '孩子', '的', '问题', '行为']
HanPL: ['如果', '仅凭', '自己', '的', '想法', '率性', '而', '为', ' ', ' ', '弄得', '不好', '就', '会', ' ', ' ', ' ', '造就', ' ', ' ', '孩子', '的', '问题', '行为']
HanPL不同的数量: 4 HanPL不同的数据是: ['仅凭', '率性', '而', '为', ' ', '弄得']
人工: ['而', '一旦', '问题', '行为', '产生', '了', ' ', ' ', '矫正', '起来', '是', '非常', '的', '困难', ' ', ' ', '有的', '孩子', '甚至', ' ', ' ', '终身', '携带', ' ', ' ']
HanPL: ['而', '一旦', '问题', '行为', '产生', '了', ' ', ' ', '矫正', '起来', '是', '非常', '的', '困难', ' ', ' ', '有的', '孩子', '甚至', ' ', ' ', '终身', '携带', ' ', ' ']
yes
人工: ['来源', ' ', ' ', '搜', '狐', '母婴', '社区', '2']
HanPL: ['来源', ' ', ' ', '搜狐', '母婴', '社区', '2']
HanPL不同的数量: 3 HanPL不同的数据是: ['搜狐', '2']
171.tx 的HanPL错误率是 0.12078380982974622

(3)命名实体识别实验

jieba分词两种算法的命名实体识别

要求: 用jieba分词工具, 根据词性获取实体对象, 从而完成[命名实体识别 \(人名、地名、机构名\) 实验](#)

- nr 人名 名词代码n和“人(ren)”的声母并在一起
- ns 地名 名词代码n和处所词代码s并在一起
- nt 机构团体“团”的声母为t, 名词代码n和t并在一起
- nz 其他专名“专”的声母的第1个字母为z, 名词代码n和z并在一起

1. jieba.analyse.extract_tags(sentence, topK=5, withweight=True, allowPOS=())

- o sentence 需要提取的字符串, 必须是str类型, 不能是list
- o topK 提取前多少个关键字
- o withWeight 是否返回每个关键词的权重
- o llowPOS是允许的提取的词性, 默认为allowPOS='ns', 'n', 'vn', 'v', 提取地名、名词、动名词、动词
- o jieba.analyse.extract_tags()提取关键字的原理是使用TF-IDF算法

```
1 with open(path+file,encoding='ansi') as txt:
2     text=txt.read()
3     kw = jieba.analyse.extract_tags(text, topK=20,
withweight=True, allowPOS=('nr', 'ns', 'nt', 'nz'))
4     print('extract_tags-----')
5     for item in kw:
6         if(item not in result1):
7             result1.append(item)
8             print(item[0], item[1])
```

2. jieba.analyse.texttrank(sentence, topK=20, withweight=False, allowPOS=('ns', 'n', 'vn', 'v'))

- o 直接使用, 接口相同, 注意默认过滤词性。
- o 基本思想
 1. 将待抽取关键词的文本进行分词
 2. 以固定窗口大小(默认为5, 通过span属性调整), 词之间的共现关系, 构建图
 3. 计算图中节点的PageRank, 注意是无向带权图

```

1 print('textrank-----')
2     kw =
jieba.analyse.textrank(text,topK=20,withweight=True,allowPOS=('nr',
'ns','nt','nz'))
3     for item in kw:
4         if(item not in result2):
5             result2.append(item)
6             print(item[0], item[1])

```

3. 所有的代码

```

1 import jieba
2 import jieba.analyse
3 import jieba.posseg as posg
4 import os
5
6
7 # files=['121.txt','171.txt']
8 path="D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集\\分词实验预处理后数据\\"
9 files=os.listdir(path)
10 for file in files:
11     print(file)
12     result1=[]
13     result2=[]
14     with open(path+file,encoding='ansi') as txt:
15         text=txt.read()
16         kw = jieba.analyse.extract_tags(text, topK=20,
withweight=True, allowPOS=('nr', 'ns','nt','nz'))
17         print('extract_tags-----')
18         for item in kw:
19             if(item not in result1):
20                 result1.append(item)
21                 print(item[0], item[1])
22
23         print('textrank-----')
24         kw =
jieba.analyse.textrank(text,topK=20,withweight=True,allowPOS=('nr',
'ns','nt','nz'))
25         for item in kw:
26             if(item not in result2):
27                 result2.append(item)
28                 print(item[0], item[1])

