中山大学计算机学院

人工智能

本科生实验报告

（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	系统结构信息计算方向	专业（方向）	信息与计算科学
学号	20337268	姓名	张文沁

一、 实验题目

> **利用朴素贝叶斯方法、k-NN分类与k-NN回归完成文本信息情感分类训练**

二、 实验内容

1. 算法原理

1. 朴素贝叶斯方法：

> 贝叶斯学派的思想可以概括为先验概率+数据=后验概率。也就是说我们在实际问题中需要得到的后验概率，可以通过先验概率和数据一起综合得到。
>
> 被频率学派攻击的就是先验概率，一般来说先验概率就是我们对于数据所在领域的历史经验，但是这个经验常常难以量化或者模型化，于是贝叶斯学派大胆的假设先验分布的模型，比如正态分布，beta分布等。
>
> 我们先看看条件独立公式，如果X和Y相互独立，则有：
>
> $$P(X,Y)=P(X)P(Y)$$
>
> 然后看条件概率公式：
>
> $$P(Y|X)=P(X,Y)/P(X)$$
>
> $$P(X|Y)=P(X,Y)/P(Y)$$
>
> 或者说：
>
> $$P(Y|X)=P(X|Y)P(Y)/P(X)$$
>
> 然后是全概率公式：
>
> $$P(X)=\sum kP(X|Y=Yk)P(Yk)\text{其中}\sum kP(Yk)=1$$
>
> 于是贝叶斯公式：
>
> $$P(Yk|X)=P(X|Yk)P(Yk)/\sum kP(X|Y=Yk)P(Yk)$$
>
> 从统计学知识回到我们的数据分析。假如我们的分类模型样本是：
>
> $$(x(1)^1,x(1)^2,...x(1)^n,y^1),(x(2)^1,x(2)^2,...x(2)^n,y^2),...(x(m)^1,x(m)^2,...x(m)^n,y^m)$$
>
> 即我们有m个样本，每个样本有n个特征，特征输出有K个类别，定义为
>
> $$C1,C2,...,CK$$
>
> 从样本我们可以学习得到朴素贝叶斯的先验分布
>
> $$P(Y=Ck)(k=1,2,...K)$$

接着学习到条件概率分布

$$P(X=x|Y=Ck)=P(X1=x1,X2=x2,...Xn=xn|Y=Ck)$$

然后得到联合分布P(X,Y)

$$P(X,Y=Ck)=P(Y=Ck)P(X=x|Y=Ck)(1)$$

$$=P(Y=Ck)P(X1=x1,X2=x2,...Xn=xn|Y=Ck)(2)$$

上式中，P(Y=CK)是由最大似然估计求得，但是条件概率比较难求，于是**假设X的n个维度之间是相互独立的**，则可以得到新的公式：

$$P(X1=x1,X2=x2,...Xn=xn|Y=Ck)=P(X1=x1|Y=Ck)P(X2=x2|Y=Ck)...P(Xn=xn|Y=Ck)$$

从上式可以看出，这个很难的条件分布大大的简化了，但是这也可能带来预测的不准确性，因为特征之间可能不独立，如果真的不独立，则建议使用其他模型。

但是一般情况下，样本的特征之间独立这个条件的确是弱成立的，尤其是数据量非常大的时候。虽然我们牺牲了准确性，但是得到的好处是模型的条件分布的计算大大简化了，这就是贝叶斯模型的选择。

