

人工智能实验报告（一）

- 院系/年级专业：数据科学与计算机学院/18级计科
- 学号/姓名：18340236/朱煜

一、实验题目

文本数据处理与KNN

二、实验原理

1、文本数据处理

(1) One-hot编码

- 根据所有文档建好词库，每个词对应二进制向量的一个维度。每个文档使用一个二进制向量来表示，当文档中有词库中的词时，对应的向量维度为 1，否则为 0。

(2) 词频表示

- **词频 (term frequency, TF)** 指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化(一般是词频除以文章总词数), 以防止它偏向长的文件。
- 根据所有文档建好词库，每个词对应向量的一个维度。每个文档使用一个向量来表示，其中每一个向量维度的值表示该词在文档中出现的次数。

(3) 词频归一化后的概率表示

- 根据所有文档建好词库，每个词对应向量的一个维度。每个文档使用一个向量来表示，其中每一个向量维度的值表示该词在文档中出现的次数与文档中词语总数的比值，表示如下

$$tf_{i,d} = \frac{n_{i,d}}{\sum_v n_{v,d}}$$

(4) 逆向文档频率

- 逆向文档频率使用文档的总数与词出现的文档数的比值衡量该词的普遍性，假设一共有 C 篇文档，第 i 个词一共在 C_i 篇文档中出现，表示如下

$$idf_i = \log \frac{C}{C_i}$$

- 为了防止出现分母为0的情况，通常我们将分式中的分母使用 $C_i + 1$ 代替

(5) TF-IDF

- **TF-IDF (Term Frequency-Inverse Document Frequency, 词频-逆文件频率)** 用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。表示为词频表示与逆向文档频率的乘积，如下

$$tf-idf_{i,j} = \frac{n_{i,j}}{\sum_{k=1}^V n_{i,k}} \times \lg \frac{C}{C_i + 1}$$

2、KNN算法

- **KNN (K Near Neighbor)** k个最近的邻居，即每个样本都可以用它最接近的k个邻居来代表。
- 在KNN算法中，常使用向量来表示各个样本，并使用距离来衡量两个样本之间的差距，差距与距离成正比。一个样本若与数据集中的k个样本最相似，在KNN分类中，采用多数投票的原则确定该样本的标签，在KNN回归中，根据k个样本的标签概率进行推算得到该样本的标签概率。
- 距离度量：在KNN算法中，常使用Lp距离与余弦相似度来作为距离度量。

Lp距离：

$$L_p(x_i, x_j) = \left(\sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|^p \right)^{\frac{1}{p}}$$

其中当 $p = 1$ 时，为曼哈顿距离：

$$L_1(x_i, x_j) = \sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|$$

$p = 2$ 时，为欧氏距离：

$$L_2(x_i, x_j) = \left(\sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|^2 \right)^{\frac{1}{2}}$$

余弦相似度：

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \cdot B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

- KNN分类伪代码如下

```
KNN_classification(Train_set, Test_set)
Input: Train_set = [V:Vector set of data to train, L:Label set of data to train],
       Test_set = [V:Vector set of data to predict]
Output: For all vector V_Test in Test_set, predict its label.

Result = {}
for all V_Test in Test_set do
    distance_set = {}
    for all V_Train in Train_set do
        distance = CalculateDistance(V_Test, V_Train)
        distance_set += distance
```

```

labels_set = {}
for i in range(0, BestK) do
    minIndex = argmin(distance_set)
    labels_set += Train_set.L[minIndex]
    deletemin(distance_set)
predict_label = LookForTheMostLabel(labels_set)
Result += predict_label
return Result

```

- KNN回归伪代码如下

```

KNN_regression(Train_set, Test_set)
Input: Train_set = [V:Vector set of data to train, L_P:Label and
Probability set of data to train],
       Test_set = [V:Vector set of data to predict]
Output: For all vector V_Test in Test_set, predict its probability of
labels.

Result = {}
for all V_Test in Test_set do
    distance_set = {}
    for all V_Train in Train_set do
        distance = CalculateDistance(V_Test, V_Train)
        distance_set += distance
    probability_set = {}
    for i in range(0, BestK) do
        minIndex = argmin(distance_set)
        labels_set += Train_set.L_P[minIndex]
        deletemin(distance_set)
    predict_probability = CalculateProbability(probability_set)
    Result += predict_probability
return Result

