中山大学数据科学与计算机学院

移动信息工程专业-人工智能

本科生实验报告

（2017-2018 学年秋季学期）

课程名称：Artificial Intelligence

教学班级	1506	专业（方向）	移动信息工程（互联网）
学号	15352116	姓名	洪子淇
联系电话	13726205766	邮箱	1102229410@qq.com
开始日期	2017/10/8	完成日期	2017/10/16

一、 实验题目

1. 理解 K 近邻算法（KNN），使用 KNN 算法处理文本情感分类以及回归问题，适当对其进行优化；
2. 理解朴素贝叶斯算法（NB），使用 NB 算法处理文本情感的分类以及回归问题；
3. 回答相关的思考题内容。

二、 实验内容

A. 文本分析

一共分为三个数据类型：**训练集、验证集、测试集**。

训练集是用来确定模型的参数的，例如平时的学习积累；**验证集**是用来确定网络结构或者控制模型复杂程度的参数的，简单来说就是修正训练集得到的模型。而**测试集**就是检验最终选择最优的模型的性能。

B. 算法分析

1. KNN

KNN 算法通过距离函数判定测试实例与训练集实例中最靠近的 K 个训练集实例，如果是分类问题，就对 K 个训练集实例进行**类属划分**，找到最靠近的类属，就是测试实例所预测的类属；如果是回归问题，就是对于 K 个测试实例的**贡献进行加权**，越近的距离赋予越大的权值，通过 K 个实例的总体贡献来得到测试实例的贡献值。KNN 算法的流程和思路非常简单，关键在于两个文本之间的距离计算：

首先是对文本转换为单词向量，可以利用曼哈顿距离、欧式距离、余弦相似度等距离计算两个文本之间的距离。

✧ **KNN 分类算法**

1) **算法描述：**

验证集的每一个样本和整个训练集对比找到前 K 个距离最近的训练集样本，距离函数是两个样本之间曼哈顿距离。找到前 K 个之后计算得到出现次数最多的

情感类属，将其作为验证集样本的预测情感。同理得到测试集样本的预测情感。

2) **数据结构设计（python）：**

a) 情感列表和单词总列表都使用一个元组（tuple）存储。

b) 每一个文本集合使用列表（**list**）存储，文本集合下的每一个样本使用字典（**dict**）存储，每个字典根据需要会设有“**emot_id**”，“**emot_predict_id**”以及“**word_id**”三个 key 值。“emot_id”和“emot_predict_id”对应了情感元组的位置，“word_id”对应了单词总表的位置。

以验证集的数据结构为例（这里截取了第一个样本）：
[{'emot_id': 3, 'emot_predict_id': 5, 'word_id': [1289, 3179, 141, 3112, 2041, 2126, 144, 2901]},.....] 对应训练集的每个样本中没有'emot_predict_id'，测试集的样本中没有'emot_id'。

3) **算法实现：**

① 遍历训练集、验证集、测试集，得到情感列表以及所有的单词列表（这里的单词包括三个文本集合不重复的单词）。

② （getSet）再次遍历训练集、验证集以及测试集，将 train_set（validation_set/test_set）转化为上述的数据结构模型。

③ （calc_distance）距离函数：通过传入两个样本之间的单词的位置下标，利用 set 函数找到之间的交集，距离公式：**曼哈顿距离，调和平均数等**。（由于数据结构的设计，只能将文本模拟成 one-hot 矩阵）

④ （find_kNN）将所要预测的样本和训练集的每个样本进行比较，得到 N 个距离（N：训练集样本个数），排序得到前 K 个距离以及对应训练集的样本坐标。返回的是一个列表包含有 K 个元组（按距离从小到大），每个元组的第一个数是该样本在训练集的位置，第二个数是训练集样本和测试样本之间的距离。

⑤ （emot_Vector）将情感 id 处理成数组形式，如果在 0 位置数组就是(1,0,0,0,0,0)，如果是在 1 的位置数组就是（0,1,0,0,0,0）

⑥ （getPredict_id）将 K 个训练样本进行分类，找到出现次数最多的情感，并将其设置为测试样本的预测情感。

⑦ （getResult）按照要求输出测试集的测试情感结果。

✧ **KNN 回归算法**

1) **算法描述：**

KNN 回归算法，与 KNN 分类算法大致相同。不同在于每个文本得到的不是情感的类属，而是所有情感的归属概率。通过由前 K 个样本的情感归属概率加权求和，之后归一化得到测试样本的预测情感归属概率，权值取测试样本与训练样本之间距离的倒数。

2) **数据结构设计**

a) 与 KNN 分类类似，单词总表和情感表都设置为元组类型

b) 由于回归问题中，每一个训练样本的中的是情感的概率参数，类似分类的结构，只是将‘emot_Predict_id’改为‘emot_Predict_Pr’并且里面存储着所有情感的概率（对应情感列表的顺序）。

3）算法实现

①～④同 KNN 分类算法

⑤（calc_emotionP）传入测试样本，训练样本以及 find_knn 的结果 knn_res，利用 map 依次获取测试样本与训练样本之间的距离，并将其转化为数组类型用于计算，各自取倒归一化之后得到 1*K 的距离向量，然后和元素为 1 的 l_emot*1 的向量进行克罗克内积得到 l_emot*K 的距离矩阵；通过 map 得到前 K 个情感归属概率的矩阵（l_emot*K），将两个矩阵进行相乘按列求和便可得到 1*l_emot 的情感归属概率，再次对此进行归一化，减小浮点误差。

⑥（getResult）按照要求输出测试集的测试情感结果。

2. NB

✧ NB 分类算法

1） 算法描述

朴素贝叶斯算法是基于贝叶斯定理与一个简单的假定：给定目标值时，属性之间相互条件独立。贝叶斯定理的简单形式如下：

$$\mathrm{P(y|x)} = \frac{P(x|y)P(y)}{P(x)}$$

分母是一个常数，所以朴素贝叶斯的目的是找到最大后验概率：

$$\mathrm{y} = \arg\max_y P(y|x) = arg\max_y \frac{P(x,y)}{P(x)} = arg\max_y P(x|y)P(y)$$

