中山大学数据科学与计算机学院

计算机科学与技术专业(超算方向) 人工智能

本科生实验报告

2018-2019学年秋季学期

课程名称：Artificial Intelligence

教学班级	计科二班+超算	专业（方向）	超算方向
学号	16337259	姓名	谢江钊

实验题目

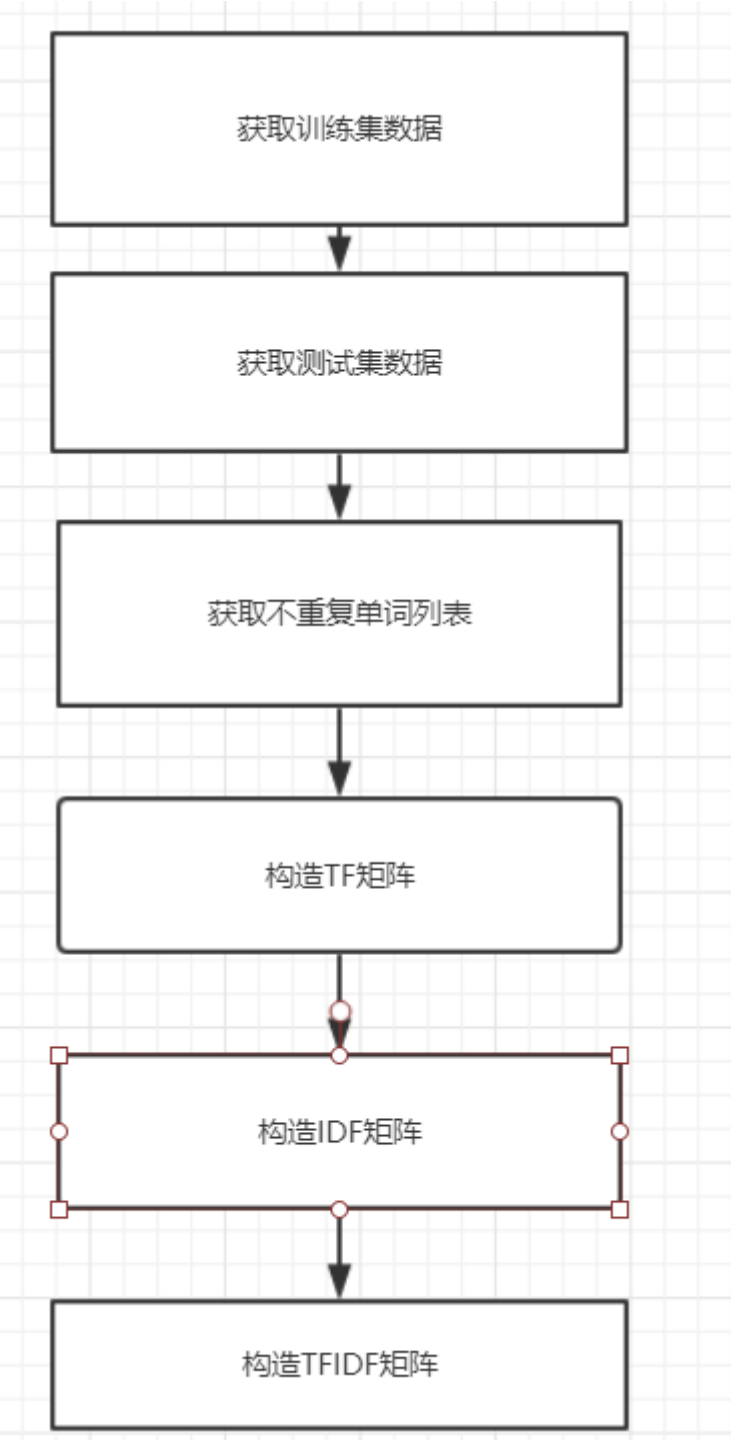
TF-IDF实验任务

将数据集“semeval”的数据表示成 TF-IDF 矩阵

KNN实验任务

实验内容

SemEval实验



这个实验主要是构建TF-IDF矩阵，这个矩阵主要由两个步骤组成：

- 构造TF矩阵
- 构造IDF矩阵

其余很多细节在这里暂时不进行讨论，下面一一进行说明。

文本的分词和OneHot矩阵的构建

文本在处理时是不参与的，我们需要将其转化为数值并且构造相应的向量或者矩阵。对于一个句子，可以将其划分成若干个独立的单词，然后用这些单词构建一个不重复的词向量或者词汇表。