```

其中关于 `CalculateProbability` 的计算，每个标签的概率根据Test与Train中选出的K个样本的距离作为权重，再乘以相对应Train样本该标签的概率，计算公式如下

$$P(label_i) = \sum_{j=1}^n \frac{P(label_i \text{ in } train_j)}{distance(train_i, test)}$$

3、相关系数

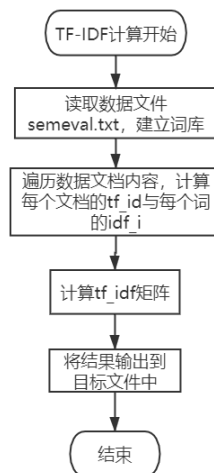
- 相关系数是研究变量之间线性相关程度的量，在KNN算法中用以衡量KNN回归结果与实际结果的相关性，以确定相关参数的有效性，最终结果介于-1到1之间。其结果为两个变量之间的协方差与标准差的商，Pearson相关系数的公式如下

$$COR(X, Y) = \frac{cov(X, Y)}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2 \sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

三、实验过程

1、实验任务1

(1) 实验流程图



(2) 关键代码

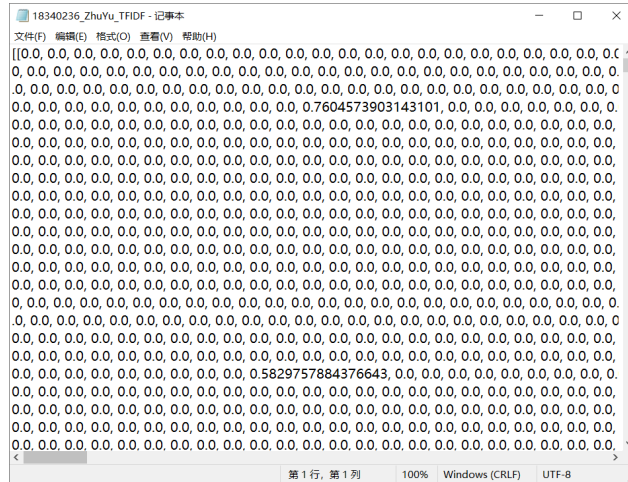
- 根据流程图，依次计算文档的归一化频率与各词的逆向文档频率，最终将两者相乘得到tf-idf矩阵，代码如下

```
#计算tf-idf矩阵
def tf_idf(sentence):
    count_dict = dict()
    word_set = set()
    word_tf = []
    #建立词库再按字典序进行排序
    for words in sentence:
        for word in words:
            word_set.add(word)
    word_set = sorted(word_set)
    for words in sentence:
        word_count = []
        for word in word_set:
            #防止文档为空，出现除0情况
            if len(words) == 0:
                word_count.append(words.count(word))
            #归一化的频率表示
            else:
                word_count.append((words.count(word))/len(words))
        #使用字典统计每个词出现的文档数
        if words.count(word) > 0:
            if word in count_dict:
                count_dict[word] +=1
            else:
                count_dict[word] = 1
        word_tf.append(word_count)
    #计算逆向文档频率
    for word in count_dict:
        count_dict[word] = math.log(len(sentence)/(count_dict[word]+1))
    #计算tf-idf矩阵
    for document in word_tf:
        for index in range(len(document)):
            document[index] = document[index] * count_dict[word_set[index]]
```

```
return word_tf
```