最后，回到贝叶斯模型，如何确定新的集合属于哪个特征类型：

既然是贝叶斯模型，当然是后验概率最大化来判断分类了。我们只要计算出所有的K个条件概率

$$P(Y=Ck|X=X(test))P(Y=Ck|X=X(test))$$

然后找出最大的条件概率对应的类别，这就是朴素贝叶斯的预测了。

而贝叶斯模型中又有不同的几种概率模型，如下面所述：

1. **高斯模型（常用于连续量）：**

> 有些特征可能是连续型变量，比如说人的身高，物体的长度，这些特征可以转换成离散型的值，
> 比如：
> 如果身高在160cm以下，特征值为1；
> 在160cm和170cm之间，特征值为2；
> 在170cm之上，特征值为3。
> 也可以这样转换，将身高转换为3个特征，分别是f1、f2、f3，
> 如果身高是160cm以下，这三个特征的值分别是1、0、0，
> 在160cm和170cm之间，这三个特征的值分别是0、1、0，
> 若身高在170cm之上，这三个特征的值分别是0、0、1。
> 不过这些方式都不够细腻，高斯模型可以解决这个问题。高斯模型假设这些一个特征的所有属于某个类别的观测值符合高斯分布，也就是使用高斯公式化计算各种特征值下的概率。

2. **多项式模型：**

$$p(x_k|e_i)=\frac{nW_{e_i}(x_k)}{nW_{e_i}} \qquad p(e_i)=\frac{N_{e_i}}{N}$$

$$\arg\max_{e_i}\prod_{k=1}^{K}p(x_k|e_i)p(e_i)$$

> 该模型常用于文本分类，特征是单词，值是单词的出现次数。
>
> 其中，nWei(Xk)是类别下ei特征出现Xk的总次数；nWei是类别下所有特征ei单词总次数。
>
> 对应到文本分类里，如果单词word在一篇分类为label1的文档中出现了5次，那么的值会增加5。如果是去除了重复单词的，那么值会增加1。是特征的数量，在文本分类中就是去重后的所有单词的数量。的取值范围是[0,1]，比较常见的是取值为1。
>
> 待预测样本中的特征在训练时可能没有出现，如果没有出现，则值为0，如果直接拿来计算该样本属于某个分类的概率，结果都将是0。在分子中加入，在分母中加入可以解决这个问题。

3. **伯努利模型：**

$$\arg\max_{e_i} \prod_{k=1}^{K} p(x_k|e_i)p(e_i)$$

$$\mathrm{p}(x_k|e_i) = \frac{n_{e_i}(x_k)}{N_{e_i}} \qquad p(e_i) = \frac{N_{e_i}}{N}$$

> 伯努利模型中，对于一个样本来说，其特征用的是全局的特征。
>
> 在伯努利模型中，每个特征的取值是布尔型的，即true和false，或者1和0。在文本分类中，就是一个特征有没有在一个文档中出现。
>
> 如果特征值值为1,那么这个特征在文档中出现过
>
> 如果特征值值为0,那么则是没有。
>
> 这意味着，“没有某个特征”也是一个特征。

4. 拉普拉斯平滑：

> 零概率问题：在计算事件的概率时，如果某个事件在观察样本库（训练集）中没有出现过，会导致该事件的概率结果是0。这是不合理的，不能因为一个事件没有观察到，就被认为该事件一定不可能发生（即该事件的概率为0）。
>
> 拉普拉斯平滑(Laplacian smoothing) 是为了解决零概率的问题。
>
> 法国数学家 拉普拉斯 最早提出用 加1 的方法，估计没有出现过的现象的概率。
>
> 理论假设：假定训练样本很大时，每个分量x的计数加1造成的估计概率变化可以忽略不计，但可以方便有效的避免零概率问题

对于一个随机变量 z，它的取值范围是{1,2,3…,k}，对于 m 次试验后的观测结果$\{z^{(1)}, z^{(2)}, z^{(3)}, \ldots, z^{(m)}\}$，极大似然估计按照下式计算：

$$\varphi_j = \frac{\sum_{i=1}^{m} I\{z^{(i)} = j\}}{m} \tag{25}$$

使用 Laplace 平滑后，计算公式变为：

$$\varphi_j = \frac{\sum_{i=1}^{m} I\{z^{(i)} = j\} + 1}{m + \mathrm{k}} \tag{26}$$