```

4. 结果截图

Prefix dict has been built successfully.	
extract_tags-----	
英语 0.8245147877833333	
托福 0.5980657464444444	
中学英语 0.489167795237037	
英语水平 0.4741505690111113	
范文 0.455231101462963	
汉语 0.43843691480722224	
求教 0.38647932404444446	
普渡 0.2574199565185185	
英语口语 0.2574199565185185	
英音 0.2574199565185185	
师夷 0.2574199565185185	
小师弟 0.2445838976185185	
英汉词典 0.23707528450555557	
新东方学校 0.23174783872037036	
查字典 0.23174783872037036	
李国锋 0.22138458338703704	
莘莘学子 0.22138458338703704	
谈何 0.22138458338703704	
云深 0.22138458338703704	
文武 0.4989796344974108	
1004.txt	
extract_tags-----	
英语 1.0	
济南 0.6708649695988823	
登门 0.6708649695988823	
孙子兵法 0.6708649695988823	
牵强附会 0.6708649695988823	
新东方学校 0.6661837677808007	
求教 0.6661837677808007	
上云 0.6661837677808007	
别太 0.6661837677808007	
钟道隆 0.5060458398006018	
练武 0.4989796344974108	
extract_tags-----	
子罕 0.8286229037865169	
热爱祖国 0.4948452087235955	
青少年 0.45652393856348317	
广大干部 0.42780454010337077	
传统美德 0.40816578143932586	
爱国主义 0.40633102236449437	
道德 0.4006845191196629	
胡锦涛 0.3633063591914607	
中华民族 0.3631727964773034	
宝贝 0.34411284275550563	
严以律己 0.3123747786966292	
辛勤劳动 0.23520652691011235	
道德规范 0.22112886834674156	
中国 0.20408903505573034	
世界观 0.19977623276134832	
珍宝 0.19683177742629215	
改革开放 0.17600566435348317	
荣誉 0.16954089868044944	
宝玉 0.15672136221910113	
中国共产党 0.1504837833694382	
extract_tags-----	
子罕 1.0	
中国 0.7410892771690482	
青少年 0.6848326867220925	
邓小平理论 0.6848326867220925	
道德 0.5540004074627747	
广大干部 0.5275131325596765	
肖秀荣 0.4634127992999366	
陶冶情操 0.4634127992999366	
爱国主义 0.4634127992999366	
门第 0.4634127992999366	
林海 0.4601265173335302	
启迪 0.4601265173335302	
弘扬 0.4601265173335302	
荣誉 0.4601265173335302	
传统美德 0.4542731332363206	

两种同时打印

进一步错误分析

要求：并人工标注分词实验中的人工标注的那2篇新闻的命名实体识别的结果，计算命名实体识别准确率（正确命名实体数目/这2篇新闻的人工标注命名实体总数），进一步做错误分析

1. 首先对人工按行进行关键字提取

```
1 def man_make_result_word(line):
2     from snownlp import SnowNLP
3     s = SnowNLP(line)
4     a=s.keywords(10)
5     return a
```

2. 用jieba工具进行关键字提取

```
1 kw = jieba.analyse.extract_tags(line, topk=20, withweight=True,
allowPOS=('nr', 'ns','nt','nz'))
```

3. 统计数据，其中 `result` 是用来表示人工分词的按行提取关键字的结果，`result1` 避免输出重复的数据

```
1 # result=man_make_result_word(line)
2 # result1=[]
3 for item in kw:
4     if item[0] in result:
5         sum+=1
6     if item not in result1:
7         result1.append(item)
8     print(item[0],item[1])
```

4. 输出正确率

```
1 print('正确率: ',sum/manmake_sum)
```

5. 全部代码

```
1 def man_make_result_word(line):
2     from snownlp import SnowNLP
3     s = SnowNLP(line)
4     a=s.keywords(10)
5     return a
6
7 import jieba
8 import jieba.analyse
9 import jieba.posseg as posg
10 import os
11
12 files=['121.txt','171.txt']
13 path="D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集\\人工分词和分析\\"
14 for file in files:
15     print(file)
16     # 人工切分的结果
17     result= []
18     manmake_sum=0
19     sum=0
20
21     result1=[]
22     with open(path+file,encoding='ansi') as txt:
23         text=txt.read()
24         text_line=text.split()
25         for line in text_line:
26             result=man_make_result_word(line)
27             manmake_sum+=len(result)
28             kw = jieba.analyse.extract_tags(line, topK=20,
29 withweight=True, allowPOS=('nr', 'ns','nt','nz'))
30             for item in kw:
31                 if item[0] in result:
32                     sum+=1
33                 if item not in result1:
34                     result1.append(item)
35                     print(item[0],item[1])
36         print('正确率: ',sum/manmake_sum)
```

