文本数据集的简单处理与 K 最近邻算法的实现

17341146 王程钥

1 算法原理

1.1 TF-IDF 矩阵处理

将文本数据进行编码，可以使文本数据拥有可计算性，进行学习。将文本和单词的出现性 01 编码可以获取 OneHot 矩阵。将 OneHot 矩阵的每一行除以这个样本的单词数可以获取 TF 矩阵。根据每个单词的出现次数进行处理可以获取 IDF 矩阵。将两个个矩阵相乘可以获得 TF-IDF 矩阵。

1.2 KNN 算法

KNN 算法是一个基于贪心的机器学习算法。它将数据的特征抽象成空间中的点，通过在空间中寻找距离测试数据节点前 k 近的训练数据节点，并对其进行相关分析和计算，完成分类、回归等任务。

对于分类任务，传统的 KNN 算法将选择测试集对应的 k 近的训练集节点中出现次数最多的标签作为测试集的分类结果。对于回归任务，传统的 KNN 算法会根据测试集对应的 k 近的训练集节点的概率分布加权获得测试集的回归结果。

2 算法流程

Algorithm1 Data precession

```
Input: Text T
Output: TF-IDF
For text in T do
    For word in text do
        word_set.append(word)
        occur[word]+=1
        OneHot[text_index][word_index]+=1
    TF[text_index]=OneHot[text_index]/len(word)
For word in word_set do
    IDF[word_index]=occur[word]
For i in text_index do
    For j in word_index do
        TF-IDF[i][j]=TF[i][j]*IDF[j]
```

算法 1 为 TF-IDF 矩阵处理的简要伪代码。输入文本数据，返回 TF-IDF 矩阵。

Algorithm2 Get prediction

```
Input: train_x, train_y, test_x ,K (x 表示特征, y 表示标签)
Output: 对 test_x 的预测结果 P
For test in test_x do
    For train in train_x do
        S.append(test 和 train 的距离)
```

```
    对数组 S 排序
    T = S 集合中前 k 小的距离
    Pred = T 的众数
    P.append(Pred)
```

算法 2 为 KNN 算法的主要部分。输入训练集的样本、标签和测试集的样本数据，对于每个样本选择 k 近点，得到并返回预测结果。以上为分类问题的代码，回归问题与之类似，只有取众数部分不同，因此不再列出。

Algorithm3 Find best parameter and get result

```
Input: train_x, train_y, valid_x ,valid_y, test_x (x 表示特征,y 表示标签)
Output: 对 test_x 的预测结果 P
Max_ans=0, maxk=0
For K in range(1,30) do
    Result = Get prediction(train, train_y, valid_x, K)
    If Accyracy(Result, valid) > max do
        max_ans = accuracy, maxk = K
P=Get prediction(train, train_y, test_x, maxk)
```

算法 3 为 KNN 算法的主程序，枚举 K 值，对于验证集找到最优的 K 值，并根据该 K 值对测试集进行预测得到结果。

3 代码解析

3.1 TF-IDF 矩阵

```
index=0
with open(input_path,'r') as in_file:
    index=0
    for line in in_file:
        num,label,text,_=re.split('\t|\n',line)
        text=text.split(' ')
        num_word.append(len(text))
        for word in text:
            if not word_pos.get(word):
                word_pos[word]=len(word_pos)
                # map word to integer
            word_list.setdefault(word,[]).append(index)
            # add word to dictionary
        index+=1
```

获取单词集和单词出现位置

```
for (key,values) in word_list.items():
    i=word_pos[key]
    length=len(values);pre=''
    for j in values:
        if j==pre: length-=1
        pre=j
```

```
        # deduct repetitive word
    IDF[i]=log10(text_size/(1+length))
    for j in values: OneHot[j][i]+=1
```

计算 IDF 矩阵

```
for i in range(text_size):
    for j in range(word_size):
        TF=OneHot[i][j]/num_word[i]
        TFIDF[i][j]=TF*IDF[j]
```

