中山大學
SUN YAT-SEN UNIVERSITY

实 验 报 告

人工智能实验报告–KNN

学　　院：数据科学与计算机学院

专　　业：计算机科学与技术

学生姓名：刘斯宇

学　　号：17341110

2019 年 9 月 6 日

目录

1 文本数据集简单处理：TF-IDF

1.1 实验原理

这个实验的主要思路是对多个文本进行编码处理，从而生成 TF-IDF 矩阵，将文本信息用数学表达式表达。

1.1.1 TF 矩阵

在介绍 TF 矩阵之前需要了解什么是 OneHot 矩阵，简单点来说就是使用一个向量来表示一篇文章，向量的长度为词汇表的大小，1 表示对应的单词存在，0 则表示不存在。将所有的文本中的单词构成一个词汇表，从而生成一个 onehot 矩阵。同样的，TF 矩阵的大小跟 OneHot 矩阵的大小是一样的，只不过 TF 矩阵的值表示的是频率，TF[i][j] 表示的是单词 word$_list[j]$ i $TF[i][j] = \frac{OneHot[i][j]}{\sum_k OneHot[i][k]}$

1.1.2 IDF 矩阵

逆文本频率。IDF 是为了反映某个词的重要性，修正仅仅用词频表示的词特征值。IDF 反应了一个词在所有文本中出现的频率，如果一个词在很多的文本中出现，那么它的 IDF 值应该低，比如一些定冠词'the','a' 等，因为这些单词并没有很大的相关性。而反过来如果一个词在比较少的文本中出现，那么它的 IDF 值应该高。比如一些专业的名词。一个极端的情况，如果一个词在所有的文本中都出现，那么它的 IDF 值应该为 0。对于一个有 N 个文本的 IDF 矩阵的公式为：

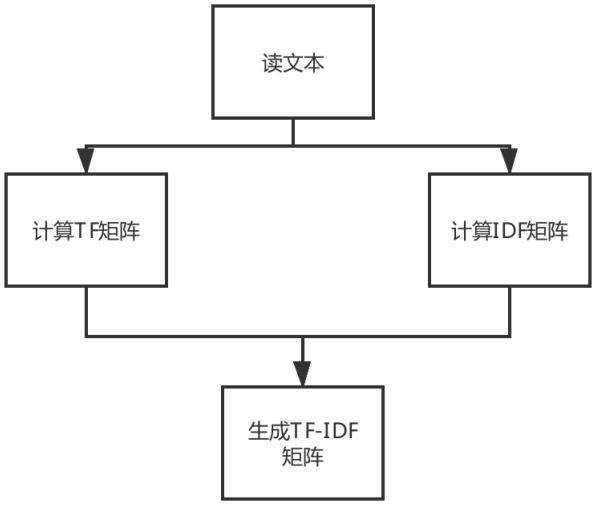
$$IDF[i] = \frac{N}{\sum_{k=0}^{N} OneHot[k][i] + 1}$$

1.1.3 TF-IDF 矩阵

TF-IDF 矩阵中的值是用来表示某个词能够概括该文本的程度。TF-IDF 的值越高，表示这个单词越能够概括这个文本，否则越不能概括。

$$TF_IDF[i][j] = TF[i][j] * IDF[j]$$

1.2 流程图



1.3 关键代码截图

1.3.1 get_TF 函数

方法：分别遍历每个文本中的每个单词，求出这个单词的频率

```
def get_TF(line,col,FILENAME,word_list,word_map):
#word_list->list          word_map->map
    TF=np.zeros((line,col))
    file=open(FILENAME,'r')
    f=file.readlines()
    index=0
    for line in f:
        line=line.strip()
        line_list=line.split('\t')
        line_word=line_list[2].split('␣')
        for word in line_word:
            j=word_map[word]
            TF[index][j]+=1/len(line_word)
        index+=1
    return TF
```

1.3.2 get_IDF 函数

方法：对于每个单词遍历每个文本，看看这个单词在不同文本出现的次数。

```
def get_IDF(col,FILENAME,line,word_list):
```

