姓名：向建宇

学号：1801210687

1. 用贝叶斯算法分析托儿所数据集

主要思想是，先用训练集算出每个类出现的概率，再分别算出每个词在每个类中的概率：

步骤：1.统计每一个类的个数，计算类概率

2.对每个类分别进行词向量计数，并通过增加拉普拉斯平滑后，得到每个类的词概率向量空间

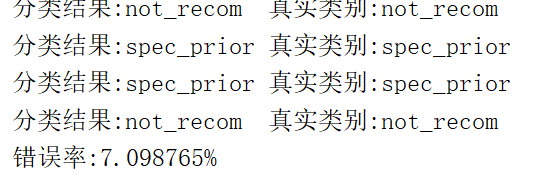
3.返回类概率和词概率向量空间

|  |
| --- |
| **""" 函数说明:朴素贝叶斯分类器训练函数  Parameters:  trainMatrix - 训练文档矩阵，即setOfWords2Vec返回的returnVec构成的矩阵  trainCategory - 训练类别标签向量，即loadDataSet返回的classVec Returns:  p0Vect,p1Vect,p2Vect,p3Vect,p4Vect 词向量顺序与下面的类概率一一对应  Pnot\_recom,Precommend,Pvery\_recom,Ppriority,Pspec\_prior """ def** trainNB0(trainMatrix,trainCategory):  numTrainDocs = len(trainMatrix) *#计算训练的文档数目* numWords = len(trainMatrix[0]) *#计算每篇文档的词条数* num\_not\_recom = 0;num\_recommend = 0;num\_very\_recom = 0;num\_priority = 0;num\_spec\_prior = 0;  **for** word **in** trainCategory:  **if** word == **"not\_recom"**:  num\_not\_recom += 1  **elif** word == **"recommend"**:  num\_recommend += 1  **elif** word == **"very\_recom"**:  num\_very\_recom += 1  **elif** word == **"priority"**:  num\_priority += 1  **elif** word == **"spec\_prior"**:  num\_spec\_prior += 1   *# 文档属于某类的概率* Pnot\_recom = num\_not\_recom/float(numTrainDocs)  Precommend = num\_recommend / float(numTrainDocs)  Pvery\_recom = num\_very\_recom / float(numTrainDocs)  Ppriority = num\_priority / float(numTrainDocs)  Pspec\_prior = num\_spec\_prior / float(numTrainDocs)   *# 创建numpy.ones数组,词条出现数初始化为1，拉普拉斯平滑,注意p0Num-p4Num和上面类对应* p0Num = np.ones(numWords);p1Num = np.ones(numWords);p2Num = np.ones(numWords)  p3Num = np.ones(numWords);p4Num = np.ones(numWords)  p0Denom = 2.0;p1Denom = 2.0;p2Denom = 2.0;p3Denom = 2.0;p4Denom = 2.0 *# 分母初始化为2,拉普拉斯平滑* **for** i **in** range(numTrainDocs):  **if** trainCategory[i] == **"not\_recom"**:  p0Num += trainMatrix[i]  p0Denom += sum(trainMatrix[i])  **elif** trainCategory[i] == **"recommend"**:  p1Num += trainMatrix[i]  p1Denom += sum(trainMatrix[i])  **elif** trainCategory[i] == **"very\_recom"**:  p2Num += trainMatrix[i]  p2Denom += sum(trainMatrix[i])  **elif** trainCategory[i] == **"priority"**:  p3Num += trainMatrix[i]  p3Denom += sum(trainMatrix[i])  **elif** trainCategory[i] == **"spec\_prior"**:  p4Num += trainMatrix[i]  p4Denom += sum(trainMatrix[i])  p0Vect = np.log(p0Num / p0Denom);p1Vect = np.log(p1Num / p1Denom); *# 取对数，防止下溢出* p2Vect = np.log(p2Num / p2Denom);p3Vect = np.log(p3Num / p3Denom);p4Vect = np.log(p4Num / p4Denom)  **return** p0Vect,p1Vect,p2Vect,p3Vect,p4Vect,Pnot\_recom,Precommend,Pvery\_recom,Ppriority,Pspec\_prior |