随后，对于每一个句子，我们给其独立分配一个向量，向量大小与不重复的词向量表大小一致，并且位置与词汇一一对应，对于句子中出现的每个单词，出现则对应为1，不出现为0，这样，我们就将句子表达成了一个向量，所有句子都有相同大小的向量来表达，从而便利于后面的处理。这里损失的句子的词序，但是不影响后面的分析。

文本编号	词汇表					
训练文本1	苹果	手机	好用	销售		
训练文本2	市民	买	手机	手机		
训练文本3	市民	觉得	苹果	手机	贵	好用

贵	好用	觉得	买	苹果	市民	手机	销售

	贵	好用	觉得	买	苹果	市民	手机	销售
训练文本1	0	1	0	0	1	0	1	1
训练文本2	0	0	0	1	0	1	1	0
训练文本3	1	1	1	0	1	1	1	0

TF矩阵的构建

TF：向量的每一个值标志对应的词语出现的次数归一化后的频率。

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

TF矩阵的构建，主要是用来给刚才的矩阵做归一化，TF矩阵表达的还是两件事：1.单词是否出现。 2.单词在句子中出现了几次。通过这两个因素，我们能通过出现次数的多少给句子做个大概的定性，判断它们的相似度。

IDF矩阵的构建

IDF：逆向文件频率

$$idf_i = log\frac{|D|}{1+|\{j:t_i \in d_j\}|}$$

这个矩阵主要是为了削减频繁出现的单词的占的权重，避免造成干扰。比如说英语里面一些介词，其实是没有必要的，那判断两个句子是否表达同一个东西就不能将它考虑进去，这是没什么意义的。相反，那些稀有的，仅仅在这个句子单独出现的就说明这个单词在句子中有独特的含义。与之前的TF矩阵结合起来，就能更好地通过向量表达这个句子的定性含义。

TF-IDF矩阵的构建

上面已经说得比较清楚了，具体的做法是:

$$tfidf_{i,j} = tf_{i,j} * idf_i$$

TF-IDF矩阵构建起来后，每一个行向量都代表了一个句子，这个行向量里面数值比较大的元素代表了这个句子的显著特征，例如一些表达情绪，公众人物的词可能会有相对大的值，因为反映了句子的主题。除此以外，不同行向量之间的距离越小，则表明两个句子表达的主题趋向于一致。

实验过程

这一部分实现相对比较简单，但是在实现时，并没有把句子表达成一个完全的向量，而是把它作为了一个字典便于进行检索。对于TF矩阵和IDF矩阵，则是用一个列表把每个句子(字典)保存起来。对于不重复的单词表，同样是表达成一个字典。

对于数据的清洗则不再赘述，在这里说些关键的点：

TF矩阵的构建：

```
tf_mat = []
for data in datas:
    data_row = copy.deepcopy(unique_word) # 每个列表要有自己的字典
    size = len(data)
    word_appear = {} #记录句子中已经出现的单词
    for word in data:
        if word not in word_appear.keys(): #如果句子中还没出现这个单词
            word_appear_page[word] += 1 #单词出现的文章数加一
        word_appear[word] = 1
        data_row[word] += 1 / size #句子对应的字典，出现一次根据TF矩阵加上相应的值
    tf_mat.append(data_row) #将字典加入列表中，加入完毕则TF矩阵构造完成
```

