一、 实验题目，

1. NB 实现分类与回归

二、 实验内容

1. 算法原理

NB 是比较适用于文本分类的一种算法。它首先假设所有特征都是相互独立的，然后把复杂的计算变简单。

朴素贝叶斯就是从联合概率公式演变出来的东西。

联合概率公式为：P(Y,X)=P(Y|X)P(X)=P(X|Y)P(Y)

P(Y|X)是 Y 的后验概率，而 P(Y)是 Y 的先验概率。在训练的时候，我们可以根据训练集的数据，计算后验概率 P(Y|X)，然后后验概率大的，则是我们要的分类。

从联合概率公式中我们就可以知道 P(Y|X)可以用 P(X|Y)P(Y)/P(X)计算得到。

朴素贝叶斯首先假设了文本的单词之间没有联系，对给定的 Y，计算每一个 x 的条件概率，相乘，然后计算得到后验概率。

朴素贝叶斯分类器是一种常见的有监督学习，常见的有两种模型，一种为多项式模型，另一种是伯努利模型。

因为朴素贝叶斯假设所有特征都是相互独立的，所以这就是一个比较大的局限性。文本之间一般都是相互关联的，我们设计算法的时候应该想法设法用上这些关联信息，但是朴素贝叶斯就干脆的丢掉了这些信息，所以结果不会好。

另外，在这次的分类算法中我们也可以明显看到它的一个缺点。当一个类别的词袋比较小的时候，经过概率连乘，这个类别极有可能会有最大的概率值。分类的话会明显偏向这一类。但是，这个类别词袋少很大一部分原因就是因为这个类别的分布小。预测的时候反而是分布最大的一类。这样一来，正确率会不如随便乱猜的好。

也就是，实际上朴素贝叶斯对样本的分布的要求是比较高的。

但是，不得不承认，就是因为相互独立的假设，所以朴素贝叶斯的实现会比较简单。这么少的数据，用上合适的方案，朴素贝叶斯也能做的比较好。还有就是朴素贝叶斯做这种多元分类比较简单。

2. 伪代码

```
多项式模型：
    计算情感c的先验概率： 类c下单词总数/整个训练样本的单词总数
    计算测试样例每个词条件概率：(类c下单词tk在各个文档中出现过的次数之和)/(类c下单词总数)
        这里如果加拉普拉斯平滑就是：(类c下单词tk在各个文档中出现过的次数之和+1)/(类c下单词总数+训练样本词数)

伯努利模型：
    计算情感c的先验概率： 类c下样例数/整个训练样本的样例总数
    计算测试样例每个词条件概率：(类c下包含单词tk的文档数)/(类c下单词总数)
        这里如果加拉普拉斯平滑就是：(类c下包含单词tk的文档数+1)/(类c下单词总数+2)


用分类算法做回归：
    把每一个训练样例按照多数投票分类
    用分类的算法计算每种情感概率

普通回归：
    每个训练样例6种情感值 * （该样例中测试样例的单词出现次数 + 1） / （该样例总词数 + 训练样例词数）
    所有训练样例得到情感值相加
    归一化


停用词：
    词语与情感相关度 = 每个训练样例的情感 * 该样例的词
    所有训练样例得到的相关度相加，得到最终相关度
    最终相关度都大于某个阈值，则为停用词
```

3. 关键代码截图（带注释）

分类：

训练

```
# 分类，进行NB计算
def trainNB_cla(dataSet, labels):

    dataSize = len(dataSet[0])
    fileSize = len(dataSet)

    pEmotion = zeros((6,1))


    pNum = zeros((6,dataSize))
    pDenom = zeros((6,1))

    for i in range(fileSize):
        pNum[int(labels[i])-1] += dataSet[i]
        pDenom[int(labels[i])-1] += sum(dataSet[i])
    # 伯努利模型
    '''
    for i in range(6):
        pEmotion[i] = (float(labels.count(str(i+1))) / float(fileSize)
    '''
    # 多项式模型
    for i in range(6):
        pEmotion[i] = (float(pDenom[i]) / float(pDenom.sum()))

    return pEmotion,pDenom,pNum
```