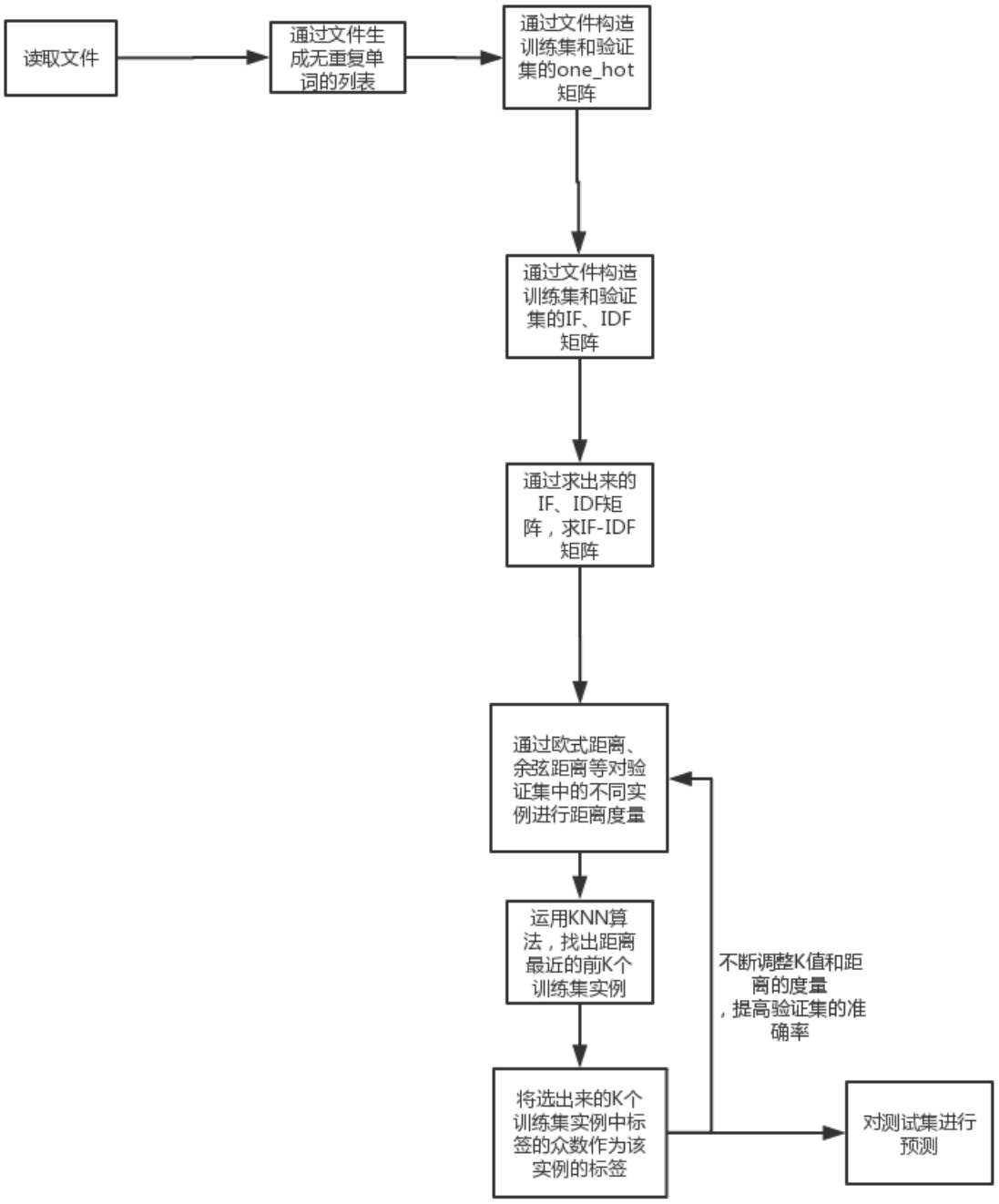
```
    #word_list->list
     IDF=np.zeros(col)
     index=0
     file=open(FILENAME,'r')
     f=file.readlines()
     D=len(f)
     for word in word_list:
         s=0
         for line in f:
             line=line.strip()
             line_list=line.split('\t')
             line_word=line_list[2].split('␣')
             if word in line_word:
                 s+=1
         IDF[index]=math.log(D/(1+s))
         index+=1
     return IDF
```

2 KNN 最近邻算法–分类问题

2.1 实验原理

KNN 最近邻算法来解决分类问题的主要思路就是在训练过程中将多个已经标记好了的样例的 Label 标记下来，然后在预测一个新的样本 T 的分类的时候，分别算出该样本与之前训练集中的所有样本的距离，找出最近的 K 个样例，将这 K 个样例中出现次数最多的 Label 作为该新样本的 Label。