(3) 实验结果

- 运行 `TF-IDF.py`，将TF-IDF矩阵输出到目标文件，结果如下，生成的TF-IDF矩阵为稀疏矩阵，由于词频归一化，稀疏矩阵的值大多小于1。

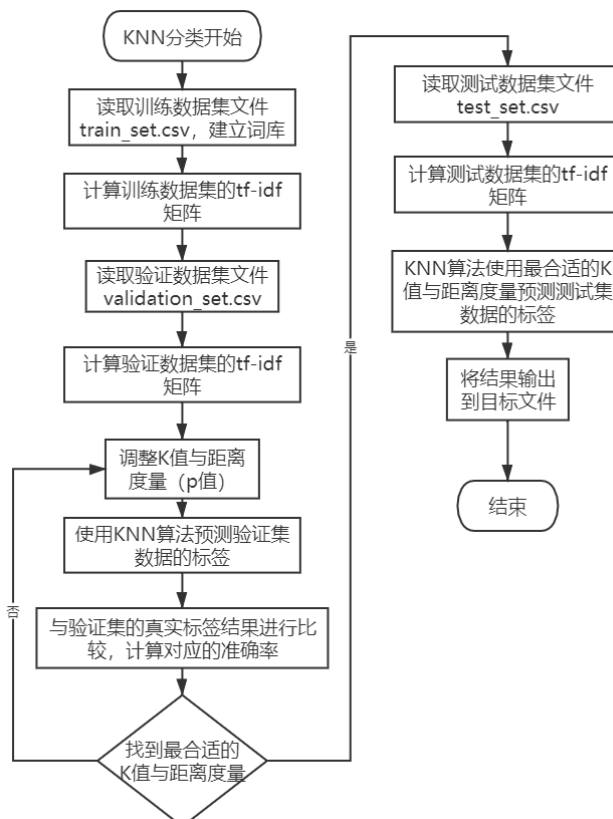


(4) 实验分析

- 由结果可知，由于词库较大而文档的词数量较少，生成的TF-IDF矩阵为稀疏矩阵，而其结果相比于One-hot编码，相应词的数值不止简单的0或1，而会根据该词的重要性进行比重的调节，出现频率高的词代表重要性较低，对应的数值较小；频率低的词代表重要性较高，对应数值较高，这充分体现了TF-IDF相较于简单的One-hot编码具有更高的代表性。