第二步：通过上面的到概率，分别算出测试集的先验概率，取最大的为预测类即可

|  |
| --- |
| **""" 函数说明:朴素贝叶斯分类器分类函数 Parameters:  vec2Classify - 待分类的词条数组  p0Vect,p1Vect,p2Vect,p3Vect,p4Vect - 对应下面类的概率词向量  Pnot\_recom,Precommend,Pvery\_recom,Ppriorityspec\_prior,Pspec\_prior - 类的概率 Returns:  not\_recom，recommend，very\_recom，priority，spec\_prior """ def** classifyNB(vec2Classify,p0Vect,p1Vect,p2Vect,p3Vect,p4Vect,Pnot\_recom,Precommend,Pvery\_recom,Ppriority,Pspec\_prior):  *#对应元素相乘。logA \* B = logA + logB，所以这里加上log(pClass1)* p0 = sum(vec2Classify \* p0Vect) + np.log(Pnot\_recom)  p1 = sum(vec2Classify \* p1Vect) + np.log(Precommend)  p2 = sum(vec2Classify \* p2Vect) + np.log(Pvery\_recom)  p3 = sum(vec2Classify \* p3Vect) + np.log(Ppriority)  p4 = sum(vec2Classify \* p4Vect) + np.log(Pspec\_prior)  p = max(p0,p1,p2,p3,p4)  **if** p == p0:  **return "not\_recom"  elif** p == p1:  **return "recommend"  elif** p == p2:  **return "very\_recom"  elif** p == p3:  **return "priority"  elif** p == p4:  **return "spec\_prior"** |

结果：测试集取后10%，训练集前90%



2.用线性回归预测ratings值，即论文实验a

参照实验，我分别取了conventional 特征和social media 特征，后面会给出结果

先分析算法：

首先用解析法可知参数为：

C:\Users\xiang\AppData\Local\Temp\1539357809(1).png

但是可能出现欠拟合，所以加入使用核来对待测点（xn）附近的点赋予更高的权重

C:\Users\xiang\AppData\Local\Temp\1539357477(1).png

于是采用局部加权线性回归（Locally Weighted Linear Regression，LWLR）：

C:\Users\xiang\AppData\Local\Temp\1539357471(1).png

所以核心代码就是这个回归参数求解过程：

|  |
| --- |
| **''' 函数说明:线性回归lwlr Returns:  testPoint \* ws - 单行预测结果 ''' def** lwlr(testPoint,xArr,yArr,k):  xMat = mat(xArr); yMat = mat(yArr).T  m = shape(xMat)[0]  weights = mat(eye((m)))   **for** j **in** range(m): *#next 2 lines create weights matrix* diffMat = testPoint - xMat[j,:]  weights[j,j] = exp(diffMat\*diffMat.T/(-2.0\*k\*\*2))  xTx = xMat.T \* (weights \* xMat)   **if** linalg.det(xTx) == 0.0:  print(**"This matrix is singular, cannot do inverse"**)  **return** ws = xTx.I \* (xMat.T \* (weights \* yMat))   **return** testPoint \* ws |

当然本次数据量比较少，用此算法可以得到比较好的结果，但lwlr也有这样的特点：

1.局部加权回归在每一次预测新样本时都会重新的确定参数，从而达到更好的预测效果。

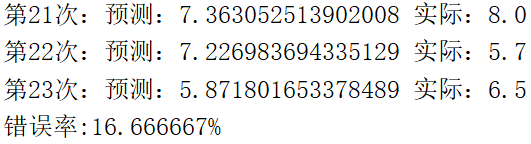
2.对于局部加权线性回归算法，每次进行预测都需要全部的训练数据（每次进行的预测得到不同的参数θ），没有固定的参数θ，所以是非参数算法。

3.当数据规模比较大的时候计算量很大，学习效率很低。并且局部加权回归也不是一定就是避免underfitting。

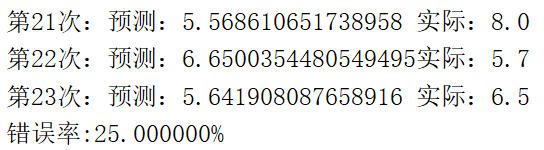
4.非参数学习算法，每进行一次预测，就需要重新学习一组 ， 是变化的，所以需要一直保留训练样本。也就是说，当训练集的容量较大时，非参数学习算法需要占用更多的存储空间，计算速度也较慢。

结果如图：

Social media的结果：取前10%为测试集，后90%训练集，k的取值为*4.8e+5*



Conventional 的结果：取前10%为测试集，后90%训练集，k的取值为*4.8e+5*



统计：

|  |  |
| --- | --- |
| Social media feature | 85.33% |
| Conventional feature | 75% |

明显正如论文所说媒体特征更加准确