测试

多项式模型

```
# 多项式模型
for i in testOneMat:
    pFinalEmo = zeros((6,1))
    for k in range(6):
        pFinalEmo[k] = pEmotion[k]

    for j in range(len(i)):
            if(i[j] != 0):
                for k in range(6):
                    pFinalEmo[k] = float(pFinalEmo[k]) * (float(pNum[k][j] + 1) / float(pDenom[k] + i.sum()))

    pp = pFinalEmo.T.argsort()
    myLabel.append((pp[0][5] + 1))
print myLabel
```

伯努利模型

```
    # 伯努利模型
    index = 0
    wordInFile = []
    for item in testOneMat:
        tmp = 0
        pFinalEmo = zeros((6,1))
        for k in range(6):
            pFinalEmo[k] = pEmotion[k]
        for i in range(len(item)):
            for j in trainOneMat:
                if (item[i] == j[i] and item[i] == 1):
                    tmp += 1
        for k in range(6):
            # 出现该词文档数 + 1 / 总词数 + 2
            pFinalEmo[k] = float(tmp + 1) / float(pDenom[k] + 2)

        pp = pFinalEmo.T.argsort()
        myLabel.append((pp[0][5] + 1))
        index += 1
    print myLabel
```

回归：

用分类的模型做回归：

```
# 用分类方法做回归，进行NB计算
def trainNB_cla_reg(dataSet, labels):

    dataSize = len(dataSet[0])
    fileSize = len(dataSet)
    # 每个训练样例分类
    pp = labels.argsort() + 1
    pp = pp.T
    maLabel = []
    for i in range(len(pp[0])):
        maLabel.append(pp[0][i])
    pEmotion = zeros((6,1))

    pNum = zeros((6,dataSize))
    pDenom = zeros((6,1))
    # 情感词袋
    for i in range(fileSize):
        pNum[(maLabel[i])-1] += dataSet[i]
        pDenom[(maLabel[i])-1] += sum(dataSet[i])

    # 多项式模型
    for i in range(6):
        pEmotion[i] = (float(pDenom[i]) / float(pDenom.sum()))
    # 伯努利模型
    '''
    for i in range(6):
        pEmotion[i] = (float(maLabel.count((i+1))) / float(fileSize))
    '''

    return maLabel,pDenom,pNum
```

```
#分类算法直接用到回归

index = 0
for i in oneMat:
    pFinalEmo = zeros((6,1))
    for k in range(6):
        pFinalEmo[k] = pEmotion[k]

    for j in range(len(i)):
            if(i[j] != 0):
                for k in range(6):
                    # 词数 + 1 / 总次数 + V
                    pFinalEmo[k] = float(pFinalEmo[k]) * (float(pNum[k][j] + 1) / float(pDenom[k] + i.sum()))

    pFinalEmo = pFinalEmo / pFinalEmo.sum(axis=0)
    myLabel[index] = (pFinalEmo.T)[0]
    index += 1
print myLabel
```

```
# regression每一行训练样例得到结果相加
index = 0
for i in testOneMat:
    pFinalEmo = zeros((6,1))
    labelIndex = 0
    # 每一行测试样例都要从所有训练样例中计算情感值
    for k in trainOneMat:
        pEmotion = zeros((6,1))
        # 计算该训练样例的各个情感值
        for ll in range(len(trainLabel[labelIndex])):
            pEmotion[ll][0] = trainLabel[labelIndex][ll]
        labelIndex += 1
        for j in range(len(i)):
            if(i[j] != 0):
                for l in range(6):
                    # 词数 + 1 / 总词数 + V
                    pEmotion[l] = float(pEmotion[l]) * (float(k[j]+1)) /  (float(k.sum() + i.sum()))
        pFinalEmo += pEmotion
    # 归一化
    pFinalEmo = pFinalEmo / pFinalEmo.sum(axis=0)
    myLabel[index] = (pFinalEmo.T)[0]
    index += 1
print myLabel
```