2、实验任务2

(1) 实验流程图



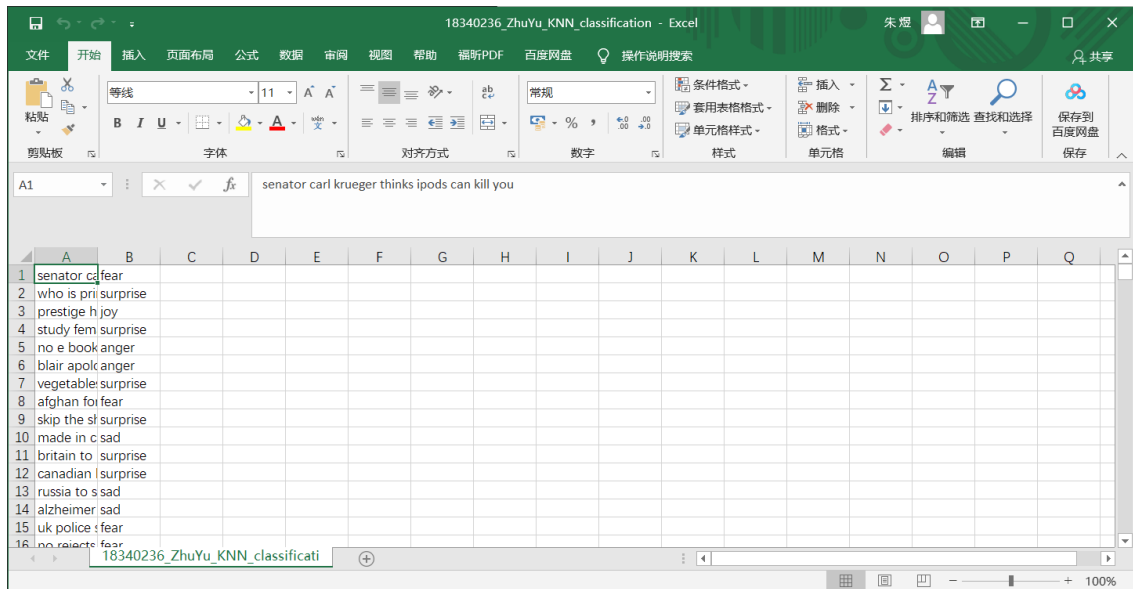
(2) 关键代码

- 根据流程图，建立验证集与训练集的tf-idf矩阵后，通过调整K值，使用KNN算法得到标签后计算准确度以选出最合适的K值，在计算距离的时候采用了曼哈顿距离，而选取的K从1到40，代码如下

```
def FindBestK(KNN_TR, KNN_VA):
    P = 1
    K = int(40)
    #采用Lp距离计算验证集各例与验证集的距离
    distance_set = []
    for VA in KNN_VA:
        distance = []
        for TR in KNN_TR:
            #距离公式
            SUM = np.sum(
                np.power(np.abs(np.array([TR[0]])-np.array([VA[0]])), P))
            distance.append(np.power(SUM, 1/P))
        distance_set.append(distance)
    #KNN算法
    Result = []
    for d in distance_set:
        index_sort = np.argsort(d)
        R_K = []
        for j in range(1, K):
            #筛选出距离最小的K个例子对应的标签
            top_k = [KNN_TR[i][1] for i in index_sort[:j]]
            #采用多数投票输出最多的标签作为结果
            R_K.append(max(top_k, key=top_k.count))
        Result.append(R_K)
    #计算每个K值对应的准确度
    AC = []
    for i in range(0, K-1):
        #计算正确个数
        count = 0
        for j in range(0, len(Result)):
            if Result[j][i] == KNN_VA[j][1]:
                count += 1
        AC.append(count/len(Result))
    #对准确度进行排序后输出效果最好的K
    K_best = np.argsort(AC)[-1]+1
    print(AC)
    return K_best
```

(3) 实验结果

- 运行 `KNN_classification.py`，将结果输出到目标文件，结果如下，可见分类结果标签大致平均分布在各个情绪上，结果较为合理



- 终端运行的结果如下，由输出的结果可知，输出结果的正确率大致在30%以上，而当K=17时达到正确率的最高值，大致为41.8%，由于验证集一共有312个样例，而17与 $\sqrt{312}$ 较为接近，最终生成的最佳K值也在合适的范围内。

K	正确率
1	0.395498
2	0.395498
3	0.389068
4	0.372990
5	0.379421
6	0.385852
7	0.382637
8	0.369775
9	0.389068
10	0.395498
11	0.382637
12	0.385852
13	0.405145
14	0.411576
15	0.408360
16	0.392283
17	0.418006
18	0.401929
19	0.398714
20	0.398714
21	0.385852
22	0.389068
23	0.372990
24	0.360129
25	0.363344
26	0.366559
27	0.379421
28	0.389068
29	0.372990
30	0.379421
31	0.376206
32	0.366559
33	0.360129
34	0.344051
35	0.356913
36	0.372990
37	0.379421
38	0.360129
39	0.363344

最好的K值为17

(4) 实验分析

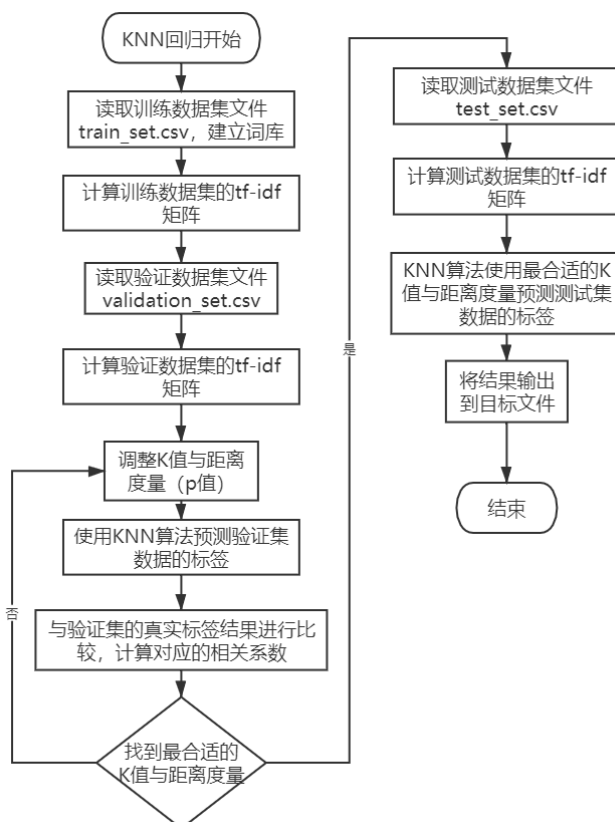
- 实验中距离度量的选取是通过手动调参的方式，根据结果发现曼哈顿距离的效果较好，其最好的K值正确率可达40%；而欧氏距离的正确率较差，正确率具体如下，可知最好的K值为1，这会导致过拟合，因此在进行分类时，使用了曼哈顿距离作为距离度量。

