中山大學
SUN YAT-SEN UNIVERSITY

《人工智能实验》

实验报告

（实验一）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 计算机科学与技术
学 生 姓 名 ：	仲逊
学　　号	16327143
时　　间 ：	2019　年　9　月　5　日

实 验 1 ：文本数据集简单处理&KNN

一. 实验目的

1. 掌握文本数据集的简单处理：TF-IDF 矩阵
2. 在给定数据集上使用 KNN 分类训练模型，预测文本情感
3. 在给定数据集上使用 KNN 回归训练模型，预测文本的各个情感占比

二. 算法原理

1.TF-IDF矩阵的处理：

TF-IDF矩阵是一种较为简易的文本处理方法，建立过程大致是one-hot矩阵->TF矩阵->TFIDF。首先矩阵使用n篇文章中的单词建立一个词典，对于每篇文章建立一个特征向量，如果该单词在一篇文章中出现，则该位置标为1，其余标为0，这就是one-hot矩阵；将one-hot矩阵为1的位置更改为该单词在这篇文章出现的频率就得到TF矩阵；对于词典中每个单词按照如下公式计算其逆向文件频率IDF：

$$idf_i = log\frac{|D|}{|\{j: t_i \in d_j\}|}$$

然后将TF矩阵中不为0的项乘上该单词的IDF就得到了IDF矩阵。

根据公示也可以看出它基于的主要思想是：对区别文档最有意义的词语应该是那些在文档中出现频率高，而在整个文档集合的其他文档中出现频率少的词语。

2.KNN分类算法：

KNN是比较常见的无监督学习方法，大致思想就是遇到一个新样本就找距离它最近的K个已经被打上标签的样本，看哪个标签最多就将其分到此类。其中的细节问题有：“距离”它最近这个距离用什么衡量，可以选择p值不同的各种L_p范数：

$$Lp(\vec{X},\vec{Y}) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{\frac{1}{p}}$$

或者余弦距离：

$$dist(\vec{X},\vec{Y}) = 1 - cos(\vec{X},\vec{Y}) = 1 - \frac{\vec{X}\cdot\vec{Y}}{|\vec{X}||\vec{Y}|}$$