对于此次的文本分析，x 是一个文本的特征向量，y 是情感类别，根据朴素斯的假定特征向量之间相互条件独立，那么公式可以表达为：

$$\mathrm{y} = \arg\max_y \mathrm{P}(y = c_k|x) = \max_y \prod_{i=1}^{M} P(x_i|y = c_k)P(y = c_k)$$

参数说明：$P(y = c_k)$表示每个类别发生的概率；$P(x_i|y = c_k)$表示特征向量在各个类别发生的概率。 所以朴素贝叶斯的学习是对这两个概率的估计，故可以采用极大似然估计上述的先验概率和条件概率。

先验概率$P(y = c_k)$的极大似然估计是：

$$P(y = c_k) = \frac{\sum_{i=1}^{N} I\left(y_i = c_k\right)}{N}$$

条件概率$P(x_i|y = c_k)$的极大似然估计是：

$$P(x_i|y = c_k) = \frac{\sum_{i=1}^{N} I\left(\sum x_i ,\ y_i = c_k\right)}{\sum_{i=1}^{N} I\left(y_i = c_k,\ \sum xj\right)}$$

其中 xj 表示样本下的单词，$I（y_i = c_k，\sum xj）$表示 c_k 类下的单词总数。

2） **数据结构设计**

文本数据结构参考 KNN 分类的文本数据结构设计，每个样本作为一个字典，key 值分别是‘emot’（选），‘words’，‘pred_emot’（选），与 KNN 不同，这里的 value 值存储的是单词，而不再是下标位置。

3） **算法实现**

① （getSet）得到训练集和验证集的单词列表，重复出现只算一次。以及训练集和验证集的数据集合。

② 对训练集合根据情感类别进行分组，对每一组下的所有样本的单词进行加和，得到每个类别下的单词出现的次数。并得到每种类别下的单词数目（包括重复的和不重复的）。

③ 根据上述的公式和已经处理好的数据计算测试样本中每个情感出现的概率，对于概率为 0 的单词特征使用拉普拉斯平滑。找到最大的概率，便是测试样本对应预测的情感。

④ （getResult）输出文件

✧ **NB 回归算法**

1） **算法描述**

NB 回归算法其实是一种最大后验概率（MAP）与贝叶斯定理的结合。融合了预先估量的先验分布信息，所以最大后验估计可以看作是正则化的最大似然估计。贝叶斯公式如下：

$$\arg\max_y \mathrm{P}(y = c_k|x) = \arg\max_y \sum_{j=1}^{N} \prod_{i=1}^{M} P(x_i|y = c_k，d_j)P(y = c_k，d_j)$$

根据最大后验估计，最终公式可以转化为：

$$\mathrm{P}(y = c_k|x) \propto \sum_{j=1}^{N} \prod_{i=1}^{M} \frac{I（x_i，dj）}{I（\sum xj，dj）} P(y = c_k，d_j)$$

其中，dj 表示每一个测试样本，$I（\sum xj，dj）$表示 dj 下的单词总数

2） **数据结构设计**

数据结构与 NB 分类大致一样，每个文本字典的 pred_eomt 存储情感归属概率列表，以及多了‘count’的 key 值，用来存储每个单词出现的个数。

3） **算法实现**

① 同 NB 分类

② （getPredit_emot）通过循环遍历，每种情感下每一个训练样本下的情感归属概率，求和得到测试样本的该种情感归属概率，得到所有的情感归属概率之后进行归一化。

③ （getResult）输出文件

✧ **拉普拉斯平滑**

测试样本中如果出现了训练集中从来没有出现过的特征词，根据上述公式计算会导

致概率为 0，这是不合理的，一方面考虑到没有在训练集出现的单词并不意味着它就不是这一类别下的特征词，另一方面概率的连乘会导致原本类属那个情感下的概率一下子变为 0，这也是不合理的，不能由于一个没有出现的特征词就否定了前面特征词的概率。

拉普拉斯平滑是为了解决测试样本中出现了训练集中没有出现过的特征词而导致在计算的过程中出现了 0 概率而引入的。

首先从简单的想法入手，给所有没有出现的单词都赋予 1 的出现次数，当训练集很大的时候，未出现的单词数目的情况也较小，所以对于这种变化可以忽略不计。当由于本实验的训练集比较小，在测试集中未出现过的单词的数目也比较多，所以影响较大，这时候我们就需要对公式进行修改。如果是用同一个公式计算，那么在分子加上 1 之后，不仅会对未出现的单词赋予出现 1 的次数，也会在原本出现的单词的次数上增加 1，为了使经过拉普拉斯平滑后的概率加和为 1，得到以下公式（此处不分类别）：

$$P_i = \frac{x_i + 1}{N + M}$$

其中，x_i 表示特征词出现的个数，N 表示总的特征词（包含重复出现的特征词），M 表示总的不重复的特征词。由于对没有出现的特征词赋予 1 的出现次数，会对本来出现次数比较少的特征词不太公平，所以为了降低新的特此词出现的影响程度，将 1 改写成α，取值一般小于 1，用于降低训练集中未被观察到的特征词的影响。

$$P_i = \frac{x_i + \alpha}{N + \alpha M}$$

C. 伪代码

1) KNN 关键变量

```
voc_tuple：单词表
emot_tuple：情感表
train_set , validation_set, test_set：训练集、验证集、测试集
knn_res ：KNN 算法得到的结果[（i，dist）]
```

KNN 关键函数：

```
def getSet(file):
    my_set = [] #初始化
    open the file and readline #打开文件，一行一行地读取文件
    for line in file:
        if not test file:
            item[‘emot_id’] = emotion   #一行中的情感
        if not train file:
            initialize   item[‘emot_predict_id’] #将预测情感初始化
        item[‘word_id’] = the index of words in voc #一行中的单词
    close the file #关闭文件
```

```
def calc_distance(a, b):
    c = a&b
    dist1 = len(a) + len(b) – 2*len(c) #曼哈顿距离公式
    dist2 = ( len(c)*( len(a)+len(b) ) / (2*len(a)*len(b)) )^4 #调和平均数
    dist2 = 1/（epsilon + dist2）
    return dist1 or dist2 # 返回曼哈顿距离或者利用调和平均数得到的距离
```

```
def find_knn（test_item，train_set，k）：
    for I in range（N）：
        dist = calc_distance（test_item[‘word_id’],train_set[‘word_id’]）
        distance += [(I, dist)]   #存储下标和距离
    sort the distance according the dist   #根据距离排序
    return distance
```

```
def getPredict_id(test_set, train_set)：#对样本进行预测
    for item in test_set:  #在测试集合的每一个样本
        knn_res = find_KNN(item, train_set, k)
        item[‘emot_predict_id’] = most emotion in knn_res
```