K	正确率
1	0.321543
2	0.321543
3	0.263666
4	0.250804
5	0.250804
6	0.221865
7	0.199357
8	0.192926
9	0.186495
10	0.180064
11	0.180064
12	0.199357
13	0.196141
14	0.209003
15	0.205788
16	0.189711
17	0.212219
18	0.215434
19	0.225080
20	0.228296
21	0.260450
22	0.263666
23	0.247588
24	0.218650
25	0.221865
26	0.225080
27	0.247588
28	0.247588
29	0.231511
30	0.218650
31	0.218650
32	0.237942
33	0.228296
34	0.231511
35	0.221865
36	0.202572
37	0.212219
38	0.221865
39	0.231511
最好的K值为1	

- 欧式距离在本次实验中效果较差，主要原因是生成的TF-IDF矩阵为稀疏矩阵，向量之间的距离差异主要在于所含词的不同，而矩阵中的数值大多为小于1的数，只有少部分出现次数较少的词会大于1，因此在欧氏距离时，进行平方后距离的差距被缩小，各词的代表性减弱，因此并没有曼哈顿距离的效果明显。

3、实验任务3

(1) 实验流程图



(2) 关键代码

- 根据流程图，建立验证集与训练集的tf-idf矩阵后，调整K值，使用KNN算法得到标签后计算相关系数以选出最合适的K值，在计算时采用了曼哈顿距离，代码如下

```
def FindBestK(KNN_TR, KNN_VA):
    P = 1
    K = int(20)
    #采用Lp距离计算验证集各例与验证集的距离
    distance_set = []
    for VA in KNN_VA:
        distance = []
        for TR in KNN_TR:
            #距离公式
            SUM = np.sum(
                np.power(np.abs(np.array([TR[0]])-np.array([VA[0]])), P))
            distance.append(np.power(SUM, 1/P))
        distance_set.append(distance)
    #筛选出距离最小的K个例子
    Result = []
    for d in distance_set:
        d_sort = np.sort(d)
        index_sort = np.argsort(d)
        R_K = []
        for j in range(1, K):
            #筛选出距离最小的K个例子对应的标签概率并除以距离
            top_k = [(np.array(KNN_TR[i][1])/(d_sort[i])).tolist()
                      for i in index_sort[:j]]
            Sum_k = np.array(top_k[0])
            #根据回归公式对得到的top_k个标签概率进行求和
            for n in range(1, len(top_k)):
                Sum_k += np.array(top_k[n])
            #对得到的结果进行归一化
            temp = np.sum(Sum_k)
            for pi in range(0, len(Sum_k)):
                Sum_k[pi] /= temp
            R_K.append(Sum_k.tolist())
        Result.append(R_K)
    #计算每个K值对应的相关系数
    corr_factor = []
    for i in range(0, K-1):
        corr_k = []
        #计算6个标签的相关系数
        for j in range(0, 6):
            alist = [row[i][j] for row in Result]
            blist = [row[1][j] for row in KNN_VA]
            #平均值
            a_avg = sum(alist)/len(alist)
            b_avg = sum(blist)/len(blist)
            #协方差
            cov_ab = np.sum(np.multiply(np.array(alist)-
a_avg,np.array(blist)-b_avg))
            #方差
            d_a = math.sqrt(np.sum(np.power(np.array(alist)-a_avg,2)))
            d_b = math.sqrt(np.sum(np.power(np.array(blist)-b_avg,2)))
            #协方差与方差之比
            corr_k.append(cov_ab/(d_a*d_b))
```

```

#最后结果取平均
corr_factor.append(sum(corr_k)/len(corr_k))

#对准确度进行排序后输出效果最好的K
K_best = np.argsort(corr_factor)[-1]+1
print('K      相关系数')
for i in range(len(corr_factor)):
    print("%-5d%f" %(i+1,corr_factor[i]))
return K_best

```

(3) 实验结果

- 运行 KNN_regression.py，将结果输出到目标文件，结果如下，由于进行了归一化操作，回归出的概率结果和为1，经过与分类结果比较，差距较大，这是由于选择的最优值的K为2，导致过拟合，其K值需要进一步修改