计算 TF 矩阵与 TF-IDF 矩阵

3.2 分类任务

```
def get_distance(x,y,type='cos'):
    if type=='L1':
        return norm(x-y, ord=1)
    elif type=='L2':
        return norm(x-y, ord=2)
    elif type=='cos':
        A=norm(x); B=norm(y)
        if A==0 or B==0: return 1e9
        return 1-np.dot(x,y)/(A*B)
    else: assert(0)
```

获取两个向量的距离

```
def gaussian(dist, a=1, b=0, c=0.3):
    return a * math.e ** (-(dist - b) ** 2 / (2 * c ** 2))
```

计算高斯距离

```
def testing(trains_x, trains_y, tests_x,
             K=13, distance='cos', weight=1):
    prediction=[]
    for test_x in tests_x:
        val=[]
        for train_x,train_y in zip(trains_x,trains_y):
            dis=get_distance(train_x,test_x,type=distance)
            val.append((dis,train_y))
        val.sort()
        # get distance and sort

        dict_feature={}
        for res in val[:K]:
            dis,word=res
            if not dict_feature.get(word): dict_feature[word]=0
            if weight==0: dict_feature[word]+=1
            else: dict_feature[word]+=gaussian(dis)
        # calculate k nearest distance
```

```
        maxn=-1;predict=''
        for key,val in dict_feature.items():
            if val>maxn: predict=key;maxn=val
        prediction.append(predict)
        # get prediction

    return prediction
```

KNN 分类预测，选择 K 近邻加权获得预测结果。

```
def solve(trains_x, trains_y, valids_x, tests_x, K_range=range(1,31),
          distance='cos', weight=1, feature='TF-IDF'):
    max_A=0; max_K=0; X=[]; Y=[]
    for K in K_range:
        predict=testing(trains_x, trains_y, valids_x,
                        K, distance, weight)
        accuracy=get_accuracy(predict,valids_y)
        print("K = %d, accuracy = %f"%(K,accuracy))
        if accuracy>max_A: max_A=accuracy; max_K=K
        X.append(K)
        Y.append(accuracy)
    print("max_K = %d, max_accuracy = %f"%(max_K,max_A))
    plot_result(X,Y,'KNN classify accuracy')

    result=testing(trains_x, trains_y, tests_x, max_K)
    write_result(result)
```

KNN 分类预测部分主函数，枚举参数，选择在验证集上的最优参数参数预测测试集。

3.3 回归任务

```
def evaluate(predict,lables):
    predict=np.array(predict)
    lables=np.array(lables)
    sum_r=0
    for i in range(6):
        A=predict[:,i]
        B=lables[:,i]
        ans,_=stats.pearsonr(A, B)
        sum_r+=ans
    return sum_r/6
```

计算相关度，先对每个情感类型计算相关系数，后对六个相关系数取平均数。

```
def testing(trains_x, trains_y, tests_x,
            K=13, distance='cos', weight=1):
    regress=[]
    for test_x in tests_x:
        val=[]
        for train_x,train_y in zip(trains_x,trains_y):
```

```
            dis=get_distance(train_x,test_x,type=distance)
            val.append((dis,train_y))
        val.sort()
        # get distance

        prob = np.zeros([6])
        for res in val[:K]:
            dis=res[0]; feature=np.array(res[1])
            if weight==0: prob=prob+feature/max(0.0001,dis)
            else: prob=prob+feature*gaussian(dis)

        if sum(prob)==0: prob = np.random.rand(6)
        prob/=sum(prob)
        regress.append(prob)
        # get regress

    return regress
```