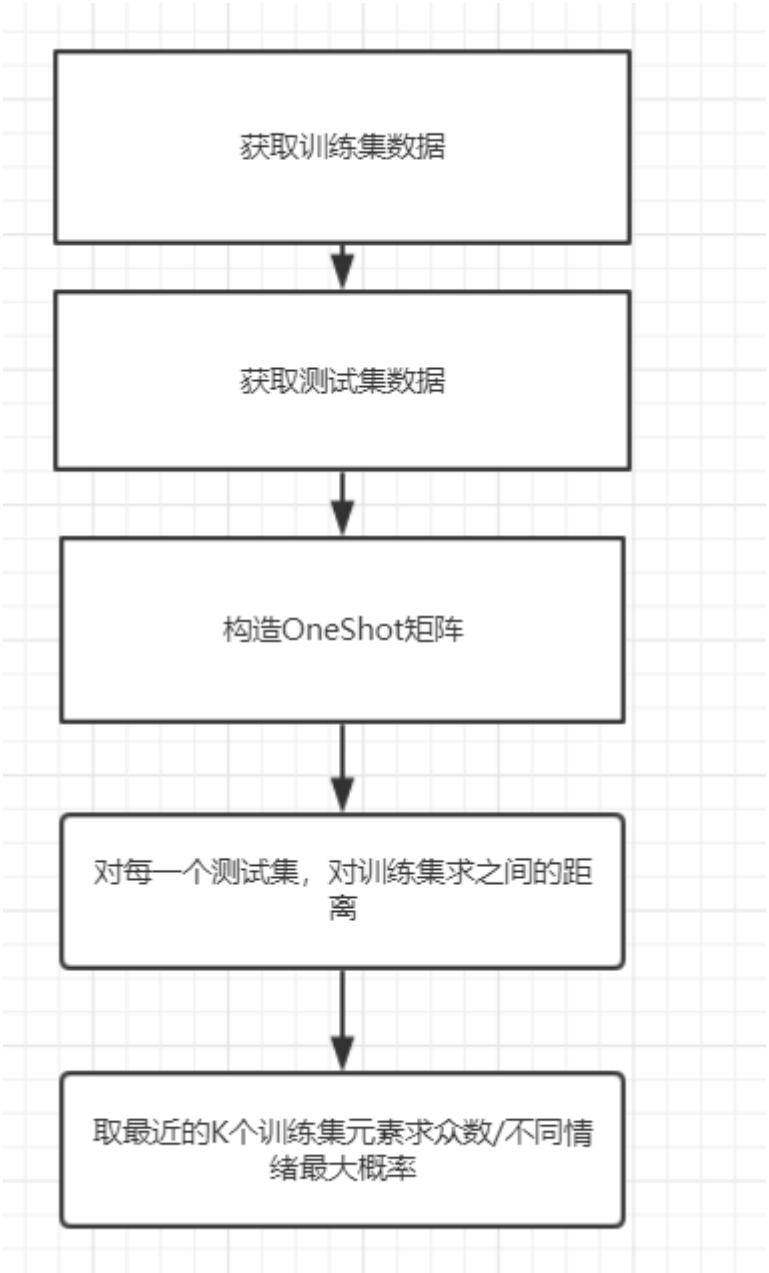
IDF矩阵的构建：

```
idf_mat = copy.deepcopy(unique_word)
for word in idf_mat.keys():
    idf_mat[word] = math.log(sentence_size / (1 + word_appear_page[word])) ##IDF矩阵构建的公式
tfidf_mat = copy.deepcopy(tf_mat)
```

TFIDF矩阵的构建:

```
tfidf_mat = copy.deepcopy(tf_mat)
for data in tfidf_mat:
    for word in data:
        data[word] *= idf_mat[word]
```

KNN分类



KNN分类我看来觉得是比较简单的，主要是

- 确定距离和k
- 取邻居中的情绪众数作为自己的值

在此之前我们已经建立过TF矩阵，往回一步建立One-hot矩阵然后视作相应的向量去计算距离取邻居即可。

距离计算

```
def cal_distance(train_exam, test_exam):
    dis = 0
    for key in train_exam.keys():
        if key != 'emotion':
            if train_exam[key] != test_exam[key]:
                dis += (train_exam[key] - test_exam[key])**2
    dis = math.sqrt(dis)
    return dis
```

这个是欧式距离，如果需要采用别的距离标准，修改这里就可以了。

k值邻居

```
def k_result(k, res):
    res.sort(key=lambda x: x['distance'])   #排序来获取最近的邻居
    emotions = {}
    for i in range(k):  #统计前k个邻居的情绪情况
        if res[i]['emotion'] in emotions:
            emotions[res[i]['emotion']] += 1
        else:
            emotions[res[i]['emotion']] = 1
    items = sorted(emotions.items(), key=lambda item: item[1]) #对这些情绪排序，取众数最大的
    return items[-1][0]
```