2) **KNN 分类**

```
main ：
#get the vocationary and the emotional set
voc = set([])
emot = set([])
for line in the three doc_file:
    emot |= set([emotion in line]) #利用 set 得到不重复的情感列表
    voc |= set([words in line])  #利用 set 得到不重复的单词总表
train_set = getSet(train_path) #得到训练集
test_set = getSet(test_path)
getPredict_id(test_set, train_set) #得到测试集的预测情感
getResult(test_set)  #输出测试集的预测情感
```

3) **KNN 回归**

```
train_set = getSet(train_path) #得到训练集
test_set = getSet(test_path)
getPredict_id(test_set, train_set) #得到测试集的预测情感
getResult(test_set)  #输出测试集的预测情感
```

4) **NB 关键变量**

```
train_voc  训练集不重复的单词
validation_voc  验证集不重复的单词
train_set , validation_set, test_set：训练集、验证集、测试集
----------------------NB 分类-------------------
N  样本总数
emot_num  每种情感的样本数的集合
words_emot  每种情感下的单词出现的次数
words_scnt_emot  每种情感下单词的总数（不重复的）
words_cnt_emot  每种情感下单词的总数（重复的）
```

5）**NB 分类**

```
train_set = getSet(train_path) #得到训练集
test_set = getSet(test_path)
train_set.group by in emotion #将训练集按照情感进行分组
words_emot = words in the group #将每一组的单词加起来
#得到每一类下不重复的单词数目
```

```
words_scnt_emot = length（words_emot，no-repeated）
#得到每一类下包含重复的单词数目
words_cnt_emot = length（words_emot，repeated）
for ei in emotion：
    for test in test_set：
        prob[ei] = words_emot[ei][test_words] / words_cnt_emot[ei] #似然度
        p_res[ei] = prob*emot_num[ei] / N #先验概率
test[‘pred_emot’] = max{p_res[ei]} #将最大的概率所属类别作为预测结果
```

5) **NB 回归**

```
train_set = getSet(train_path) #得到训练集
test_set = getSet(test_path) #得到测试集
for test in test_set：  #对于测试集的每个测试样本
    for e in emotion：
        for i_train in train_set：
            Len_words = len(i_train[‘words’]) #得到每个训练样本的单词数目
            for wd in test：      #对于测试样本下的每个单词
                P *= count(train[wd]) / Len_words
                P_res += P * I _train[e] #后验概率
    normalize the P_res
```

D. 关键代码截图

1) 文本处理&输出文件

由于四个算法的文本处理和输出文件大致一样，这里只列出了 KNN 分类的文本处理及输出文件函数：

FUNCTION getSet：

```python
voc_tuple = tuple(voc)   #训练集、验证集、测试集出现的单词表（不重复）
emot_tuple = tuple(emot) #情感种类表
# 读取训练集，验证集，测试集并初始化
def getSet(fpath, target=False, predict=False):
    my_set = []
    fileI = open(fpath)
    fileI.readline() #去掉第一行
    for line in fileI: #第二行开始一行一行处理
        row = line.strip('\n').split(',')
        item = {}
        if target:
            item['emot_id'] = emot_tuple.index(row[1])
        if predict:
            item['emot_predict_id'] = -1
        item['word_id'] = []
        #如果在单词表里面就映射到word_id下 ，并且对每一行的单词进行除重
        for w in (set(row[0 if target else 1].split()) & voc):
            item['word_id'] += [voc_tuple.index(w)]
        my_set += [item]
    fileI.close()
    return my_set
```

FUNCTION getResult：

```
# 输出文件
def getResult(filestring,my_set):
    fileO = open(filestring,'w',newline='')
    writer = csv.writer(fileO)
    writer.writerow(['textid','label'])
    for i in range(len(my_set)):
        itemres = [str(i+1), emot_tuple[my_set[i]['emot_predict_id']]]
        writer.writerow(itemres)
    fileO.close()
```

2) KNN 分类算法

FUNCTION calc_distance:

```
def calc_distance(a, b):
    # 曼哈顿距离 & 欧式距离下降准确率
    c = [x for x in a if x in b]
    dist = len(a) + len(b) - 2 * len(c) # 曼哈顿
    return dist
```

FUNCTION getPredict_id:

```
def getPredict_id(test_set, train_set, validation=False):
    error = 0
    for item in test_set:
        knn_res = find_kNN(item, train_set, k)#前K个在训练集的下标以及与测试样例之间的距离
        emot_res = np.zeros(len(emot_tuple)) #初始化结果(0,0,0,0,0,0,0)
        for i in range(k):
            #将前K个情感转换为数组进行相加
            emot_res += emot_Vector(train_set[knn_res[i][0]]['emot_id'])
        pred_id = list(emot_res).index(max(list(emot_res))) # 找到最大值的下标,即为预测情感下标
        item['emot_predict_id'] = pred_id #更新测试文本中的预测情感
        if validation :
            if item['emot_predict_id'] != item['emot_id']: #在验证集中,如果预测情感与标准情感不同则error+1
                error += 1
    return [test_set, error]
```