6. 输出结果

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\Administrator\AppData\Local\Temp\jieba.cache
Loading model cost 1.691 seconds.
Prefix dict has been built successfully.
```

```
周密 8.50251495046
高明 8.30596627237
高水平 7.95003509939
托运 11.2616203224
周转 7.75649201785
中华 7.21232293803
正确率: 0.008674101610904586
171.txt
长大 7.09606313191
小岳 12.2912397395
小岳 6.14561986975
音乐学院 4.472425297185
小岳 8.194159826333333
安静 2.4935609079433334
东西 4.77328420082
那小岳 11.9547675029
乌黑 5.9401633768
小玮 3.9849225009666664
小玮 5.97738375145
爸爸妈妈 4.784972155845
爸爸妈妈 9.56994431169
小玮 6.831295715942857
顺顺利利 1.7877690415571428
平平安安 1.4999257528999999
爸爸妈妈 1.3671349016699998
小玮 4.78190700116
爸爸妈妈 1.913988862338
幼儿园 1.7204720570860002
原谅 1.6012549635420001
小玮 7.969845001933333
爸爸妈妈 3.1899814372299997
严重性 3.1226927196066665
小玮 8.966075627175
原谅 2.0015687044275
原谅 4.003137408855
长大 2.3653543773033334
阳光 2.20164752641
东西 1.5910947336066668
文小岳 11.9547675029
小美 13.900677652
懊悔莫及 11.9547675029
小美 6.950338826
幼儿园 4.301180142715
幼儿园 8.60236028543
宝贝 7.65651075131
宝贝 3.828255375655
美的 11.9547675029
小美 11.1205421216
范畴 7.42987814819
矫正 9.31571017331
搜狐 8.81308131674
正确率: 0.048863636363636366
```

附加题

要求：【附加题】用课件中HMM算法[可参考](#)来进行命名实体识别的模型训练与测试，【训练数据可以用：[实验数据/第5章/1980_01rmrb.txt](#)，已上传在005-006-实验代码参考与数据集.zip】并对比jieba与HMM模型各自或者共同的实体识别错误样例；

```
1 class HMM(object):
2     def __init__(self):
3         pass
4
5     def try_load_model(self, trained):
6         pass
7
8     def train(self, path):
9         pass
10
11    def viterbi(self, text, states, start_p, trans_p, emit_p):
12        pass
13
14    def cut(self, text):
15        pass
```

```
1 class HMM(object):
2     def __init__(self):
3         import os
4
5         # 主要是用于存取算法中间结果，不用每次都训练模型
6         self.model_file = 'D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实
7         验代码参考与数据集\\实验分词所用词典-dict.txt.big\\hmm_model.pkl'
8
9         # 状态值集合
10        self.state_list = ['B', 'M', 'E', 'S']
11        # 参数加载,用于判断是否需要重新加载model_file
12        self.load_para = False
13
14        # 用于加载已计算的中间结果，当需要重新训练时，需初始化清空结果
15    def try_load_model(self, trained):
16        if trained:
17            import pickle
18            with open(self.model_file, 'rb') as f:
19                self.A_dic = pickle.load(f)
20                self.B_dic = pickle.load(f)
21                self.Pi_dic = pickle.load(f)
22                self.load_para = True
23        else:
24            # 状态转移概率（状态->状态的条件概率）
25            self.A_dic = {}
26            # 发射概率（状态->词语的条件概率）
27            self.B_dic = {}
28            # 状态的初始概率
29            self.Pi_dic = {}
30            self.load_para = False
31
32        # 计算转移概率、发射概率以及初始概率
33    def train(self, path):
```