即在分母上加上取值范围的大小，在分子加 1。

引入到贝叶斯公式中便产生了下述的概率公式：

$$Bernoulli: p(x_k|e_i) = \frac{n_{e_i}(x_k) + 1}{N_{e_i} + 2}$$

$$Multinomial: p(x_k|e_i) = \frac{nw_{e_i}(x_k) + 1}{nw_{e_i} + V_{e_i}}$$

其中Vei是类别为ei的文本中不重复单词的数量。

2. k-邻近算法：

1. **KNN-classification:**

说起KNN,会想起一个比较经典的聚类算法K_means，但其实，这两种算法之间是有区别的，K_means本质上是无监督学习而KNN是监督学习，Kmeans是聚类算法而KNN是分类（或回归）算法。

那么在聚类,分类,回归中最重要的一个衡量指标是**距离**,首先介绍几种常见的距离算法

1. 曼哈顿距离:

$$dist(x_i, x_j) = \sum_{k=1}^{n} |x_i^k - x_j^k|$$

2. 欧氏距离:

$$dist(x_i, x_j) = \sqrt{\sum_{k=1}^{n} (x_i^k - x_j^k)^2}$$

3. 明可夫斯基距离(一种概括距离):

$$dist(x_i, x_j) = (\sum_{k=1}^{n} |x_i^k - x_j^k|^p)^{1/p}$$

4. 切比雪夫距离:

$$dist(x_i, x_j) = \lim_{x->\infty} (\sum_{k=1}^{n} |x_i^k - x_j^k|^p)^{1/p}$$

总而言之，KNN算法思想可以用一句话概括：**如果一个样本在特征空间中的K个最相似（即特征空间中最邻近，用上面的距离公式描述）的样本中的大多数属于某一个类别，则该样本也属于这个类别。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。**

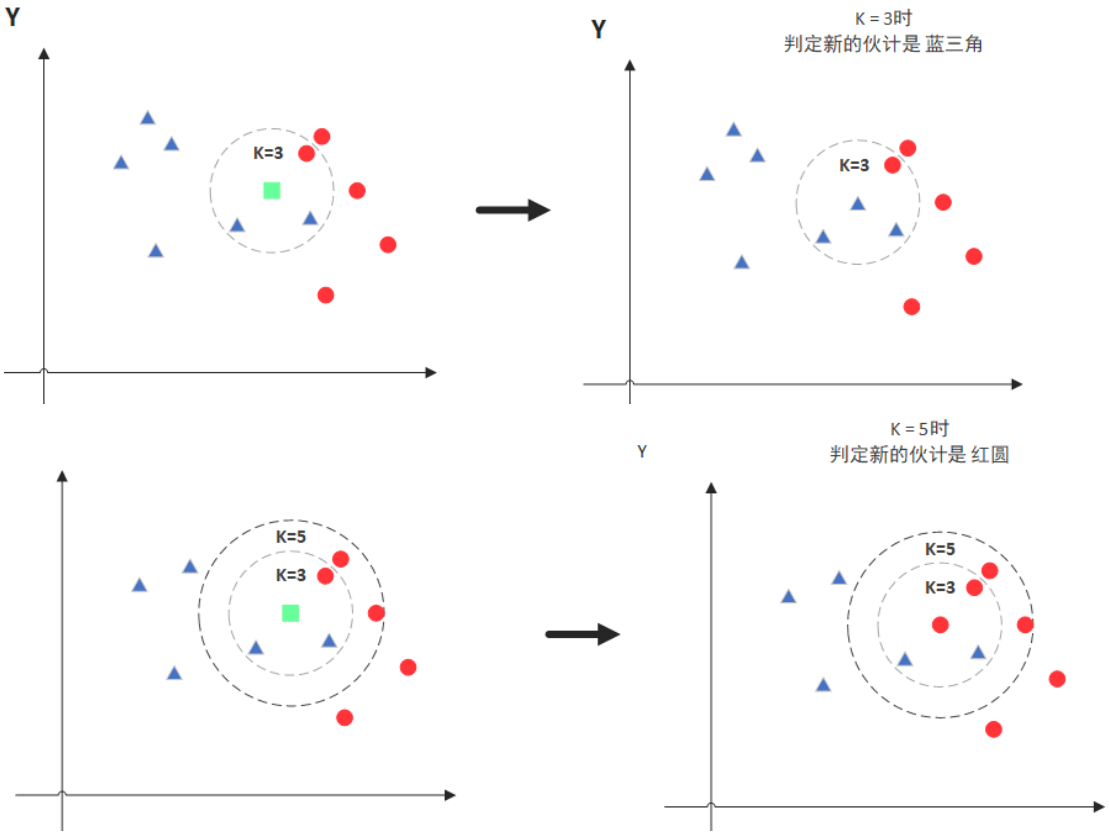
于是步骤大致如下:

1. 计算目标点到其他点的距离

2. 选取距离最近的K个点

3. 选取众数作为目标点的特征

示例如下:



于是很容易看到K的不同对结果的影响.那么**下面讨论KNN算法中另一个至关重要的变量K的取值如何最优.**

可以通过**交叉验证**的方法得到最优K,（将样本数据按照一定比例，拆分出训练用的数据和验证用的数据，比如6：4拆分出部分训练数据和验证数据），从选取一个较小的K值开始，不断增加K的值，然后计算验证集合准确率，最终找到一个比较合适的K值.

经过验证,随着K的增大,准确率先升后降,具体数值见结果部分.

那么最后,KNN在计算距离时并不一定要使用"距离",可以采用相似度进行替代,例如进行加权平均的概率,或者余弦相似度,Jaccard相似度.

如下,余弦相似度,计算得到的结果实际为两个向量之间夹角的余弦值,余弦值越大,说明这两个向量越相似.

$$\cos(\theta) = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n}(x_i)^2} \times \sqrt{\sum_{i=1}^{n}(y_i)^2}}$$

Jaccard相似度:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

jaccard系数取值范围[0,1]
当A==B时，jaccard系数为1；当A与B不相交，jaccard系数为0
jaccard距离表示样本或集合的不相似程度，jaccard距离越大，样本相似度越低。故jaccard距离用于描述不相似度，缺点是只适用于二元数据的集合

2. *伪代码*

3. *关键代码展示（带注释）*

1. **贝叶斯模型:**

伯努利模型:

```
def Bernoulli(N,str):
    P = []
    P_ei = []
    Nei = [0,0,0,0,0,0]
    Nei_xk = []
    emo_array = get_emotion(data_train,N)
    for i in range(N):
        Nei[emo_array[i]]  += 1
    for j in range(6):
        P_ei.append(Nei[j]*1.0/N)
    for m in range(len(str)):
        index = [0,0,0,0,0,0]
        for n in range(N):
            if str[m] in words_train[n]:
                index[emo_array[n]] += 1
        Nei_xk.append(index)
    for i in range(6):
        index_P = P_ei[i]
        for k in range(len(str)):
            P_Xk_Ei = (Nei_xk[k][i]*1.0 + 1) / (Nei[i]*1.0+2)
            index_P = index_P * P_Xk_Ei
        P.append(index_P)
    return P
```

多项式算法:

```
def Polynomial(N,str):
    P = []
    P_ei = []
    NWei = [0,0,0,0,0,0]
    Nei = [0,0,0,0,0,0]
    NWei_xk = []
    V = 0
    emo_array = get_emotion(data_train,N)
    for i in range(N):
        NWei[emo_array[i]] += len(words_train[i])
        V += len(words_train[i])
        Nei[emo_array[i]]  += 1
    for j in range(6):
        P_ei.append(Nei[j]*1.0/N)
    for m in range(len(str)):
        index = [0,0,0,0,0,0]
        for n in range(N):
            if str[m] in words_train[n]:
                index[emo_array[n]] += 1
        NWei_xk.append(index)
    for i in range(6):
        index_P = P_ei[i]
        for k in range(len(str)):
            P_Xk_Ei = (NWei_xk[k][i]*1.0 + 1) / (Nei[i]*1.0 + V)
            index_P = index_P*P_Xk_Ei
        P.append(index_P)
    return P
```