3) KNN 回归算法

FUNCTION calc_emotionP:

```
def calc_emotionP(test_item, train_set, knn_res):
    # 计算情感概率
    k = len(knn_res)
    l_emot = len(emot)
    dist = list(map(lambda x: x[1], knn_res))
    dist = np.array(dist)
    dist_inv = 1 / (epsilon + dist) # 分母非零
    dist_inv /= sum(dist_inv) # 归一化
    # dist_inv = (dist_inv - min(dist_inv)) / (max(dist_inv) - min(dist_inv)) #标准化
    emot_Pr_matrix = []
    for i in range(l_emot):
        emot_Pr_matrix += [list(map(lambda x: train_set[x[0]]['emot_Pr'][i], knn_res))]
    emot_Pr_matrix = np.array(emot_Pr_matrix)
    #进行克罗克内积，将矩阵扩展后再相乘
    emot_Pr_matrix *= np.kron(dist_inv, np.ones([l_emot, 1]))
    res = np.sum(emot_Pr_matrix, axis=1) #按列求和
    res /= sum(res) # 再次归一化，减小浮点误差
    return list(res)

for item in test_set:
    knn_res = find_kNN(item, train_set, k)
    item['emot_predict_Pr'] = calc_emotionP(item, train_set, knn_res)
```

4) NB 分类算法

根据算法分析，求出所需要的各个参数：

```
N = len(train_set)  # 样本总数
emot_num = collections.Counter(list(map(lambda x: x['emot'], train_set))) # 每种情感的样本数的集合
emotion = tuple(sorted(emot_num.keys()))    # 情感列表，用于索引
denominator = N+len(emotion)
pei = list(map(lambda x: (emot_num[x])/denominator, emotion))   # 每种情感占样本总数的概率的集合

df = pd.DataFrame(train_set)
words_emot = list(df['words'].groupby(df['emot'])) #根据情感对train_set进行分组
words_emot = list(map(lambda w: reduce(lambda x,y:x+y, list(w[1])), words_emot)) #整合同类别的单词
words_emot = list(map(lambda x: collections.Counter(x), words_emot)) #记录了每个单词出现的频数
words_scnt_emot = list(map(lambda x: len(x), words_emot)) #每种情感下单词的总数（不重复的）
words_cnt_emot = list(map(lambda x: sum(x.values()), words_emot)) #每种情感下单词出现的总数（重复的）
```

计算后验概率，将似然度函数转化为对数求和方式进行计算

```
lambda2 = 0.5
def f(x, e):
    #拉普拉斯平滑:
    return math.log((words_emot[e][x]+lambda2)/(words_cnt_emot[e]+sigma*lambda2))
    # return (words_emot[e][x]+lambda2)/(words_cnt_emot[e]+sigma*lambda2)
def getPredit_emot(test_set, train_set, validation=False):
    error = 0
    for item in test_set:
        words = tuple(item['words'])
        emot_id = range(len(emotion)) #索引的情感id
        # prob = list(map(lambda e: reduce(lambda x,y: x*y, [f(w,e) for w in words]),emot_id))
        #将公式转化为对数求和的方式计算
        prob = list(map(lambda e:math.exp(reduce(lambda x,y:x+y,[f(w,e) for w in words])),emot_id))
        P_res = tuple(np.array(pei) * np.array(prob))
        item['pred_emot'] = emotion[P_res.index(max(P_res))] #取得最大概率下的情感类别作为预测结果
        if validation:
            if item['pred_emot'] != item['emot']:
                error += 1
    return [test_set, error]
```

5) NB 回归算法

```
def f(wd, train):
    num = (train['count'][wd] + lambda1)
    denom = (len(train['words']) + lambda1*Len_voc)
    return num / denom

def getPredit_emot(test_set,train_set):
    for item in test_set:
        prob = []
        for e in range(len(emotion)):
            sigma = 0
            for i_train in train_set: #在某一类别下遍历训练集，根据公式计算测试样本在这一类别下发生的概率
                total_words = len(i_train['words'])
                p = [f(wd, i_train) for wd in item['words']]
                sigma += i_train['emot'][e]*reduce(lambda a,b:a*b, p)
            prob += [sigma]
        item['pred_emot'] = list(np.array(prob) / sum(prob)) #归一化
    return
```

6) **拉普拉斯平滑**

分类和回归在平滑过程中相似，都在加入了验证集未出现的单词的数目

```
lambda2 = 0.5
def f(x, e):
    #拉普拉斯平滑:
    return math.log((words_emot[e][x]+lambda2)/(words_cnt_emot[e]+sigma*lambda2))
```

E. 创新点&优化

a) 删去部分无意义的虚词：

分析文本数据可以看出来有一些意义的单词对于情感的分类没有什么作用，比如一些虚词，以及所有格，所以我在四个实验中都堆单词进行了部分的预处理，将部分虚词排除到单词表外。（分析情感与单词之间的联系的时候，不仅仅是单个单词的出现，还

得考虑单词的关联度与情感的维系以及日常单词出现在某情感之间的概率，所以虚词表的构造只能对关键字的提取起到了一定的优化，不过不是很大。）

举个例子来说，在训练文本中由于格式问题，把所有格的分号变成了单个 s，或 re 的存在，而在词义上所有格的含义其实不是很大（标绿的可能是所有格）：

```
dr mcdreamy choked in grey s scuffle,sad
gunmen abduct filipino woman in nigeria,fear
escape to prague without the summer hordes,joy
confusion reigns in the expanding digital world,surprise
alaska s northern lights captivate japanese,joy
riot warning for france suburbs,sad
hussein s niece pleads for father s life,sad
nadal wants australian open crown,joy
blog here come the globes,joy
dispute over iraqi cleric said to have gone to iran,anger
thai govt moves to smooth over relations with south,joy
rwanda mulls three child limit,surprise
beware of peanut butter pathogens,fear
celebrities protest malibu gas facility,joy
freed muslim terror suspect says britain is police state,fear
happy birthday ipod,joy
advantage cardinals with suppan in game,joy
vegetables may boost brain power in older adults,surprise
matt shines with his alpine finish,joy
japan s economy grows,joy
believing scripture but playing by science s rules,surprise
pig cells hope for diabetes cure,surprise
vaccines mandate or choice,surprise
tbs to pay m fine for ad campaign bomb scare,fear
bombings say muslim rebels won t negotiate,fear
u s diverts troops to fight taliban,anger
livingstone s transport empire may extend,joy
```