KNN 回归预测，选择 K 近邻加权获得概率回归结果。

```
def solve(trains_x, trains_y, valids_x, tests_x, K_range=range(1,31),
          distance='cos', weight=1, feature='TF-IDF'):
    max_R=0; max_K=0; X=[]; Y=[]
    for K in K_range:
        predict=testing(trains_x, trains_y, valids_x,
                        K, distance, weight)
        R=evaluate(predict, valids_y)
        print("K = %d, R = %f"%(K,R))
        if R>max_R: max_R=R; max_K=K
        X.append(K); Y.append(R)
    print("max_K = %d, max_R = %f"%(max_K,max_R))
    plot_result(X,Y,'KNN regress accuracy')
    result=testing(trains_x, trains_y, tests_x, max_K)
    write_result(result)
```

KNN 回归预测部分主函数，枚举参数，选择在验证集上的最优参数预测测试集。

4 优化点与部分说明

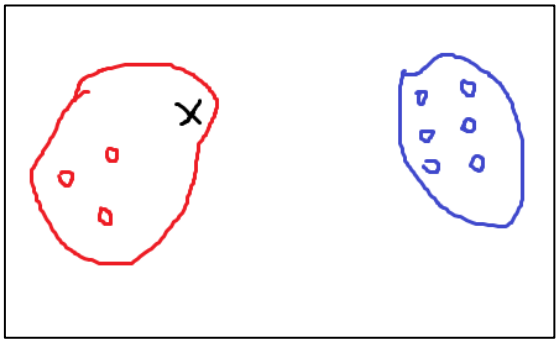
4.1 距离度量

传统的 KNN 算法使用的是欧几里得距离，即 L2 范数。实际上对于文本数据，使用余弦距离是一个更自然的选择。余弦距离更注重两个向量在方向上的差异而不是长度，更注重一个单词是否出现而不是出现几次。这更符合文本学习的要求。

$$\mathrm{dis}(\mathrm{A},\mathrm{B}) = 1 - \cos(\mathrm{A},\mathrm{B}) = 1 - \frac{A * B}{|A||B|} \qquad (1)$$

4.2 距离加权

本次实验的数据集存在样本不均衡的问题。表 1 为训练集中 label 的统计，总共 598 个数据中，标签为 disgust 的数据只有 19 个，而标签为 joy 的数据有 217 个。KNN 算法是基于众数的，样本不均衡会带来很大的问题。如图 1 所示，对于测试集黑色 X，它应该是属于红色集合的。但是如果 k>6，那么蓝色点就肯定会比红色点多，X 也就会被分到蓝色集合内。



（图 1 样本不均衡问题）

因此给距离一个权重，选择加权距离最大的 label 作为预测结果是一个更好的方法。我们希望选择一个单调递减的非负函数，在 x 趋近于 0 的时候值较大，在 x 足够大时值较小。同时，为了防止噪声的影响，在 x 很小的时候函数值不能过大，否则就相当于 1 临近算法了。因此，我最终选择了高斯函数对距离进行加权。

$$\text{Gaussian}(x) = a * e^{\frac{-(x-b)^2}{2*c^2}} \qquad (2)$$

标签	anger	Disgust	Fear	Joy	Sad	Surprise
数量	40	19	90	217	124	108

（表 1 样本标签统计）

4.3 特征选取

课件中使用了 OneHot 矩阵作为特征。因为之前第一部分的实验实现了 TF-IDF 矩阵，因此我测试了 TF-IDF 矩阵和 Onehot 矩阵这两种文本特征并进行了比较。结果发现使用 TF-IDF 矩阵的实验结果更优。

4.4 全 0 特征

测试集文本中出现的单词可能不在训练集中出现。对于这些文本，生成的特征向量是一个全 0 向量。如果选择余弦距离，那么计算结果是正无穷。因此我们返回一个无穷大的值，代入高斯函数的结果是 0。对于这种情况，我们会随机产生一个结果。

5 实验结果

5.1 节为算法正确性测试，后面几节为对于若干参数的隔离实验。

在这一节中，我调整了距离度量方式，距离加权，特征选取，K 值这几个变量，获取实验结果并进行比较。实验结果可见下表或本章的图片。下表的结果为四舍五入后的结果。

对于 5.2 节-5.5 节，在测试的时候统一使用[1,4,7,10,13,16,19,22,25]这几个 k 值。对于 5.6 节，使用[1,30]区间内的所有 k 值。