2. **KNN模型**

KNN-classification

```
# The classic KNN with Eulic-distance
def Eucli_distance_KNN(str):
    # the sum of each emotion
    lable = [0,0,0,0,0,0]
    distance = []
    # as i mentioned before, I used a (not_in_corpus) to get extra distance
    not_in_corpus = 0
    # str's one-hot
    index = [0 for n in range(len_corpus)]
    for j in range(len(str)):
        if str[j] in corpus:
            index_position = corpus.index(str[j])
            index[index_position] += 1
        else: # if it's not in corpus, count 1
            not_in_corpus += 1
    _,emotion_train = get_emotion(data_train,Len_words)
    # distance and emotion
    for i in range(Len_words):
        # get rid of the trouble to cauclate test's corpus

        # Eulic distance^2
        Eulic = not_in_corpus
        for j in range(len_corpus):
            Eulic += math.pow(One_hot[i][j] - index[j],2)
        index_distance = math.sqrt(Eulic)
        # get distace and emotion in the meantime
        distance.append((index_distance,emotion_train[i]))
    # to get first Kst
    distance.sort(key=lambda x:x[0])
    for j in range(K+1):
        lable[distance[j][1]] += 1
    # return the max's lable
    return lable.index(max(lable))
```

KNN-regression

```
# The regressive KNN with Eulic distance
def Eucli_distance_KNN_regression(str):
    lable = [0,0,0,0,0,0]
    distance = []
    not_in_corpus = 0
    index = [0 for n in range(len_corpus)]
    for j in range(len(str)):
        if str[j] in corpus:
            index_position = corpus.index(str[j])
            index[index_position] += 1
        else:
            not_in_corpus += 1
    # in regressive KNN, we just need all 6 emotions
    all_emotion,_ = get_emotion(data_train,Len_words)
    for i in range(Len_words):
        Eulic = not_in_corpus
        for j in range(len_corpus):
            Eulic += math.pow(One_hot[i][j] - index[j],2)
        index_distance = math.sqrt(Eulic)
        distance.append(index_distance)
    distance.sort()
    # to get final P
    for i in range(6):
        index = 0
        for j in range(K):
            index = index + (all_emotion[j][i]/distance[j])
```

```
        lable[i] = index
    return lable.index(max(lable))
```

3. **预处理:**

TF-IDF矩阵计算:

```
def feature_select(list_words):
    #总词频统计
    doc_frequency=defaultdict(int)
    #print(doc_frequency)
    for word_list in list_words:
        for i in word_list:
            doc_frequency[i]+=1

    #计算每个词的TF值
    word_tf={}  #存储没个词的tf值
    for i in doc_frequency:
        word_tf[i]=doc_frequency[i]/sum(doc_frequency.values())

    #计算每个词的IDF值
    doc_num=len(list_words)
    word_idf={} #存储每个词的idf值
    word_doc=defaultdict(int) #存储包含该词的文档数
    for i in doc_frequency:
        for j in list_words:
            if i in j:
                word_doc[i]+=1
    for i in doc_frequency:
        word_idf[i]=math.log(doc_num/(word_doc[i]+1))

    #计算每个词的TF*IDF的值
    word_tf_idf={}
    for i in doc_frequency:
        word_tf_idf[i]=word_tf[i]*word_idf[i]

    # 对字典按值由大到小排序
    dict_feature_select=sorted(word_tf_idf.items(),key=operator.itemgetter(1),reverse=True)
    return dict_feature_select
```

One-Hot矩阵计算:

```
# One-hot
# corpus 语料库
One_hot = []
for i in range(len(words)):
    index = [0 for n in range(len_corpus)]
    for j in range(len(words_train[i])):
        index_position = corpus.index(words_train[i][j])
        index[index_position] += 1
    One_hot.append(index)
```

Bag of Words 与 One-hot仅在加法上有不同,对性能和结果影响不大,在此不做展示

距离计算:

欧氏距离:

```
for i in range(Len_words):
    Eulic = not_in_corpus
    for j in range(len_corpus):
        Eulic += math.pow(One_hot[i][j] - index[j],1)
    index_distance = math.sqrt(Eulic)
    distance.append(index_distance)
distance.sort()
```

曼哈顿距离：

```
for i in range(Len_words):
    Eulic = not_in_corpus
    for j in range(len_corpus):
        Eulic += math.pow(One_hot[i][j] - index[j],1)
    distance.append(Eulic)
distance.sort()
```

余弦相似度：

```
for i in range(Len_words):
    len_A = 0
    len_B = 0
    product = 0
    for j in range(len(words[i])):
        # to get each sentecne's vector
        index_str[corpus.index(j)] += 1
    for k in range(corpus):
        # to get sentence's length
        len_A += index_str[k]
        # to get sentence(waiting for verification)'s length
        len_B += index[k]
        # to get their product
        product += index[k] * index_str[k]
    distance.append(1-(product*1.0)/(len_A*len_B*1.0))
distance.sort()
```

Jaccard相似度

```
index = [0 for n in range(len_corpus)]
for j in range(len(str)):
    if str[j] in corpus:
        index_position = corpus.index(str[j])
        index[index_position] += 1
    else:
        not_in_corpus += 1

for i in range(Len_words):
    # because we have known the sum of words that are not in the corpus, so the length of corpus &
Eulic is the size of the union set and their difference is the size of intersection
    Eulic = not_in_corpus
    size_sentence = len(words[i])
    distance.append((len(str) - Eulic)*1.0/(size_sentence + Eulic))
```

4. *创新点&优化（如果有）*

> **1. 在计算欧氏距离时，因为采用了One-hot矩阵的方法，所以如果按照一般步骤来，需要计算两次语料库，但是因为一旦不在语料库中一定会有距离1，于是在计算欧氏距离时，遇到不在初始语料库的词语，直接进行加一操作，免去了拓展语料库的麻烦**

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）

> **具体结果见csv文件,该处展示准确率**

贝叶斯模型:

多项式模型:

```
y\launcher' '57156' '--' 'f:\CodeFile for Python\AI'
PS F:\CodeFile for Python>  F:; cd 'F:\CodeFile for
y\launcher' '57169' '--' 'f:\CodeFile for Python\AI'
The accuracy is : 0.4308681672
0    5    3
1    2    3
2    3    5
3    2    3
4    4    3
5    3    4
6    2    2
7    3    3
8    5    3
9    2    3
10    5    3
11    4    4
12    4    3
```

伯努利模型：

```
3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 4, 3, 3, 3, 4, 3, 3, 3,
The accuracy is : 0.3536977492
```

KNN模型:

普通KNN模型:(因为最大值返回的时候返回下标较小的那个,而众数容易重复,误差较大)

对K的尝试：

1. K = sqrt(N)

```
022.6.0\pythonFiles\lib\python\debu
The accuracy is : 0.3954983923
[3, 3, 3, 3, 3, 3, 4, 3, 3, 3, 4, 3,
 3, 3, 3, 3, 3, 3, 3, 4, 3, 2, 3, 3
```

2. K = N/5

```
PS F:\CodeFile for Python>  F:; cd
022.6.0\pythonFiles\lib\python\debu
The accuracy is : 0.3601286174
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
```