整理部分虚词库如下：

```
above to across after against along amony another around as at of before
behind below beneath besides between by during for in into near on onto
or other per under until up upon via while with within after all and
before if a that now when whenever where whether yet else every he she
hers herself him himself his I it its me mine my myself our ours
ourselves some something that their themselves these they this those us
we you your yours yourself yourselves can have will would could b c d e f
g h i j k l m n o p q r s t u v w x y z
```

b) 结构优化与距离优化（KNN）：

对于 KNN 进行了数据结构的优化之后会导致 tf 矩阵以及 tf_idf 矩阵难以实现。从本质看，tf 矩阵是考虑到了词频出现的概率以便更好地对关键字进行提取，更加准确地计算两个样本之间的距离。这是曼哈顿距离的缺陷，所以采用了调和平均数乘上两个样本之间的交集以便更好地提取关键字。

c）选取不同的 K 值（KNN）：

K 值选择为 1，就会类属到最接近的训练样本。当 K 为训练样本数时，那么测试样

本的类属就会一直是训练样本中数目最多的类别。所以选取不同的 K 值对于机器学习很重要。根据这次实验，K 值一般是 20 左右最好，即大概是样本数目的 1/2 次方。

d）对前 K 个训练样本的权重标准化（KNN 回归）

简单地取距离的倒数作为权重的话，每个权重的对比度不够高，对权重进行标准化（dist-min）/（max-min），能够更好地放映每一个样本之间的对比度，更能放映不同样本之间的距离差距，放大训练样本的影响系数。

e）对于拉普拉斯平滑公式分母加权的设置（NB）

由于每个单词下对应情感的计算都是基于一条相同的公式，这就会导致在分子加上 1 进行平滑之后，也会在已经出现的单词的次数增加 1，所以对于分母也需要相对应地加上训练集不重复的单词的个数。这里优化的是还在分母上加上了测试集未出现的单词的个数，和之前 KNN 的单词表加上未出现的单词一样，把这一部分的单词也考虑到单词总表进去。

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

（结果展示用的是 KNN 验收的小数据集，回归的情感概率是主观依据。）

分类的训练集：

Words	label
hello introduce these are TAs	joy
hello this is a test	joy
some TAs have no girlfriend	sad
some TAs have girlfriend	joy
TAs live happy	joy
hello you all have no girfriend	sad

回归的训练集：

Words	anger	disgust	fear	joy	sad	surprise
hello introduce these are TAs	0	0	0	0.4	0	0.6
hello this is a test	0	0.4	0.1	0.1	0.4	0
some TAs have no girlfriend	0	0	0	0.1	0.8	0.1
some TAs have girlfriend	0	0	0	0.8	0.1	0.1
TAs live happy	0	0	0	1	0	0
hello you all have no girfriend	0	0	0	0	1	0

测试集：

id	Words	label
1	some of you have no girlfirend	

[KNN:基于 onehot 矩阵，使用了曼哈顿距离，其中取 K=3]

KNN 分类结果展示：

textid	label
1	sad

KNN 回归结果展示：

textid	anger	disgust	fear	joy	sad	surprise
1	0	0	0	0.418182	0.481818	0.1

NB 分类回归展示(采取多项式，简单的拉普拉斯平滑)：

textid	label
1	sad

NB 回归展示：

textid	anger	disgust	fear	joy	sad	surprise
1	0	0.049867	0.012467	0.378551	0.443643	0.115472

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）

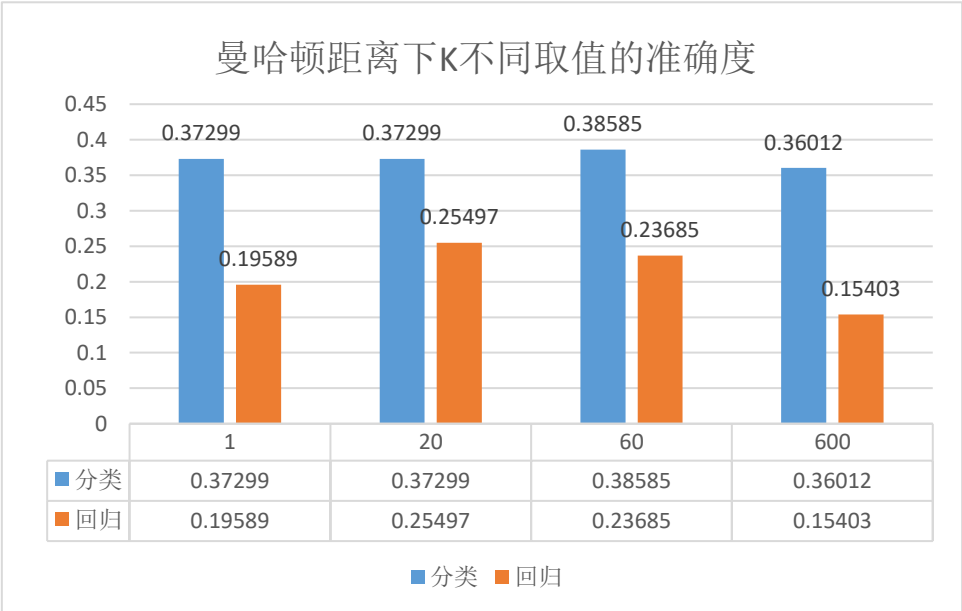
（使用的是实验的大数据集，通过使用验证集的分类准确率和回归相关系数作为评测指标）

1.未优化的 KNN

对于没有优化的 KNN，K 取 1，分类：表示归于最近的训练样本的类属；回归：表示和最近的训练样本具有相同的情感归属概率。两个样本之间的距离采取曼哈顿距离表示，得到的结果与正确结果比较：分类的准确率是 0.37299，回归的相关系数为 0.19589