距离度量	特征	距离加权	分类/k 值	回归/k 值
L2	TF-IDF	否	32% / 1	0.28 / 25

L1	TF-IDF	否	40% / 13	0.33 / 13
Cos	TF-IDF	否	44% / 10	0.38 / 4
Cos	OneHot	否	38% / 7	0.33 / 7
cos	TF-IDF	是	50% / 10	0.43 / 10

（表 2 实验结果总结）

5.1 算法正确性测试

我构造了如下的小型的训练集和测试集，并进行了训练。比如训练集的第一句和测试集的第一句都含有 big win,两者的标签都是 joy。第二句都含有 pensions 和 revoke，两者的标签也都是 sad。

实验结果显示预测成功，准确率 100%。

	A	B
1	Words (split by space)	label
2	europe retain trophy with big win	joy
3	senate votes to revoke pensions	sad
4	the amounts you have to pay for a bomb scare	fear
5	pair of satellites will document sun in d	joy
6	malaysian airasia x to fly in july	joy

	A	B
1	Words (split by space)	label
2	Real Madrid has a big win	joy
3	pensions will be revoke	sad
4	I am scare	fear
5		

```
3
K = 10, accuracy = 1.000000
['joy', 'sad', 'fear']
3
K = 11, accuracy = 1.000000
['joy', 'sad', 'fear']
max_K = 10, max_accuracy = 1.000000
```

（图 2-4 从左往右依次为训练集，测试集，实验结果）

5.2 标准 KNN 的结果

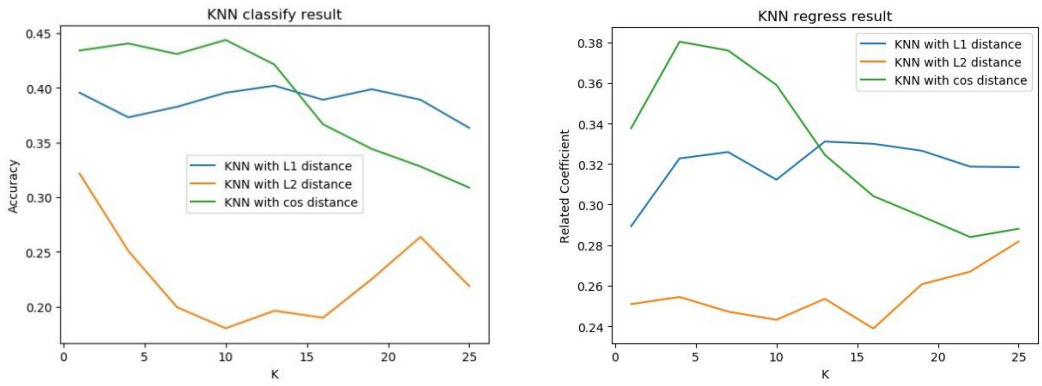
根据实验要求文件，模型采用 L2 范数计算距离，采用 OneHot 矩阵作为特征，且没有进行距离加权操作。我没有使用 OneHot 特征，而是使用了更优秀的 TF-IDF 特征。我选取了一些 k 值，在分类任务和回归任务下分别测试了这种条件下的分类结果。由表 3 可见，实验 KNN 算法大致可以获得 32%的准确率和 0.28 的相关系数。

K 值	1	4	7	10	13	16	19	22	25
分类	32.15%	25.08%	19.94%	18.01%	19.61%	18.97%	22.51%	26.37%	21.87%
回归	0.251	0.254	0.247	0.243	0.254	0.239	0.261	0.267	0.282

（表 3 实验结果）

5.3 距离度量

我选取了 k=1,k=3,k=8,k=13,k=20，在分类和回归两个任务上分别测试了 L1 范数，L2 范数，余弦距离三种不同的距离度量方式下分类准确率的结果。如图所示，使用余弦距离的分类准确率最高，其次是 L1 范数，其次是 L2 范数。