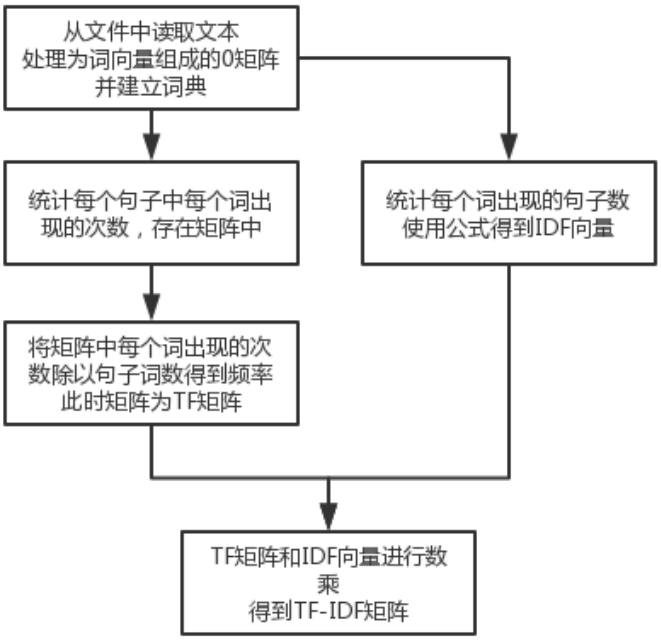
以及选择什么样的K值能够取得比较好的结果。

3.KNN回归算法：

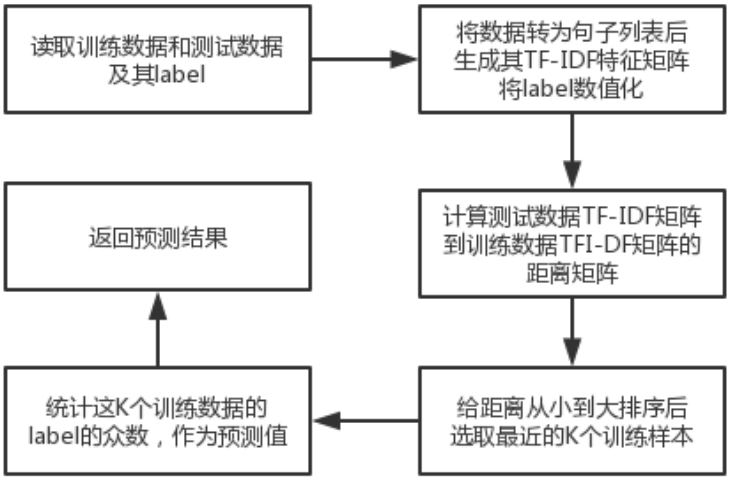
KNN也可以用来做回归，和分类不同的地方是已知样本的标签是连续的属性值，思想与分类大同小异，还是遇到一个新样本就找距离它最近的K个已经知道其属性值的样本，只需使用到这K个样本的距离倒数对各个属性值加权，即可预测新样本的属性值，其中的细节问题与分类相同。

三. 流程图

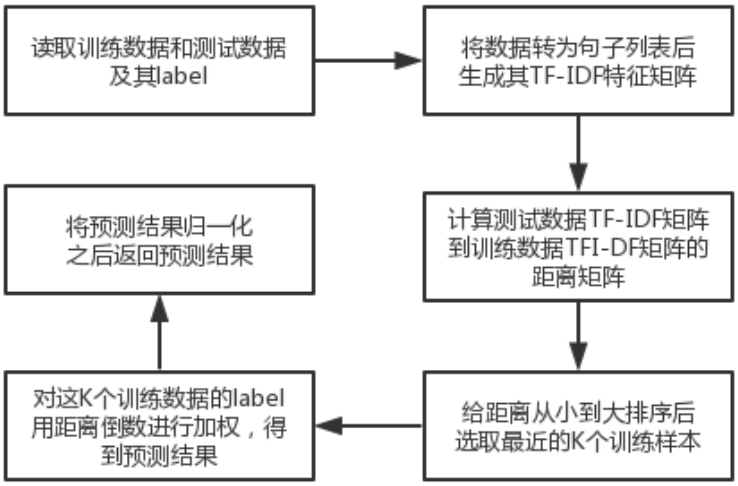
- 生成TF-IDF矩阵



- KNN分类



- KNN回归



四. 代码截图

- 生成TF-IDF矩阵

```
def get_tfidf(senten_list,words_list):
    '''
    根据给定词典和文章建立 tfidf 矩阵
    input: senten_list 句子列表 每个句子是由 word 构成的列表
           words_list 由不重复单词组成的列表
    return: tfidf 矩阵，类型为 pandas 的 dataframe
    '''
    vocabulary=set(words_list)
    samples_num=len(senten_list)     #样本数
    idf=[]                           #逆向文本频率
    #生成维度为 sample 数×词汇表长度的 0 矩阵,
    tfidf=pd.DataFrame(np.zeros((samples_num,len(words_list)),dtype=np.int),
                       columns=words_list)
    #现在的矩阵元素表示对应单词在该 sample 中出现了多少次
    for i in range(len(senten_list)):
        for word in senten_list[i]:
            if word in vocabulary:
                tfidf[word][i]+=1
    #计算每个单词对应的 idf
    for word in words_list:
        idf.append(np.log10(samples_num/(len([1 for x in tfidf[word].values
                   if x>0])+1)))
    #计算频率，现在的矩阵元素表示对应的词语出现的频率，即 tf 矩阵
    for i in range(samples_num):
```

```
        tfidf.loc[i]/=tfidf.loc[i].values.sum()
    #将矩阵每项x对应的idf 得到tf-idf 矩阵
    for i in range(len(words_list)):
        tfidf[words_list[i]]*=idf[i]

    return tfidf
```

■ 处理TF-IDF矩阵的文件读写：

```
if __name__ == '__main__':
    senten_list=[]                          #句子列表
    semeval=open("D:/AI Lab/lab1/lab1_data/semeval.txt","r")
    for line in semeval.readlines():
        #去除换行符后，以"\t"为间隔划分
        tem=line.strip().split("\t")
        #从句子中建立词汇表
        sentence=tem[2].split(" ")
        senten_list.append(sentence.copy())
    semeval.close()
    #建立词典
    words_list=get_words_list(senten_list)
    #获取 tfidf 矩阵
    tfidf=get_tfidf(senten_list,words_list)
    np.savetxt("16327143_zhongxun_TFIDF.txt",np.mat(tfidf.values),fmt="%.6f")
```