直接做回归：

```
# 回归，进行NB计算
def trainNB_reg(dataSet, labels):

    dataSize = len(dataSet[0])
    fileSize = len(dataSet)

    pEmotion = zeros((6,1))

    pNum = zeros((6,dataSize))
    pDenom = zeros((6,1))
    # 计算每一类情感的每个词总数以及总词数
    for i in range(fileSize):
        pNum += dataSet[i]
        pDenom += sum(dataSet[i])

    # 急计算每种情绪的概率
    for i in range(6):
        pEmotion[i] = (float(pNum[i].sum()) / float(pDenom.sum()))


    return pEmotion,pDenom,pNum
```

4. 创新点&优化（如果有）

假如一个词和 6 种情感的相关度都很高，那么这个词用来做预测的话其实意义不大。这种词

大概就是 for，is 之类的介词，它们存在广泛，但是并不能依靠它们确定情感值。
所以停用掉这些词。
每个词计算和每种情感相关度，要是一个词和所有情感的相关度都大于我们设置的阈值，那么这个词就是停用词。
停用词

```
# 制作停用词
def stop_word():

    voca,length,mat1,mat3 = reg_vocabulary()

    trainMat,trainLabel,trainOneMat = TF_IDF('Dataset_train.csv',voca,length,"regression")

    rela = zeros((6,len(trainOneMat[0])))
    # 计算相关度
    for i in range(len(trainOneMat)):
        for j in range(6):
            rela[j] += trainLabel[i][j] * trainOneMat[i]

    re = rela.T
    index = 0
    stop_word = []
    for i in range(len(re)):
        # 相关度在每个情感都是比较大的，则为停用词
        if((re[i][0] > 0.3 and re[i][1] > 0.3 and re[i][2] > 0.3 and re[i][3] > 0.3 and re[i][4] > 0.3 and re[i][5] > 0.3)):
            stop_word.append(voca[i])
            index += 1
    print stop_word
    print index
    new_voca = []
    # 新的词汇表
    for i in voca:
        if(i not in stop_word):
            new_voca.append(i)
    return new_voca
```

看一下停用的是哪些词

```
['for', 'is', 'on', 'to', 's', 'new', 'in', 'iraq', 'uk', 'from', 'of']
11
```

一共停用了 11 词，这些词看起来大部分是无意义的介词。所以停用词的制作还是比较成功的，现在的问题就是，它们到底有没有用。

三、 实验结果及分析

1. 实验结果展示示例

用多项式模型，不用拉普拉斯平滑，则大部分的概率都为 0，正确率就取决于排序的方式了。
这里用的是从小到大排列，取最大，于是很大部分为 6，正确率是 0.185.

```
[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
0.185
```

用上拉普拉斯平滑，那么就有了许多的样例预测为了 2.这里有将近 900 个预测为 2，于是正确率下降至 0.037。

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 1, 1,
2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1,
2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 3, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1,
2, 1, 1, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2,
1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2,
1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
1, 2, 2, 1, 2, 2, 2, 2, 2, 2]
0.037
```

改成用伯努利模型，则是 0.026 的正确率

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
0.026
```

从上面我们可以看到，最多的预测情感是 2，也就是 disgust。Disgust 词袋最小，所以条件概率中的分母会比较小。在多次进行乘法计算之后，分母小的优势更加凸显，总的概率就大了。这样一来，就会有许多的样例被预测为了 disgust。但是，我们可以看到，这是一个分布不太均匀的样本（很大部分原因是样例数量小）。本来 disgust 应该有最小的概率，但是这里却变成了最大的。所以正确率会很低。当然，这个正确率和我们排序的方案还是有关联的。

回归：（跑了各种方案，这是详细的图，可以略看，最终有对比图，可以知道各种方案结果。）

使用前面分类用的模型，所以，首先要判断每一个样例属于哪一类情感。按照多数投票原则，发现 1-6 类情感分别有 109,44,24,57,3,9 个样例。

```
109 44 24 57 3 9
```