2.优化的 KNN

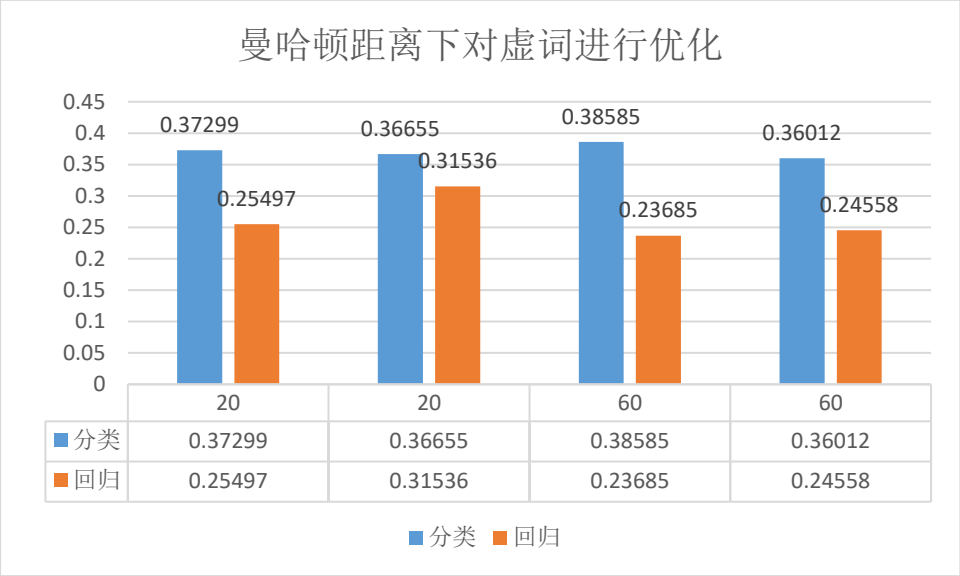
① 选取不同的 K 值



分析可得在 K = 60 的时候分类的准确率较高，K=20 的时候，回归的相关系数比较高

② **删除部分虚词**

取 K 为定值，在词库删去部分虚词，忽略虚词产生的距离影响



【单数表示没有去掉虚词的测评结果；双数表示去掉部分虚词后的测评结果】

观测可知，K＝20 的时候，去掉部分虚词分类的准确率会稍稍下降，不过在回归问题中，相关系数会有所提高，不过结合理论分析，除掉部分虚词对于准确度的提高不是特别明显，由于现实生活中往往词的类属虽然和虚词没有很大的关系，主要是和经验词频出现，以及词组联系有关，没有一个强大的词库，得到特征词之间的相关度，以及特征词类属的似然权重，就难以在准确度有很大的突破。

③ **采用不同的距离公式**

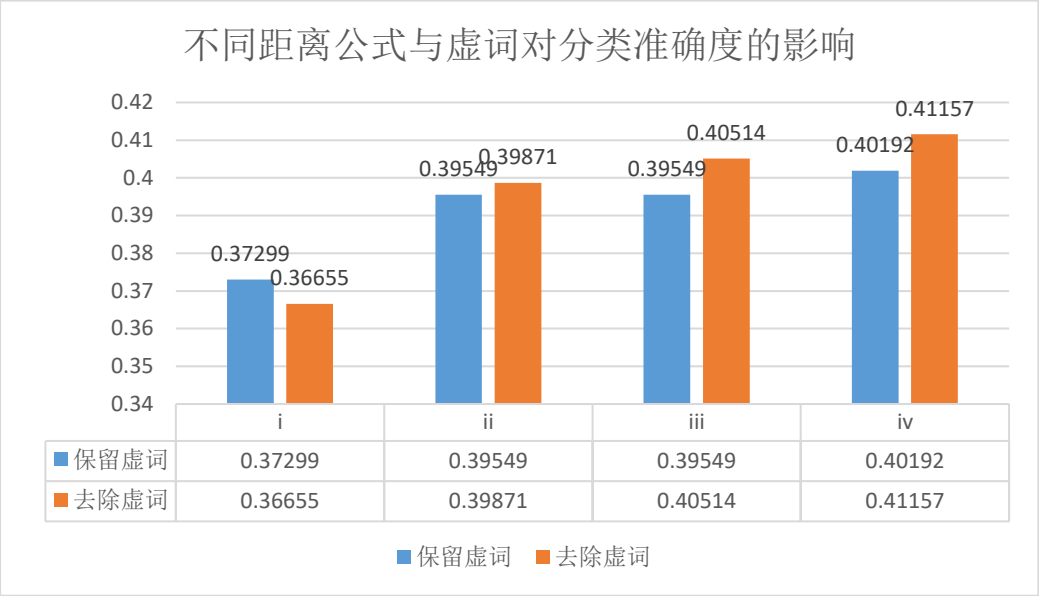
这里采用了 K=20，分别根据以下距离公式依次计算两个样本之间的距离，除了曼哈顿距离之外其余都是出于对两个样本之间交集的单词的权重比较大的考虑（a，b 表示两个样本，c 表示两个文本之间的交集），所以返回的是该距离公式得到的倒数：

i. len(a) + len(b) - 2 * len(set(a) & set (b))　　曼哈顿距离
ii. (2 * len(c) / (len(a) + len(b)))
iii. len(c) ** 4
iv. (len(c)*(len(a)+len(b)) / (2*len(a)*len(b)))**4　　调和平均数

这里分为两个部分：一部分没有去掉虚词以及去掉虚词的测评。

结果预测：当利用曼哈顿距离的时候虚词作为交集的主要部分可能会把很多具有多个虚词的句子拉近距离，导致误差的产生。所以在曼哈顿距离下，去掉虚词可能会对准确度有所提高，也会有所下降，都是在可理解的范围之内。如果从交集的特征词的权重的角度考虑，由于虚词是在句子不做情感部分的构成，删除虚词反而更能精确地找到关键的特征词，从而进一步优化了 KNN 算法。