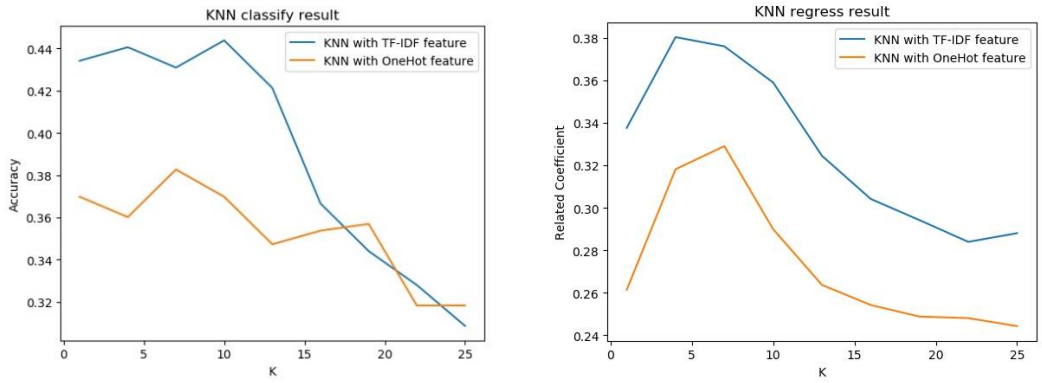


（图 5-6 不同距离度量的分类与回归结果）

5.4 特征选取

我测试了使用 OneHot 矩阵和 TF-IDF 矩阵两种特征选取方式的情况下，使用余弦距离，

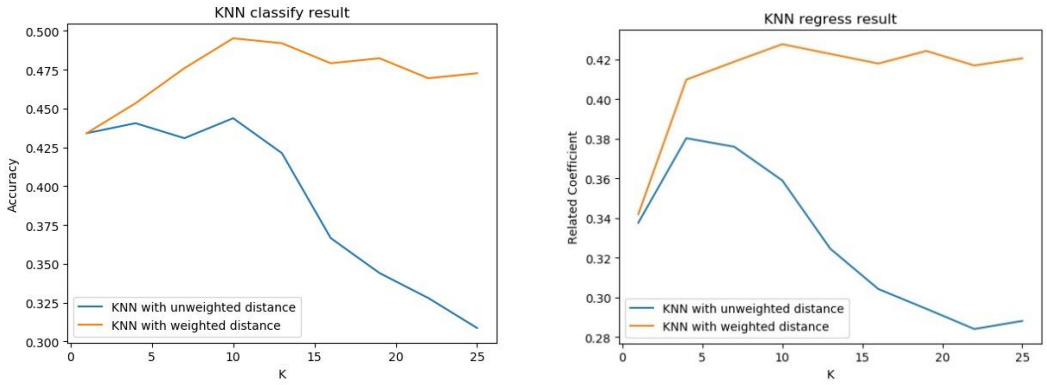
不使用距离加权时分类和回归任务的结果。可以看到无论是回归还是分类，使用 TF-IDF 特征的预测结果都明显优于 OneHot 特征。