实际上，样例分布如此不均匀，我们不用跑程序也可以知道第五类情感会占很大的比值。看一下在验证集上结果。

1.24E-07	1.66E-06	2.00E-05	9.13E-07	0.993782	0.006195
1.36E-08	2.76E-07	1.60E-05	3.99E-07	0.995845	0.004139

结果的确是这样的，sad 的值已经是 0.999 了，所以完全偏向 sad。

```
-0.0290313260837
-0.0648790892755
0.0788746412001
-0.00931254266696
-0.0167989900846
-0.0808235422308
Out[106]: -0.020328474856910729
```

	anger	disgust	fear	joy	sad	surprise
r	-0.02903	-0.06488	0.078875	-0.00931	-0.0168	-0.08082
average	-0.02033					
evaluation	负相关 gg					

上面是 validation 的结果。第一张是用的自己写的相关系数，第二张是用 TA 给的 validation。
主要就是看一下自己的相关系数函数写对了没有。
看一眼测试集，应该也是这样的现象。

6.40E-10	6.93E-08	4.01E-06	6.67E-08	0.998958	0.001038
3.25E-11	5.74E-08	2.18E-07	7.43E-09	0.999233	0.000767

	anger	disgust	fear	joy	sad	surprise
r	-0.02903	-0.07453	-0.01835	-0.0438	-0.07955	-0.04815
average	-0.0489					
evaluation	负相关 gg					

结果是负相关的。
所以事实证明，不能把前面的分类的模型用在这个那么分布不均匀的回归数据集上。

用伯努利模型稍微好一点。

	anger	disgust	fear	joy	sad	surprise
r	0.084364	0.150676	0.056981	-0.17998	-0.02245	0.090144
average	0.029955					
evaluation	极弱相关 加油哦					

但是仍然是极弱相关。

	anger	disgust	fear	joy	sad	surprise
r	0.160534	0.089384	0.109485	-0.10118	0.055116	-0.01057
average	0.050463					
evaluation	极弱相关 加油哦					

假如把所有文本的 6 种情感除以全部情感值相加，那么可以知道这些情感的比例相近。因此，最终得到的 6 情感标签会十分相近。

```
[[ 0.00406463]
 [ 0.00406504]
 [ 0.00406463]
 [ 0.00406504]
 [ 0.00406463]
 [ 0.00406504]]
```

	anger	disgust	fear	joy	sad	surprise
r	2.51E-16	-2.4E-16	-5.3E-16	2.56E-16	-6.5E-16	-5E-16
average	-2.3E-16					
evaluation	负相关 gg					

0.166658	0.166675	0.166658	0.166675	0.166658	0.166675

所以这种方法不可行。

每个情感值乘上当前训练样例中文本的出现比例（TF），这个就要求一定要拉普拉斯平滑了。不然出来的结果基本上都会是 0.
用原始的方法，每一个训练样例情感值乘上该样例得到的条件概率（平滑后），所有训练样例相加，然后归一化。
One_hot 矩阵

	anger	disgust	fear	joy	sad	surprise
r	0.150346	0.185519	0.27196	0.241196	0.231265	0.129995
average	0.201714					
evaluation	极弱相关 加油哦					

测试

	anger	disgust	fear	joy	sad	surprise
r	0.304925	0.133157	0.270675	0.311783	0.207756	0.149954
average	0.229708					
evaluation	极弱相关 加油哦					

有重复词的计算，和 one_hot 还是相差不多

验证集：

	anger	disgust	fear	joy	sad	surprise
r	0.153274	0.17875	0.267081	0.235435	0.228805	0.126238
average	0.198264					
evaluation	极弱相关 加油哦					

TF 矩阵也差不多

验证集：

	anger	disgust	fear	joy	sad	surprise
r	0.134316	0.165175	0.239363	0.208738	0.20764	0.117904
average	0.178856					
evaluation	极弱相关 加油哦					

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

(原始算法结果)

准确率在下面和优化后的一起对比

|----------如有优化，请重复 1，2，分析优化后的算法结果----------|

加入停用词

停用词的条件严格一点点

验证集：

	anger	disgust	fear	joy	sad	surprise
r	0.135297	0.209016	0.326668	0.267473	0.221247	0.051056
average	0.201793					
evaluation	极弱相关 加油哦					

停用词如下

```
['for', 'is', 'on', 'to', 's', 'new', 'in', 'iraq', 'uk', 'from', 'of']
11
```