■ 计算距离矩阵（以计算Lp范数为例）：

```
def compute_distances(X_train, X_test, p):
    """
    input: X_train 训练集 numpy 矩阵
           X_test 测试集 numpy 矩阵
           p Lp 范数 整数
           根据给定训练集和测试集，计算距离矩阵
    return: 距离矩阵，mat[i][j]表示第 i 个验证样本到第 j 个训练样本的距离
    """
    test_num=X_test.shape[0]
    dists=[]
    for i in range(test_num):
        dist=np.power(np.sum(np.power(np.abs(X_train-X_test[i]),p),axis=1),1/p)
        dists.append(dist)

    return np.array(dists)
```

- KNN分类：

```
def KNN_classification(X_train,y_train,X_test,y_test,k,p):
    """
    input: X_train 训练集 numpy 矩阵  y_train 训练标签 numpy 矩阵
           X_test 测试集 numpy 矩阵   y_test 测试集标签 numpy 矩阵
           k KNN 的参数 整数
           p Lp 范数 整数
    return: y_predict 预测结果 numpy 矩阵
    """
    dists=compute_distances(X_train, X_test, p)
    sorted_mat=y_train[np.argsort(dists)]        #根据距离排好序的训练集标签
    y_predict=[]
    for sorted_array in sorted_mat:
        #获取最近的K 个训练样本标签的众数
        y_predict.append(np.argmax(np.bincount(sorted_array[:k])))
    #统计正确率
    right_pred=0
    for i in range(len(y_predict)):
        if y_predict[i]==y_test[i]:
            right_pred+=1
    print("accuracy = ",right_pred/len(y_predict))

    return np.array(y_predict)
```

- KNN回归：

```
def KNN_regression(X_train,y_train,X_valid,y_valid,k,p):
    """
    input: X_train 训练集 numpy 矩阵  y_train 训练标签 numpy 矩阵
           X_test 测试集 numpy 矩阵  y_test 测试集标签 numpy 矩阵
           k KNN 的参数 整数
           p Lp 范数 整数
    return: y_predict 预测结果 numpy 矩阵
    """
    dists=compute_cos_distances(X_train, X_valid)
    sorted_dists=np.sort(dists)                    #排好序的距离
    sorted_mat=y_train[np.argsort(dists)]          #根据距离排好序的训练集标签
    y_predict=[]
    for m in range(sorted_mat.shape[0]):
        sorted_array=sorted_mat[m]
        #各个情感的概率，用列表存储
        p=[]
        for j in range(sorted_array.shape[1]):
            #使用距离倒数对最近K 个样例的情感指标进行加权
```

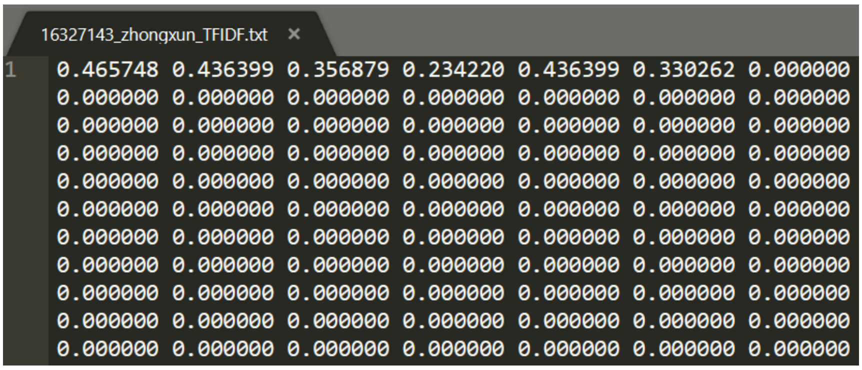
```
            p_of_motion=0
            for i in range(k):
                p_of_motion+=sorted_array[i][j]/sorted_dists[m][i]
            p.append(p_of_motion)
        y_predict.append(p.copy())
    #归一化处理
    y_predict=normalize(np.array(y_predict))
    #计算相关系数的平均值
    sum_corrcoef=0
    for i in range(y_valid.shape[1]):
        sum_corrcoef+=np.corrcoef(y_valid[:,i],y_predict[:,i])
    print("corrcoef:",(sum_corrcoef/y_valid.shape[1])[0][1])

    return y_predict
```