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	senator carl krueger thinks ipods can kill you	0.060702	0.151999	0.303623	0.034272	0.089041	0.360364										
2	who is president	0.129699	0.024705	0.08029	0.065703	0.389356	0.310247										
3	prestige high	0	0	0.025506	0.23569	0.395222	0.343581										
4	study female	0	0	0	0.709959	0	0.290041										
5	no e book	0.10228	0.068784	0.08733	0.291064	0.089053	0.361489										
6	blair apolc	0.071532	0.088469	0.265041	0	0.412346	0.162612										
7	vegetable	0	0	0.168831	0.357754	0.00963	0.463785										
8	afghan fo	0.152134	0	0.400468	0.068807	0.300655	0.077937										
9	skip the sl	0	0.145987	0	0.134259	0	0.719754										
10	made in c	0.015477	0.1366	0.185578	0.249793	0.054121	0.358432										
11	britain to	0.200898	0	0.041897	0.387405	0.111746	0.258054										
12	canadian	0.135498	0.002828	0.292228	0.099259	0.242368	0.227819										
13	russia to s	0.197142	0.122864	0.385683	0.048584	0.09717	0.148556										
14	alzheimer	0.105582	0.128351	0.118006	0.270269	0.120035	0.257757										
15	uk police	0.222272	0.205709	0.281869	0	0.290149	0										
16	no reject	0.153691	0.035341	0.377691	0	0.163885	0.274382										

- 终端的运行结果如下，当K=2时，相关系数突增到0.31左右，主要原因是因为验证集的特殊性，因此该结果存在一定偶然性，而当K=14时，相关系数为0.30，为第二大的相关系数，因此对结果进行稍微修改

```

K      相关系数
1      0.289196
2      0.313170
3      0.290017
4      0.288203
5      0.282094
6      0.286471
7      0.286804
8      0.266038
9      0.270153
10     0.271128
11     0.270952
12     0.283458
13     0.290096
14     0.300278
15     0.291206
16     0.289428
17     0.288269
18     0.281010
19     0.282202
最好的K值为2

```

- 选择K=14作为新的K值，新结果如下，

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	senator cz	0.128595	0.122399	0.179191	0.07605	0.190913	0.302851										
2	who is pri	0.054892	0.086588	0.163265	0.197177	0.160006	0.338071										
3	prestige h	0.050556	0.020309	0.144392	0.30006	0.141116	0.343568										
4	study fem	0.083546	0.121933	0.160012	0.206231	0.062904	0.365375										
5	no e book	0.097644	0.091484	0.170873	0.21595	0.219326	0.204724										
6	blair apok	0.215922	0.052408	0.254842	0.003027	0.358005	0.115795										
7	vegetable	0.032726	0.020672	0.178733	0.326194	0.140521	0.301154										
8	afghan fo	0.13747	0.063639	0.28321	0.081552	0.238611	0.195519										
9	skip the sl	0.073209	0.137793	0.158015	0.122914	0.11744	0.390629										
10	made in c	0.067277	0.044591	0.191908	0.252383	0.24546	0.19838										
11	britain to	0.112988	0.02089	0.210025	0.31523	0.107023	0.233845										
12	canadian	0.064482	0.050124	0.224189	0.210316	0.193413	0.257477										
13	russia to s	0.180887	0.039781	0.314507	0.091694	0.178019	0.195112										
14	alzheimer	0.10991	0.06487	0.137682	0.189754	0.202868	0.294915										
15	uk police	0.192125	0.101556	0.397802	0.072622	0.154561	0.081334										
16	no reject	0.140102	0.110965	0.186768	0.14758	0.165377	0.240208										

(4) 实验分析

- 与分类相同，在回归中选择了曼哈顿距离作为距离度量，其原因与分类类似，同样是由生成的TF-IDF矩阵的特征影响到欧氏距离的区分度。
- 在本次实验中，经过计算得到的最好K值为2，其相关系数大致为0.31，而这样的K值选择容易导致过拟合，只有当输入的样例与训练样例相似时才会起到更好的效果。
- 根据运行结果可知，当K=14时，其相关系数大致为0.30，与最优的K值的相关系数差距并不大，这样的K可以减少估计误差，但与输入实例较远的训练实例也会对预测起作用，使预测发生错误。在本次实验中，个人认为K=14更符合最合适的K值。但由于对实际结果不能有效确认，仍需要进一步地验证以确定最好的K值。

四、实验思考

1. 经过实验了解到了KNN算法的基本过程，其算法简单而容易解释，由于没有建立模型的过程，其训练速度相较于其他算法要快，这是KNN算法的优点。
2. 但由于KNN算法在计算上需要花费大量时间，输出分类或回归结果相较于其他算法较久。而且KNN必须存储所有数据点，存储要求高，而其对异常值敏感，准确性受噪声或无关数据的影响，在选取距离度量与K值的时候需要进行多次调参权衡，但结果不尽人意，这是KNN算法的缺点。