这样的停用词只有 11 个，个数还是比较少的。毕竟这里一共有三千多个词。所以稍微把条件放松一点，看看结果。这样就有了 14 个停用词，a 和 by 明显也是应该停用的词。所以这次的结果应该好些。

```
['for', 'is', 'a', 'on', 'to', 's', 'new', 'in', 'iraq', 'uk', 'from', 'of', 'by', 'study']
14
```

	anger	disgust	fear	joy	sad	surprise
r	0.195345	0.270583	0.363575	0.250752	0.226923	0.185079
average	0.24871					
evaluation	极弱相关 加油哦					

结果提高了比较多，虽然不是很棒，但是已经令人感动了。
我迫不及待想要跑下测试集了。

	anger	disgust	fear	joy	sad	surprise
r	0.366555	0.151189	0.300831	0.33167	0.233907	0.220869
average	0.267503					
evaluation	极弱相关 加油哦					

再来一个，15 个停用词。

```
['for', 'is', 'a', 'on', 'to', 's', 'new', 'in', 'iraq', 'uk', 'from', 'of', 'by', 'study', 'man']
15
```

在验证集上：

	anger	disgust	fear	joy	sad	surprise
r	0.196887	0.260294	0.364003	0.248128	0.227145	0.185241
average	0.24695					
evaluation	极弱相关 加油哦					

验证集结果和前面差不多，看看测试集。

	anger	disgust	fear	joy	sad	surprise
r	0.368625	0.151396	0.301227	0.334222	0.233562	0.220858
average	0.268315					
evaluation	极弱相关 加油哦					

好的，结果也是将近 0.27.
停用词的要求放松，个数多一点，结果会好挺多啊。再试试大一点的停用词集。停用词是 18 个。

```
['for', 'is', 'a', 'on', 'to', 's', 'new', 'in', 'iraq', 'us', 'photographer', 'uk', 'from', 'of', 'by', 'study', 'man', 'u']
18
```

但是 photographer 和 study 这种应该并不是噪音的词也被划进了停用词集，这个停用词再用同样方法扩充就很容易起反作用了。

	anger	disgust	fear	joy	sad	surprise
r	0.128335	0.260504	0.360801	0.23822	0.229953	0.170454
average	0.231378					
evaluation	极弱相关 加油哦					

果然如此，差了一丢丢。
这次的停用词是 19 个。把 and 加进来，感觉应该还是不错的。比上面好一点。（好想人工

把 photographer 和 study 从停用词集拉出去啊，可以这样就很人工不智能了。看着它们破坏我的相关系数）

```
['for', 'is', 'a', 'on', 'to', 's', 'new', 'in', 'and', 'iraq', 'us', 'photographer', 'uk', 'from', 'of', 'by', 'study', 'man', 'u']
19
```

	anger	disgust	fear	joy	sad	surprise
r	0.127699	0.264174	0.365434	0.236654	0.228831	0.18115
average	0.23399					
evaluation	极弱相关 加油哦					

似乎要差一点了
可以看到，停用词还是有用的。

对比

用多项式分类模型	-0.02033	-0.0489
用伯努利分类模型	0.029955	0.050463
训练集6种情感分别的均值	gg	
普通回归方法（one_hot）	0.201714	0.229708
普通回归方法（有重复词）	0.198264	
普通回归方法（TF）	0.178856	
停用词（11个）	0.201793	
停用词（14个）	0.24871	0.267503
停用词（15个）	0.24695	0.268315
停用词（18个）	0.231378	
停用词（19个）	0.23399	

从这里，我们可以看到，使用了停用词，正确率是有提高的。但是太多停用词，又会造成相关系数下降。因为过多停用词就会把一些有用词也划进了停用词集里面。
用了 14 个停用词有最大验证集相关系数，0.24871。但是用了 15 个停用词，在验证集的相关系数会稍稍下降一点点，但是在测试集上跑出了 0.268315 的相关系数，比前面好一点点。
总的来说，用上合适的停用词，相关系数会得到提高。但是再怎么提高，也不会太高，毕竟这个训练集真的小。