五. 实验结果及分析

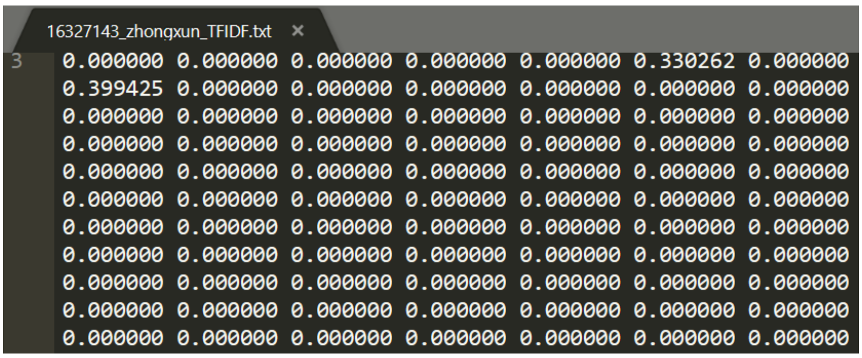
■ 结果展示

1.计算TF-IDF矩阵

tfidf.py文件读取semeval.txt文件内容建立词典，计算得到TF-IDF矩阵（本处计算使用的是分母加1的公式）后写入txt文件中，得到符合预期的TF-IDF矩阵：


16327143_zhongxun_TFIDF.txt ×

```
1  0.465748 0.436399 0.356879 0.234220 0.436399 0.330262 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```


16327143_zhongxun_TFIDF.txt ×

```
3  0.000000 0.000000 0.000000 0.000000 0.000000 0.330262 0.000000
   0.399425 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
   0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```

经过对几组比较简单的语句计算TF-IDF矩阵，可以验证得到的矩阵是正确的。

2.使用KNN分类预测给定数据集的情感类型（都使用TF-IDF矩阵作为特征）

对测试集文本进行分类写入文件的结果如图所示：

	A	B	C	D	E	F
1	textid	label				
2	1	surprise				
3	2	joy				
4	3	joy				
5	4	joy				
6	5	sad				
7	6	sad				
8	7	surprise				
9	8	sad				
10	9	surprise				
11	10	joy				
12	11	fear				
13	12	surprise				
14	13	fear				
15	14	joy				
16	15	fear				
17	16	fear				
18	17	joy				
19	18	sad				
20	19	joy				
21	20	fear				
22	21	joy				
23	22	joy				

16327143_zhongxun_KNN_classific

可以看出label都是是6种情感中的一个，且数据分布合理

3.使用KNN回归预测给定数据集的情感概率

对测试集进行六种情绪的概率预测结果写入文件后如图所示：

	A	B	C	D	E	F	G	H
1	textid	anger	disgust	fear	joy	sad	surprise	
2	1	0.139991	0.221013	0	0.066682	0.215762	0.356552	
3	2	0.123485	0.177025	0.044513	0.170226	0.305604	0.179147	
4	3	0	0	0.021278	0.414431	0.172068	0.392224	
5	4	0.02441	0.030462	0.059089	0.416206	0.095989	0.373843	
6	5	0.097638	0.093057	0.038563	0.289011	0.207263	0.274469	
7	6	0.099811	0.040211	0.267993	0	0.485899	0.106086	
8	7	0.000114	0	0.09296	0.44319	0.00084	0.462896	
9	8	0.089593	0	0.320892	0.075424	0.4251	0.088991	
10	9	0	0.11373	0	0.138184	0	0.748086	
11	10	0.107582	0.129895	0.191452	0.218853	0.101973	0.250246	
12	11	0.139904	0.036671	0.305974	0.156108	0.213779	0.147564	
13	12	0.056207	0	0.117347	0.315599	0.113289	0.397557	
14	13	0.120994	0.070942	0.313206	0.182412	0.187382	0.125063	
15	14	0.117562	0.050261	0.156118	0.375604	0.161036	0.139419	
16	15	0.166181	0.134432	0.419425	0.018965	0.212924	0.048072	
17	16	0.157449	0.089261	0.36984	0.112234	0.124353	0.146863	
18	17	0.06262	0.114834	0.071105	0.397343	0.166548	0.18755	
19	18	0.100083	0.017308	0.141943	0.054915	0.256186	0.429564	
20	19	0.031372	0.008587	0.138168	0.407712	0.148118	0.266043	
21	20	0.10406	0.06943	0.147403	0.380262	0.129779	0.169066	
22	21	0.168201	0.043818	0.173978	0.292136	0.148492	0.173375	
23	22	0.063658	0.037495	0.065496	0.367706	0.076305	0.389339	