这里根据k个邻居决定了预测的值。

KNN回归

回归相比分类区别不大，只是引进了概率，邻居的情绪不再是单一的而是存在概率的。因此利用公式进行统计：

$$P(test1\ is\ happy) = \frac{train1\ probability}{d(train1, test1)} + \frac{train2\ probability}{d(train2, test1)} + \frac{train3\ probability}{d(train3, test1)}$$

这里可见加了权重，距离更近的会在计算概率的时候占据更大的比重。

为了对实验结果进行评测，不能再像分类那样直接计算一个正确率，因此引入相关系数：

$$r(X,Y)=\frac{Cov(X,Y)}{\sqrt{Var[X]\,Var[Y]}}$$

距离计算

这一个和回归是一样的

k值邻居

```
def k_result(k, res):
    res.sort(key=lambda x: x['distance'])
    emotions = {'anger': 0, 'disgust': 0, 'fear': 0, 'joy': 0, 'sad': 0, 'surprise': 0}
    emotions_map = ['anger', 'disgust', 'fear', 'joy', 'sad', 'surprise']
    for i in range(k):
        for one_emo in range(6):
            try: #每一种情绪都要处于自身的距离，这里跟公式是相符合的
                emotions[emotions_map[one_emo]] += res[i]['emotion'][one_emo] / res[i]['distance']
            except: #出现了相同的句子，被除数不能为0应该直接采纳训练的数据
                for one_emo in range(6):
                    emotions[emotions_map[one_emo]] = res[i]['emotion'][one_emo]
                return emotions
    # 这一部分是在做归一化
    emotion_sum = sum(emotions.values())
    for key in emotions_map:
        emotions[key] = emotions[key] / emotion_sum
    return emotions
```

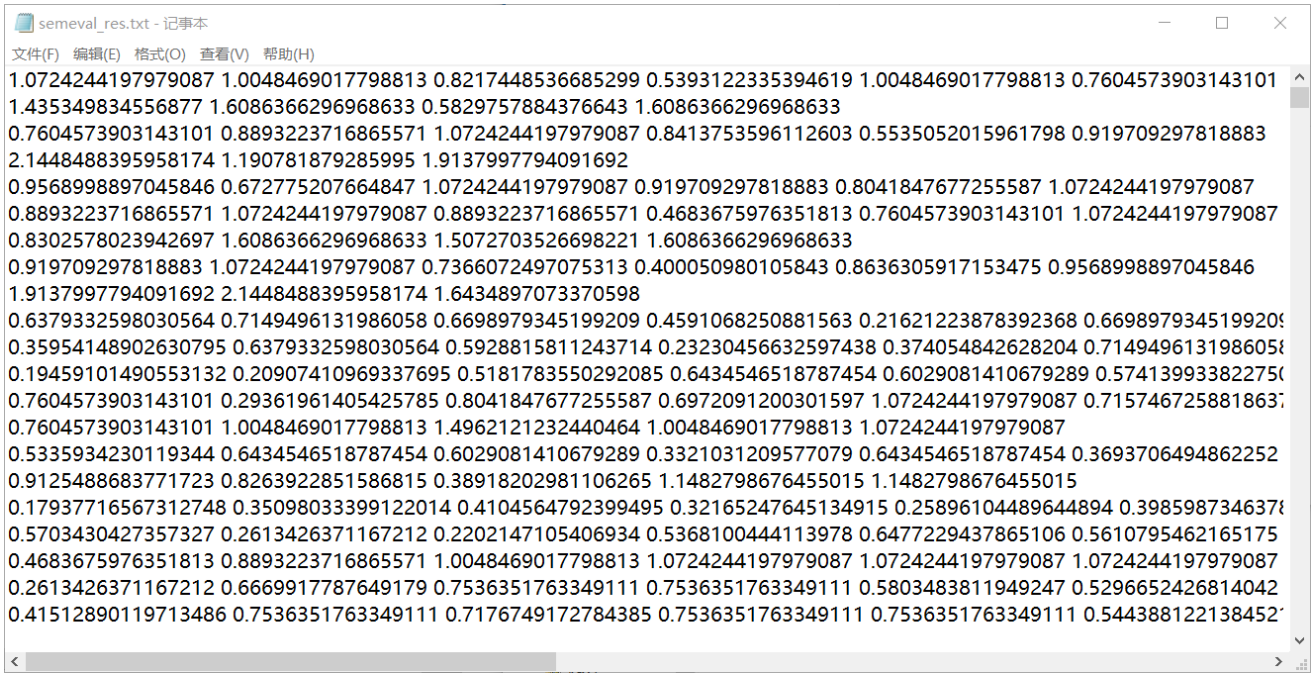
这里说明一下，数据结构跟TFIDF矩阵那块是相近的，每个句子处理成向量后用字典的方式存储。尤其针对于情绪这一数据，则用key=emotion这一键值存储，然后根据回归或者分类存放列表或者字符串的value。