```

34
35     # 重置几个概率矩阵
36     self.try_load_model(False)
37
38     # 统计状态出现次数, 求p(o)
39     Count_dic = {}
40
41     # 初始化参数
42     def init_parameters():
43         for state in self.state_list:
44             self.A_dic[state] = {s: 0.0 for s in self.state_list}
45             self.Pi_dic[state] = 0.0
46             self.B_dic[state] = {}
47
48             Count_dic[state] = 0
49
50     def makeLabel(text):
51         out_text = []
52         if len(text) == 1:
53             out_text.append('S')
54         else:
55             out_text += ['B'] + ['M'] * (len(text) - 2) + ['E']
56
57         return out_text
58
59     init_parameters()
60     line_num = -1
61     # 观察者集合, 主要是字以及标点等
62     words = set()
63     with open(path, encoding='utf8') as f:
64         for line in f:
65             line_num += 1
66
67             line = line.strip()
68             if not line:
69                 continue
70
71             word_list = [i for i in line if i != ' ']
72             words |= set(word_list) # 更新字的集合
73
74             linelist = line.split()
75
76             line_state = []
77             for w in linelist:
78                 line_state.extend(makeLabel(w))
79
80             assert len(word_list) == len(line_state)
81
82             for k, v in enumerate(line_state):
83                 Count_dic[v] += 1
84                 if k == 0:
85                     self.Pi_dic[v] += 1 # 每个句子的第一个字的状态, 用于计
算初始状态概率
86
87                 else:
88                     self.A_dic[line_state[k - 1]][v] += 1 # 计算转移概率
89                     self.B_dic[line_state[k]][word_list[k]] = \
+ 1.0 # 计算发射概率

```

```

90
91         self.Pi_dic = {k: v * 1.0 / line_num for k, v in
self.Pi_dic.items()}
92         self.A_dic = {k: {k1: v1 / Count_dic[k] for k1, v1 in v.items()}
93                         for k, v in self.A_dic.items()}
94         #加1平滑
95         self.B_dic = {k: {k1: (v1 + 1) / Count_dic[k] for k1, v1 in
v.items()}
96                         for k, v in self.B_dic.items()}
97         #序列化
98         import pickle
99         with open(self.model_file, 'wb') as f:
100             pickle.dump(self.A_dic, f)
101             pickle.dump(self.B_dic, f)
102             pickle.dump(self.Pi_dic, f)
103
104         return self
105
106     def viterbi(self, text, states, start_p, trans_p, emit_p):
107         v = [{}]
108         path = {}
109         for y in states:
110             v[0][y] = start_p[y] * emit_p[y].get(text[0], 0)
111             path[y] = [y]
112         for t in range(1, len(text)):
113             v.append({})
114             newpath = {}
115
116             #检验训练的发射概率矩阵中是否有该字
117             neverSeen = text[t] not in emit_p['S'].keys() and \
118                 text[t] not in emit_p['M'].keys() and \
119                 text[t] not in emit_p['E'].keys() and \
120                 text[t] not in emit_p['B'].keys()
121             for y in states:
122                 emitP = emit_p[y].get(text[t], 0) if not neverSeen else 1.0
123                 #设置未知字单独成词
124                 (prob, state) = max(
125                     [(v[t - 1][y0] * trans_p[y0].get(y, 0) *
126                      emitP, y0)
127                     for y0 in states if v[t - 1][y0] > 0])
128                 v[t][y] = prob
129                 newpath[y] = path[state] + [y]
130                 path = newpath
131
132             if emit_p['M'].get(text[-1], 0) > emit_p['S'].get(text[-1], 0):
133                 (prob, state) = max([(v[len(text) - 1][y], y) for y in
('E', 'M')])
134             else:
135                 (prob, state) = max([(v[len(text) - 1][y], y) for y in states])
136
137             return (prob, path[state])
138
139     def cut(self, text):
140         import os
141         if not self.load_para:
142             self.try_load_model(os.path.exists(self.model_file))
143             prob, pos_list = self.viterbi(text, self.state_list, self.Pi_dic,
self.A_dic, self.B_dic)

```

```
143     begin, next = 0, 0
144     for i, char in enumerate(text):
145         pos = pos_list[i]
146         if pos == 'B':
147             begin = i
148         elif pos == 'E':
149             yield text[begin: i+1]
150             next = i+1
151         elif pos == 'S':
152             yield char
153             next = i+1
154     if next < len(text):
155         yield text[next:]
156
157
```

```
1  hmm = HMM()
2  hmm.train('D:\\workspaces\\AI\\ip\\人工智能\\第二次实验数据\\实验代码参考与数据集\\实
   验分词所用词典-dict.txt.big\\trainCorpus.txt_utf8')
3
4  text = '这是一个非常棒的方案！'
5  res = hmm.cut(text)
6  print(text)
7  print(str(list(res)))
```