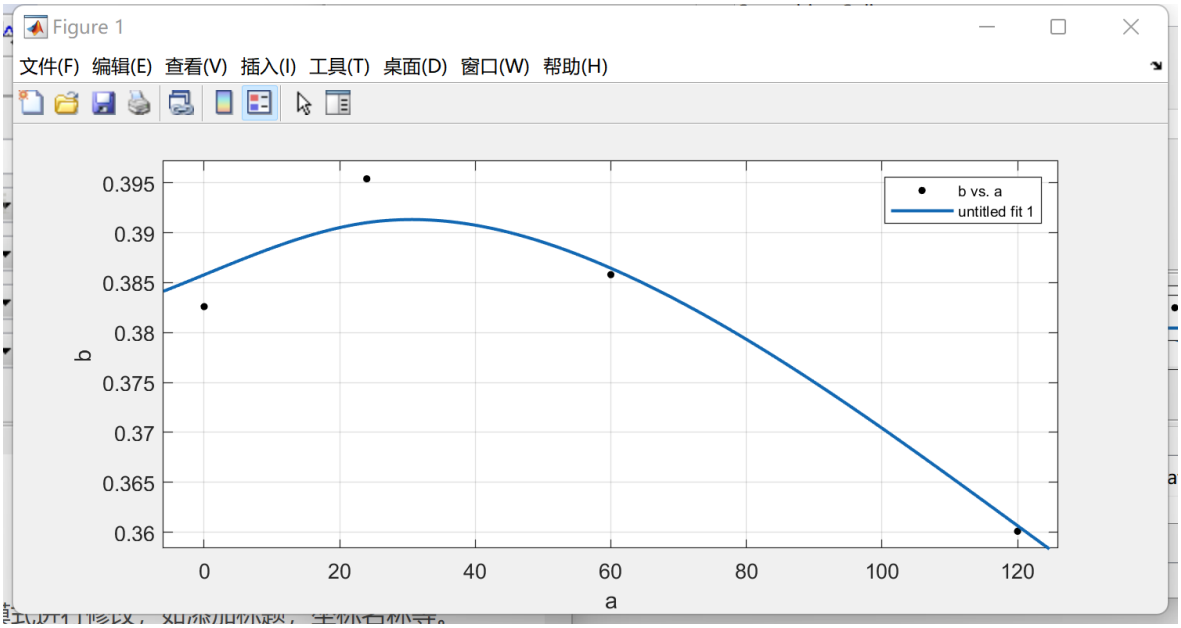
3. K = N/10

```
022.6.0\pythonFiles\lib\python\deb
The accuracy is : 0.38585209
[3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4,
```

4. K = N/100

```
022.6.0\pythonFiles\lib\python\deb
The accuracy is : 0.3826366559
[2, 3, 2, 2, 3, 3, 2, 3, 2, 2, 3,
 2, 2, 2, 3, 3, 3, 2, 3, 2, 2, 3,
```

下面是用matlab根据有限的数据点拟合得到的曲线:最高点大致在sqrt(N)处



对距离的尝试：

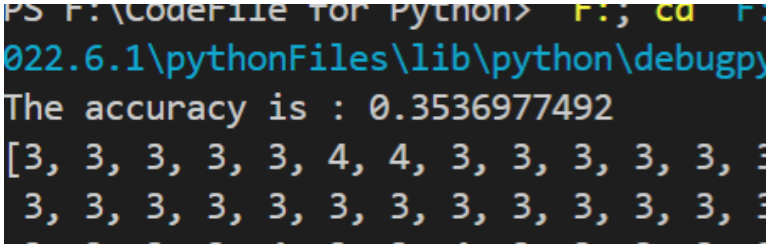
1. 欧氏距离如上图所示，在38%附近徘徊
2. 曼哈顿距离：

```
# In regressive KNN, we just need all 6 emotions
all_emotion,_ = get_emotion(data_train,Len_words)
for i in range(Len_words):
    Eulic = not_in_corpus
    for j in range(len_corpus):
        Eulic += math.pow(One_hot[i][j] - index[j],1)
    #index_distance = math.sqrt(Eulic)
    #distance.append(index_distance)
    distance.append(Eulic)
distance.sort()
```

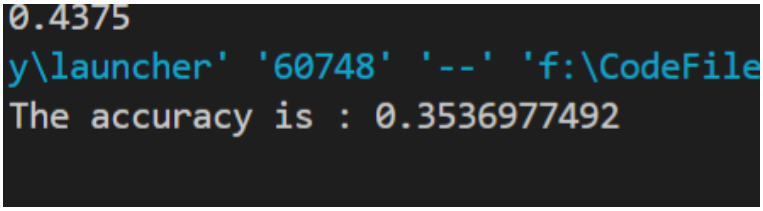
```
022.6.1\pythonFiles\lib\python\debu
The accuracy is : 0.3536977492
[3, 3, 3, 3, 3, 4, 4, 3, 3, 3, 3, 3
```