16327143_zhongxun_KNN_regressio

稍加验算可以看出每个样例各个情感的概率之和为1，带入验证csv文件得到的相关系数和程序中得到的也相同

■ 模型性能展示和分析

在本次实验中，对预测准确率高低的影响因素主要有两个：一个是距离度量方式，另一个是KNN最近邻算法中参数K的选择。距离度量方式我尝试了p取1到4的Lp范数和余弦距离，与准确率的关系如下表所示，由于p≥3时效果都比较差，不在此列出：

KNN分类准确度

K	欧式距离（p=2）	曼哈顿距离（p=1）	余弦距离
1	0.321543	0.395498	0.437299
2	0.308681	0.424437	0.434084
3	0.263665	0.408360	0.421222
4	0.257234	0.385852	0.437299
5	0.234726	0.395498	0.4212219
6	0.218650	0.395498	0.430868
7	0.202572	0.385852	0.437299
8	0.180064	0.372990	0.453376
9	0.183280	0.382637	0.414791
10	0.170418	0.372990	0.446945
11	0.170418	0.366559	0.434083
12	0.189710	0.376206	0.411575
13	0.196141	0.405145	0.427652
14	0.215434	0.405145	0.405144
15	0.205788	0.408360	0.395498
16	0.196141	0.398713	0.414791
17	0.215434	0.405145	0.405145
18	0.212218	0.405145	0.392283
19	0.225080	0.392283	0.389067

一开始我选择了非常常用的欧氏距离，可以看到效果很差，最高的只有K=1时的0.32，后来开始调整p值，最终发现曼哈顿距离的效果较好（K=2,0.42），其余范数效果都比较差，最后选择的是余弦距离，其实按照直觉衡量两个向量的接近程度最好的就是向量夹角，经过尝试后发现的确如此，余弦距离的效果比之前所有的距离都好，最好的是K=8时，验证集的预测准确率达到了0.45。

在使用KNN回归预测给定数据集情绪概率时也同样如此，由于欧式距离的效果比较差，故没有使用，下表是使用曼哈顿距离和余弦距离作为衡量指标的回归相关系数。

KNN回归相关系数

K	曼哈顿距离（p=1）	余弦距离
1	0.289196	0.339308
2	0.329804	0.366629

3	0.315956	0.374948
4	0.320302	0.390761
5	0.316668	0.395884
6	0.319548	0.390939
7	0.322671	0.392865
8	0.305133	0.392551
9	0.309216	0.391216
10	0.309054	0.395429
11	0.309558	0.383520
12	0.321650	0.381800
13	0.326924	0.370842
14	0.336819	0.358545
15	0.329251	0.358500
16	0.327055	0.356859
17	0.326773	0.351595
18	0.320606	0.344807
19	0.323033	0.345304

同样的，也是余弦距离更胜一筹，最高是K=5时，在验证集上的预测结果的相关系数平均值达到了0.395884

六. 思考题

1. IDF的第二个计算公式中分母多了个1是为什么？

如果只用训练集中出现的单词建立词典，测试集中可能出现训练集中没有的单词，这时计算逆向文件频率的分母就为0，产生除0错误，所以用分母加1来解决这个问题。

2. IDF数值有什么含义？TF-IDF数值有什么含义？

IDF是总文章数除以该单词出现的文章数再取对数得到的结果，从公式中可以看出，IDF越小说明这个词在词典中越常见。

而TF-IDF矩阵的思想是：对区别文档最有意义的词语应该是那些在文档中出现频率高，而在整个文档集合的其他文档中出现频率少的词语，因此如果一个词在TF-IDF矩阵的值较大，则可以认为它对于分类比较重要。

3. 为什么使用距离的倒数作为权重？

两个文本向量的距离越小，则说明它们蕴含的情感越相近，就应该给它比较大的情感权重，用距离的倒数作为权重可以使得距离越小，情感权重越大。

4. 同一测试样本的各个情感概率总和应该为1，如何处理？

对按照距离倒数加权求和得到的情感数值进行归一化处理，将每个情感概率除以该样本中各情感概率的总和，这样得到的数值就可以保证各个情感概率总和为1。

七. 参考文献

1) https://docs.scipy.org/doc/numpy/

2) https://pandas.pydata.org/pandas-docs/stable/