实验结果及分析

实验结果

SemEval实验

已经附在文件中:



KNN分类

欧式距离下，取不同的k：

K	准确率
1	0.3633440514469453
2	0.29260450160771706
3	0.38263665594855306
4	0.3665594855305466
5	0.3858520900321543
6	0.40192926045016075
7	0.4115755627009646
8	0.40192926045016075
9	0.40192926045016075
10	0.40514469453376206
11	0.41479099678456594
12	0.42765273311897106
13	0.41479099678456594
14	0.3987138263665595
15	0.3987138263665595
16	0.39228295819935693
17	0.3987138263665595
18	0.40192926045016075
19	0.39228295819935693

采用曼哈顿距离：

K	准确率
1	0.3633440514469453
2	0.2958199356913183
3	0.37942122186495175
4	0.37942122186495175
5	0.3890675241157556
6	0.3987138263665595
7	0.40514469453376206
8	0.3954983922829582
9	0.40192926045016075
10	0.3987138263665595
11	0.4115755627009646
12	0.4180064308681672
13	0.41479099678456594
14	0.3890675241157556
15	0.3954983922829582
16	0.3858520900321543
17	0.3890675241157556
18	0.39228295819935693
19	0.38263665594855306
20	0.38263665594855306

KNN回归

欧式距离下，取不同的k：

K	相关系数
1	0.19762183462359695
2	0.2152888988975209
3	0.2692930844999953
4	0.2749753104480014
5	0.24547036852944495
6	0.24761830609581095
7	0.24949998127255654
8	0.24841248552672401
9	0.24964404633238388
10	0.24527138471804413
11	0.2553485551482967
12	0.2606659573099478
13	0.2598099314241365
14	0.2672839497214113
15	0.2699938928791597
16	0.26512068809288764
17	0.25703575435766335
18	0.24770252867269957
19	0.24464334135780294
20	0.24299777465822991

采用曼哈顿距离：

K	回归系数
1	0.203291590087877 14
2	0.2343034168356356
3	0.27916862597787984
4	0.2880003798661946
5	0.2613570481078732
6	0.25017637081674393
7	0.2505934178268823
8	0.2511313421261893
9	0.25597241832240275
10	0.25547041835778356
11	0.26234869305038416
12	0.26304063199415273
13	0.2685962612128177
14	0.26704191523893445
15	0.2713478515770608
16	0.2627459280205611
17	0.2585861502125066
18	0.24783093353882055
19	0.24772176096750953
20	0.2506381277940795

对比之下曼哈顿距离和欧式距离的准确率没有比较大的区别，但是随着K值的改变，准确率有着先升后降的趋势。说明K值的取值是KNN模型下很关键的一个因素。数据量大约在300+的情况下，K值确实在根号数据量的大小附近取得了最优值。

思考题

1. IDF的第二个计算公式中分母多了个1是为了什么？

   防止分母为0出现错误，如果测试集中没有这个单词出现

2. IDF数值有什么含义？TF-IDF数值有什么含义？

   IDF数值越大，说明单词在句子中意义越大，反之说明意义越小。TF-IDF数值表示了单词对句子性质的影响有多大。IDF用来避免介词这种频繁出现的单词可能导致的认为两个不相干的句子因为介词相近而被认为是相似含义句子的可能性。TF-IDF则已经处理完毕，可用向量距离来判断句子之间的差距。