2.2 流程图


读取文件
通过文件生成无重复单词的列表
通过文件构造训练集和验证集的one_hot矩阵
通过文件构造训练集和验证集的IF、IDF矩阵
通过求出来的IF、IDF矩阵，求IF-IDF矩阵
通过欧式距离、余弦距离等对验证集中的不同实例进行距离度量
运用KNN算法，找出距离最近的前K个训练集实例
将选出来的K个训练集实例中标签的众数作为该实例的标签
不断调整K值和距离的度量，提高验证集的准确率
对测试集进行预测

2.3 关键代码截图

2.3.1 测试验证集准确率函数

方法：采用余弦距离来度量文本之间的距离。用 TF-IDF 矩阵求出验证集中的每个文本与训练集中的样本的距离，找出与该样本距离最近的前 K 个样本，把这 K 个样本中的标签众数作为该文本的标签，与该文本的实际标签进行对比。

```
def calcos(VALTFIDF,TFIDF,K,mood_map,val_mood_map):
"""
VALTFIDF->        TFIDF      TFIDF->        TFIDF
K->      K        mood_map->
val_mood_map->

                  K
"""
 line=len(VALTFIDF)
 col=len(VALTFIDF[0])
 num=len(TFIDF)
 acc=0
 for i in range(line):
     distance={}
     for j in range(num):
         res=0
         len1=0
         len2=0
         for k in range(col):
             len1+=np.power(VALTFIDF[i][k],2)
             len2+=np.power(TFIDF[j][k],2)
             res+=VALTFIDF[i][k]*TFIDF[j][k]
         len1=np.power(len1,0.5)
         len2=np.power(len2,0.5)
         distance[j]=res/(len1*len2)
     temp_distance=sorted(distance.items(),key=lambda item:item[1],reverse=True)
     print(temp_distance)
     res_map={}
     for m in range(K):
         mood=mood_map[temp_distance[m][0]]
         if mood not in res_map.keys():
             res_map[mood]=1
         else:
```

```
                res_map[mood]+=1

        res_map=sorted(res_map.items(),key=lambda item:(item[1],-ord(item[0][0])))

        if res_map[-1][0] == val_mood_map[i]:
            acc+=1
        print(i,res_map,val_mood_map[i],acc)

    print(1.0*acc/line)
```

2.4 实验结果

KNN 分类

准确率	欧式距离	余弦距离
K=14	38%	43%
K=$\sqrt{n}$	37%	41．5%

2.5 实验结果分析

在用 KNN 解决分类问题的时候，我最开始是直接用 OneHot 矩阵来进行欧式距离的计算，发现效果其实还不错，准确率有 30%+，然后之后直接用 OneHot 来进行余弦距离的计算，结果发现结果很糟糕，只有 20%，最后跟同学们讨论后发现这样用余弦距离是毫无意义的，之后采用 TF-IDF 矩阵用余弦距离算了之后准确率还是不错的，但是我并没有发现 K 与准确的关系，K=$\sqrt{n}$ 也不总是最优的，还有就是我发现我的预测效果其实真的是不算好的，基本大部分都被预测成了 joy，这可能跟训练集中 joy 的文本比较多相关吧，可能过拟合了。