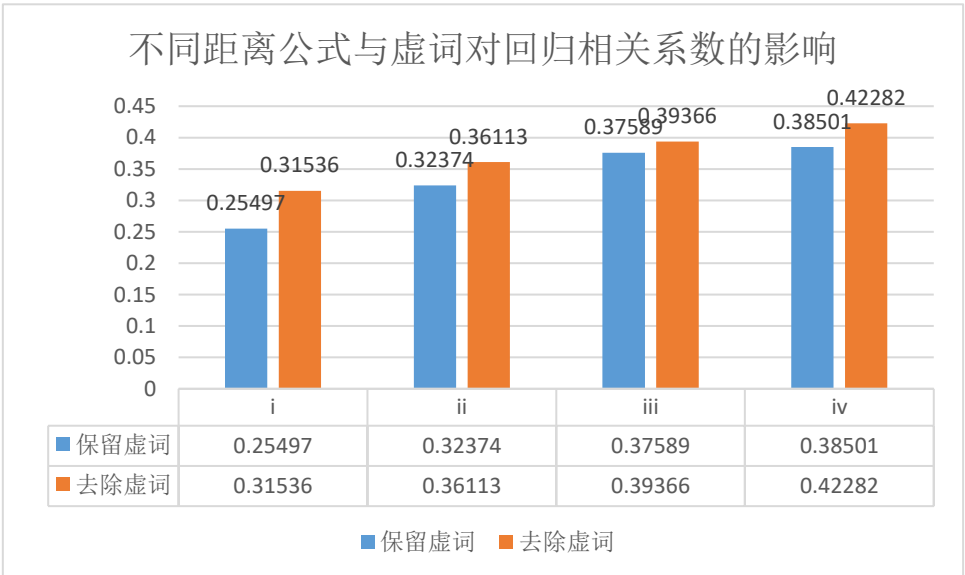
得到实际测评结果如下：



从结果预测可知：

在考虑关键的特征词的层面上，第四个距离公式，即是利用了调和平均数来优化对关键特征词的加权较有效地提高了准确度。

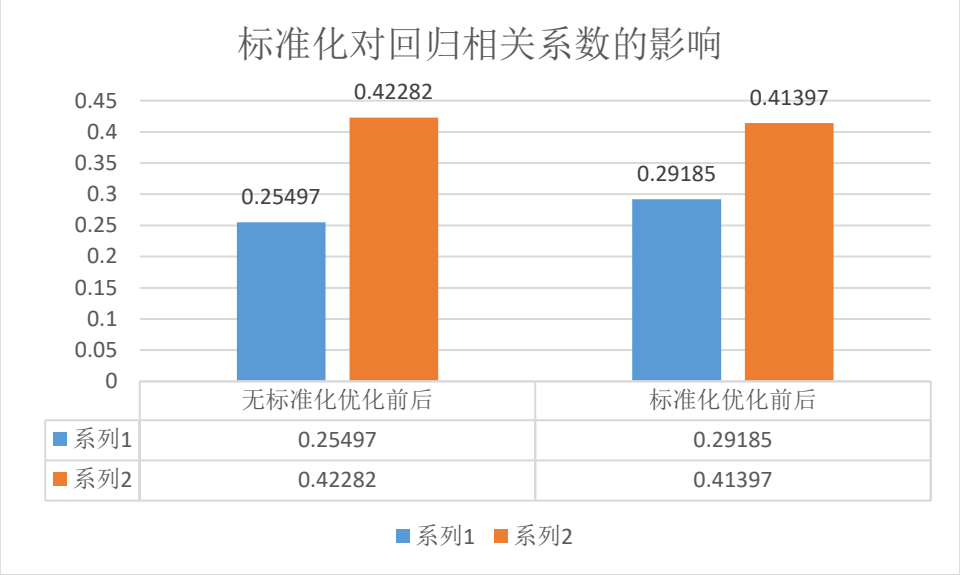
在考虑了关键词之后再对文本数据的虚词进行处理也会进一步地提高 KNN 算法的准确度。



从测评结果也可以看出，距离公式对回归问题中的相关系数的影响，也是考虑了调和平均数对关键特征词的提取具有良好的效果，结合结果预测的分析，可知在关键字的提取的时候可以考取去掉部分虚词。

④ 对权重进行标准化

这个主要是针对回归问题而言的。下面是相同的 K 取值下，考虑了距离公式的选取以及去掉虚词等优化前后权重是否标准化的对比：



从测评结果可以看出，在没有做出任何优化的时候，对权重进行标准化会对回归问题的相关系数的准确度有所提升，提高了 0.29185-0.25497 = 0.03688，而即便进行了各种优化之后，仍然会对相关系数产生影响，降低了 0.42422 − 0.41397 = 0.01025 ；所以标准化的作用已经不明显了。

3.未优化的 NB

没有优化的 NB 即没有实现拉普拉斯平滑，分类的准确度只有 0.06，而回归问题的相关系数 0.12807.