> 正确率低于之前欧氏距离

3. 余弦相似度



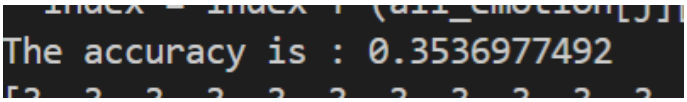
4. Jaccard相似度，原理见上面解释（和余弦相似度、曼哈顿距离都准确率一致，也许因为3，4容易判断？）



> **综上所述，欧氏距离和sqrt(N)的组合准确度最高，于是所有最终代码使用的都是这两种**

KNN回归模型:

直接选取了sqrt(N)作为K值:



要哭了

	I	J	K	L	M
		anger	disgust	fear	joy
	r	0.159036315	0.311907884	0.168963313	0.123863433
	average	0.166775308			
	evaluation	极弱相关 加油哦小辣鸡			

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

> **总结:**
>
> 贝叶斯方法更加快速, 但是准确率较低, 多项式模型准确率高于伯努利模型.
>
> KNN方法因为复杂度为O(M * N * V), 于是耗时更长

四、 思考题

1. 朴素贝叶斯中，伯努利模型和多项式模型分别有什么优缺点?

> 二者的计算粒度不一样，多项式模型以单词为粒度，伯努利模型以文件为粒度，因此二者的先验概率和类条件概率的计算方法都不同。
>
> 计算后验概率时，对于一个文档d，多项式模型中，只有在d中出现过的单词，才会参与后验概率计算，伯努利模型中，没有在d中出现，但是在全局单词表中出现的单词，也会参与计算，不过是作为“反方”参与的。
>
> **总而言之，在实验过程中发现多项式比伯努利模型有更高的精度，但是多项式模型容易出现极端情况。**

因为多项式合理考虑了每个单词而不只是句子，单词会更有代表性。但是会有极端情况，例如整个句子都是 the the the，此时该句子不具有任何参考价值。

2. IDF数值有什么含义？TF-IDF数值有什么含义?

$$TF\omega = \frac{\text{在某一类词条中}\omega\text{出现的次数}}{\text{该文档中总单词数}}$$

词频TF计算了一个单词在文档中出现的次数，一个单词的重要性和它在文档中出现的次数成正比。一个词预测主题的能力越强，权重越大，反之，权重越小。

所有统计的文章中，一些词只是在其中很少几篇文章中出现，那么这样的词对文章的主题的作用很大，这些词的权重应该设计的较大。IDF就是在完成这样的工作.

$$TF - IDF = TF * log(\frac{\text{文档数}}{\text{该单词出现的文档数} + 1})$$

逆向文档频率IDF，是指 一个单词在文档中的区分度。一个单词出现在的文档数越少，就越能通过这个单词把该文档和其他文档区分开。IDF越大就代表该单词的区分度越大。

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

所以TF-IDF实际上是词频TF和逆向文档频率IDF的乘积。这样我们倾向于找到TF和IDF取值都高的单词作为区分，即这个单词在一个文档中出现的次数多，同时又很少出现在其他文档中。这样的单词适合用于分类。如果包含词条t的文档越少，IDF越大，则说明词条具有很好的类别区分能力。

3. kNN中为什么是距离的倒数？如果要求同一测试样本的各个情感概率总和为1，应该如何处理?

因为距离越远表示相似度越低，那么相似度越低，权重也应该相应地更低，于是选择使用倒数作为权重。

可以重新进行概率的分配，例如，使P1 = P1/(P1+P2+...+Pn)，其他类似。

五、 参考资料

https://blog.csdn.net/Cyril_KI/article/details/107302163