3 KNN 最近邻算法–回归问题

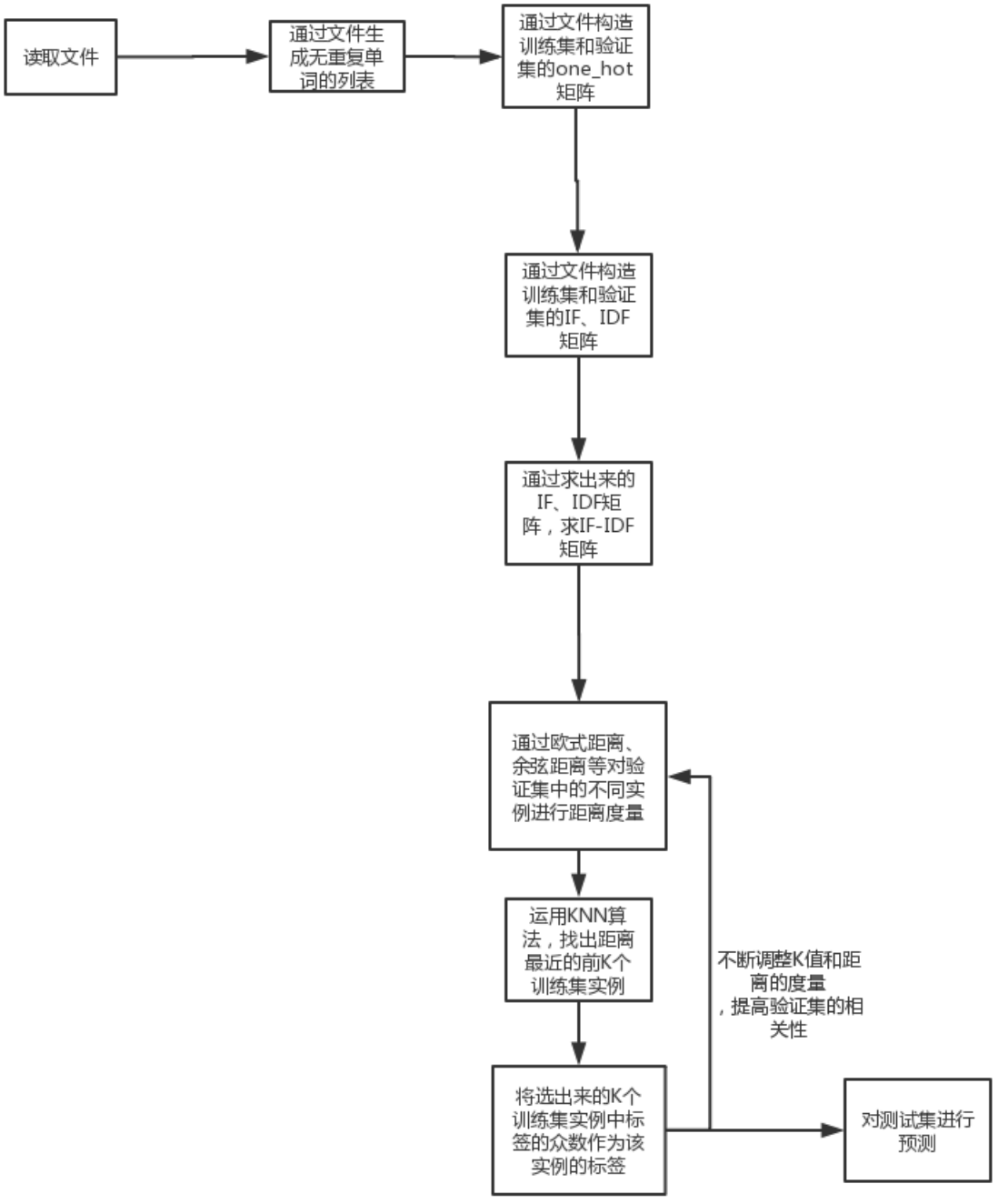
3.1 实验原理

KNN 解决回归问题，其实还是 K 最近算法，只不过这次是连续值的预测。在已经标记好的样本中寻找与预测文本最近的 K 个文本，设这 K 个样本分别为 $S_1, S_2, \ldots, S_K$, 距离分别为 d[1],d[2],...,d[K]，第 i 个样本的第 j 种情感的概率为 S[i][j], 设预测的文本为 T，那么文本 T 的第 j 种情感的概率为

$$T[j] = \sum_{m=0}^{K} \frac{S[m][j]}{d[m]}$$

当然在编程的时候，通常为了避免出现处以 0 的情况，通常在分母加一个很小的常数。当然最后需要将每种感情归一化，得出最后的概率。

3.2 流程图



3.3 关键代码截图

3.3.1 预测函数

方法：采用余弦距离来度量文本之间的距离。用 TF-IDF 矩阵求出验证集中的每个文本与训练集中的样本的距离，找出与该样本距离最近的前 K 个样本，把这 K 个样本中的标签众数作为该文本的标签，然后用公式求出相应的概率。最后还需要的一个步骤是归一化，各种情感的份额作为最后的概率。

```
def calcos(VALTFIDF,TFIDF,K,train_mood_po):
"""
VALTFIDF->        TFIDF      TFIDF->        TFIDF
K->       K       train_mood_po->

                  K
"""
 line=len(VALTFIDF)
 col=len(VALTFIDF[0])

 num=len(TFIDF)

 posibility=np.zeros((line,6))

 for i in range(line):
     distance={}
     for j in range(num):
         res=0
         len1=0
         len2=0
         for k in range(col):
             len1+=np.power(VALTFIDF[i][k],2)
             len2+=np.power(TFIDF[j][k],2)
             res+=VALTFIDF[i][k]*TFIDF[j][k]
         len1=np.power(len1,0.5)
         len2=np.power(len2,0.5)
         distance[j]=res/(len1*len2)
     distance=sorted(distance.items(),key=lambda item:item[1],reverse=True)
     for k in range(K):
         index=distance[k][0]
         for m in range(6):
             posibility[i][m]+=train_mood_po[index][m]/(1-distance[k][1])
```

```
        print(posibility[i])

    for i in range(line):
        t=sum(posibility[i][:])
        for j in range(6):
            posibility[i][j]/=t
        print(sum(posibility[i][:]))
    f=open('res2.csv','w')
    for i in range(line):
        for j in range(6):
            f.write(str(posibility[i][j]))
            f.write(',')
        f.write('\n')
```