4.优化的 NB

①采用拉普拉斯平滑

根据算法分析，需要把验证集未出现的单词也加入分母中并且调节分子的参数的大小，公式如下：

$$P_i = \frac{x_i + \alpha}{N + \alpha M}$$

其中 M 表示训练集和验证集没有重复出现的单词

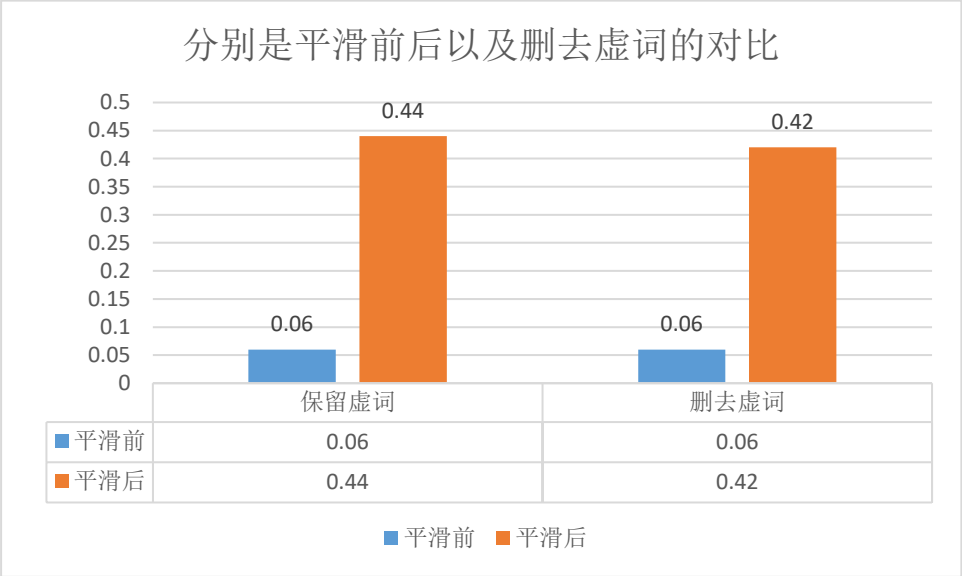
在分类算法中，M 是训练合计与验证集的单词的种数。

（为了排版好看：把下面的测评结果分析放在这里）

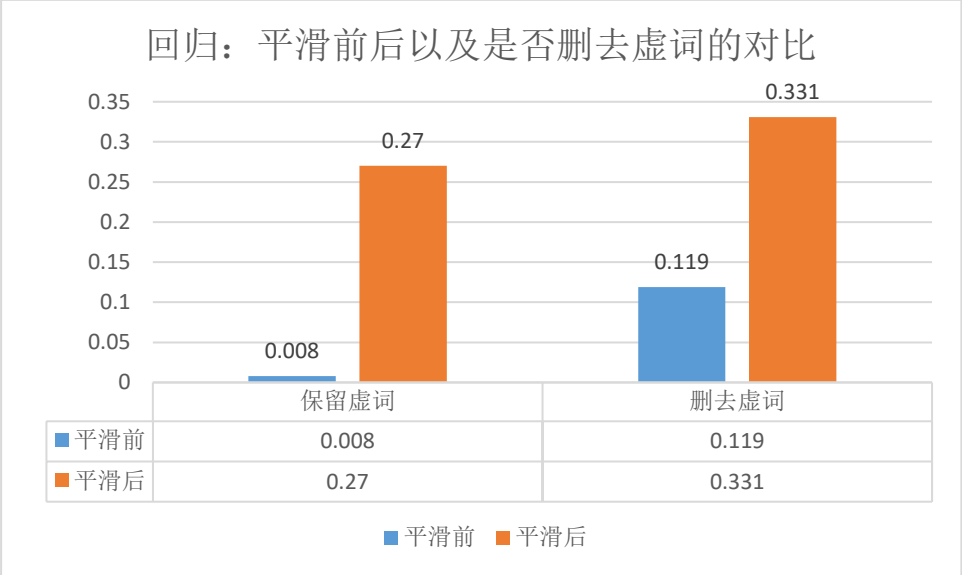
首先在平滑前会导致测试样本中没有出现的单词的时候的概率的连乘直接变成 0，这样子就忽视了其他重要的特征词的比重，所以错误率非常高。对此需要进行平滑处理，这里加的是训练集和验证集的单词的总数，为了使概率的加和趋近与 1。而α参数设置了 0.5；下面左边的是没有平滑和平滑之间的对比。

至于为什么在分类问题对准确率有所提高的除去部分虚词的操作，在朴素贝叶斯反而会使准确率下降：主要是因为，朴素贝叶斯已经考虑得到词频的效果了，而我在分析情感与文本的时候有提到过，情感的产生与单词的组合之间有着很难划分的关系。简单地调出虚词或许在 onehot 矩阵中以及在提取关键字的时候有作用，但是考虑到朴素贝叶斯基于先验概率

和似然度的乘积得到后验概率，基于一定的样本经验以及词频的权重。所以不能简单地删去虚词，或者说不能主观地认为部分虚词应该被删去。所以需要结合实际情况，实验证明在 NB 分类的时候不应该删去虚词。



下面是关于回归在平滑前和平滑后以及是否去除虚词的对比。



分析可得，在回归算法上，删除虚词会比不删虚词的效果要好。

四、思考题

1. 在 KNN 算法处理回归问题上，为什么选取距离的倒数作为概率的权重？

由于距离越小就意味着越接近那一类属，而距离的倒数就是简单地将权重对应增大。

2. 在 KNN 算法处理回归问题上，如何让同一测试样本的各个情感概率总和为 1？

有两种归一化，首先在列向求取类属的概率的时候可以列向对其进行权重的归一化，然后得到所有类属的概率之后可以横向再次归一化，减少浮点误差，让样本中的各个情感概率

总和为 1。

列向归一化：

$$\mathrm{p}(\mathrm{e}) = \sum_{i=1}^{k} \frac{\mathrm{Pi\_train}}{di(traini,\ test) * \sum_{j=1}^{k} \frac{1}{di(traini,\ test)}}$$

行向归一化：

$$\mathrm{p(e)} = \frac{P(e)}{\sum_{i=1}^{n} p(i)}$$

3. 在 KNN 算法中，在矩阵稀疏程度不同的时候，曼哈顿距离和欧式距离有什么区别，为什么？

当矩阵特别稀疏的时候，在 KNN 算法中，曼哈顿距离和欧式距离的效果没什么不同，直观来看，任何两个样本的特征向量都是错开的。

当矩阵稠密的时候，通过观察曼哈顿距离与欧式距离之间的表达式知道欧式距离会更加专注计算同纬度特征向量的距离，所以欧式距离回比曼哈顿更加精确。

欧式距离的平方：$(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots . + (x_n - y_n)^2$

曼哈顿距离的平方：$(|x_1 - y_1| + |x_2 - y_2| + .. + |x_1 - y_1|)^2$

$$= (x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots . + (x_n - y_n)^2 + (x_1 - y_1)(x_2 - y_2) +$$

$$(x_1 - y_1)(x_3 - y_3) + \cdots . (x_1 - y_1)(x_n - y_n) + \cdots \ldots$$

（注意：对于 One-hot 矩阵，在数值上，曼哈顿距离一直是欧式距离的平方，这也在另一方面说明了 One-hot 矩阵的缺陷）。

4. 朴素贝叶斯算法处理分类问题上，伯努利模型和多项式模型分别有什么优缺点？

首先区别一下伯努利模型和多项式模型，两者的计算粒度是不同的，伯努利模型基于文件，而多项式基于特征词。计算起来伯努利模型比较简单，不过由于基于文本会忽视了词频的概念，从而降低了出现较多的词的权重。多项式考虑到了词频，增加了关键字的拟合，不过数据处理上稍微比伯努利模型复杂一点

5. 如果测试集出现中出现了一个之前全词典中没有出现过的词该如何解决？

把该单词计入到训练集的单词表中。对于 KNN 算法，也需要考虑这个单词对于训练文本之间的距离。对于 NB 算法，则是在平滑的过程中还要在分母加上新单词的数目。

|---------- 如有优化，重复 1，2 步的展示，分析优化后结果 ----------|

实验问题记录：