（图 7-8 不同特征的分类与回归结果）

5.5 距离加权

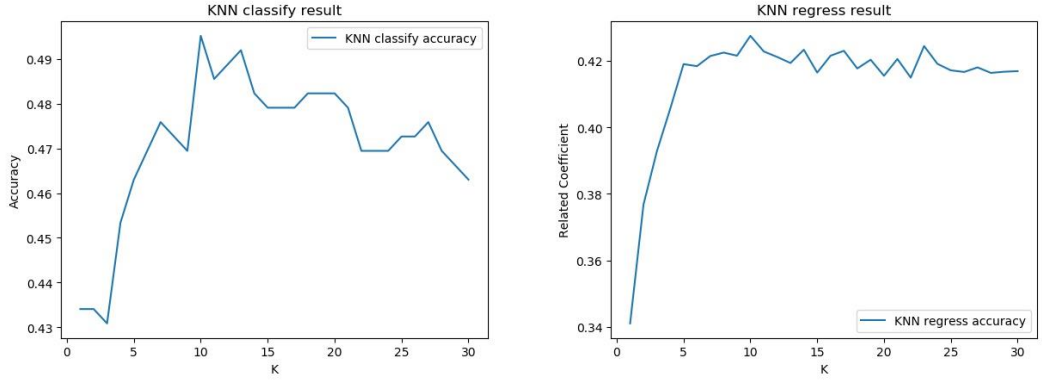
根据 4.2 节，我测试了在回归和分类任务下，使用余弦距离时使用与不使用距离加权的结果。如图，使用距离加权可以较好地克服样本类别不均衡的问题，使得 k 较大的时候准确率依然能保持在一个很高的水准，不受样本不均衡问题的影响。



（图 9-10 距离加权的分类与回归结果）

5.6 不同的 K 值

我们加上了第四章中提到的所有优化，包括距离加权，特征选取，距离表示计算。我们在（1,30）的范围内枚举 K 值，最终得到如下的分类和回归的预测结果。如图所示，对于分类任务，当 k=10 的时候分类准确率最高，可以达到 49.5177%。对于回归任务，当 k=10 的时候相关系数最高，可以达到 0.427461。



（图 11-12 不同 k 值的分类与回归结果）

以下是分类任务的混淆矩阵。总体而言 joy，sad，surprise 这三个类别的准确率较高，这主要也是因为这三类样本的数量较多。准确率总体和样本数量成正比。由于 joy 类的样本较多，所以很多数据都会被预测为 joy 类。

标签\分类	anger	disgust	fear	joy	sad	surprise
anger	**19.04%**	4.76%	33.33%	23.81%	14.29%	4.76%
disgust	0	**23.08%**	23.07%	23.07%	23.08%	7.69%
fear	1.85%	1.85%	**33.33%**	22.22%	29.63%	11.11%
joy	1.79%	0	8.04%	**68.75%**	8.93%	12.50%
sad	3.08%	0	15.38%	18.46%	**50.77%**	12.31%
surprise	4.35%	0	6.52%	36.96%	10.87%	**41.30%**

（图 13 混淆矩阵，x 行 y 列表示标签为 x 时预测结果为 y 的概率）

6 思考题

6.1 IDF 的第二个计算公式中分母多 1 的原因

防止有单词在文章中没出现，出现除以零导致运行时错误。

6.2 IDF 数值和 TF-IDF 数值的含义

1) IDF 是逆向文本频率，它衡量了单词的重要程度。包含该单词的文本越少，IDF 值越大，说明该单词越重要。

2) TF-IDF 中的每一项表示一个词在这个文本中的重要程度。TF-IDF 为单词在文本中出现频率和单词重要性的乘积。在一个文本中，一个词出现次数越多，TF-IDF 值越高。单词越重要，TF-IDF 值也越高。

6.3 回归问题距离为什么要取倒数

距离越近的样本点对答案的贡献显然应该越大。距离的倒数随着距离的增大而减小，满足距离越小贡献越大的特点，且当距离无穷大时贡献只会趋近于零不会变成负数。所以使用距离的倒数加权。

6.4 统一测试样本各个情感概率总和为 1

预测结束后对概率分布进行归一化，除以概率分布的总和，即可得到和为 1 的概率分布。

7 结论

本次实验要求使用 KNN 算法在文本数据集上完成了文本分类和回归的任务。我实现了 KNN 算法，并针对训练集样本不均衡的问题，在距离度量，距离加权等部分对 KNN 算法进行了改进。最终我的方法最终在分类任务上获得了 49.5177%的准确率，在回归任务上获得了 0.427461 的相关性。