3.4 实验结果

KNN 回归

相关性	余弦距离	欧式距离
K=4	38.9%	28%

3.5 实验结果分析

在用 KNN 解决回归问题的时候，我最开始是直接用欧式距离来进行的计算，发现效果其实还不好，相关性只有 23%+，之后采用 TF-IDF 矩阵用余弦距离算了之后相关性能够提升到 38 将 K 换成了 6，发现相关性能够到达 40% 以上了。

4 思考题

4.1 IDF 的第二个计算公式中分母中多了一个 1 是为什么？$idf_i = \log \frac{|D|}{1+|\{j:t_i \in d_j\}|}$

这里是为了防止出现除以 0 的情况，如果某个单词只出现了在了训练集，而没有出现在验证集和测试集中，那么验证集和测试集的 IDF 矩阵的计算的时候就可能会出现除以 0 的情况。

4.2 IDF 数值有什么含义？TF-IDF 数值有什么含义？

对于 TF 矩阵，它的值描述了一个单词在该文本中出现的频率，也就是说，对于 TF 矩阵，TF[i][j] 的值表示的是在第 i 个文本中，$word_list[j]$ 单词在这个文本中所出现的比例。

$$TF[i][j] = \frac{OneHot[i][j]}{\sum_k OneHot[i][k]}$$

对于 IDF 矩阵–表示的是逆文本频率。IDF 是为了反映某个词的重要性，修正仅仅用词频表示的词特征值。IDF 反应了一个词在所有文本中出现的频率，如果一个词在很多的文本中出现，那么它的 IDF 值应该低，比如一些定冠词'the','a' 等，因为这些单词并没有很大的相关性。而反过来如果一个词在比较少的文本中出现，那么它的 IDF 值应该高。比如一些专业的名词。一个极端的情况，如果一个词在所有的文本中都出现，那么它的 IDF 值应该为 0。对于一个有 N 个文本的 IDF 矩阵的公式为：

$$IDF[i] = \frac{N}{\sum_{k=0}^{N} OneHot[k][i] + 1}$$

TF-IDF 矩阵中的值是用来表示某个词能够概括该文本的程度。TF-IDF 的值越高，表示这个单词越能够概括这个文本，否则越不能概括。

$$TF_IDF[i][j] = TF[i][j] * IDF[j]$$

4.3 同一个测试样本的各个情感概率总和应该为 1，应该如何处理？

这里我们采用的方法是归一化，也就是用每种感情的计算结果的占比来当作最后每种情感的概率。设情感 i 的计算结果为 m_i，那么最后情感的 i 的概率为

$$P_i = \frac{m_i}{\sum_j m_j}$$

4.4 在用 KNN 进行回归的时候，为什么每种概率的权重是距离的倒数？

因为如果某个训练样本越接近测试样本的话，那么他们的距离越小，该训练样本所能代表这个样本的概率就越大，那么这个样本的权重就需要越大，而距离的倒数刚刚好越大。

5 实验收获

这个实验做了我我一星期之久，总体来说多次错误的原因都是因为我对 Python 的语法忘了很多，导致经常写一些常见的错，而这些 bug 又是极其隐蔽的，比如我在将情感的概率归一化的时候，我就直接写了 'posibility[i][j]/=sum(posibility[i][:])' 导致前后的 'sum(posibility[i][:])'不一样，诸如此类的错误还是有很多，还有就是测试集和训练集的文本格式不一样也导致了我 de 了很久，这个实验让我了解了什么是 KNN 算法，就是找 K 个样本中的众数算法，这个算法实现起来还是比较简单的，训练速度还是比较快的，但是预测速度是相当慢的，导致我每次跑结果都会跑很久。