

中山大学数据科学与计算机学院

计算机科学与技术 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	16级计科二班	专业 (方向)	计算机科学与技术
学号	16337334	姓名	周启恒

(1) 算法原理

TFIDF

1. 读取文件, 将句子分成单独的词汇, 并汇总成一个word_list。
2. 根据word_list对每个句子构建一个one-hot矩阵。
3. 根据公式, 将one-hot矩阵处理为TF矩阵, 对word_list进行处理得到IDF矩阵
4. $TF * IDF$ 得到TF-IDF

最终IDF的每一项运用这个公式进行计算:

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

KNN-classification & KNN-regression

算法理解:

首先通过对训练集和验证集的处理, 得到one-hot矩阵。

取验证集中的一项, 计算它与训练集所有数据之间的距离, 选择L组最近的数据, 并找出L组中类别的众数, 将它作为验证集此项的预测结果。

1. classification: 对验证集所有数据进行上述操作, 得到一个预测结果集 (离散), 直接与真实结果进行比对, 计算准确率, 准确率可代表 $k=L$ 时预测的效果。
2. regression: 对验证集所有数据进行上述操作, 得到一个预测可能性集 (连续), 计算它与真实结果集的相关系数, 这个相关系数对应的是 $k=L$ 时预测的效果。

遍历L, 可找到一个最佳的k, 将它作为最终对测试集分类时的参数。

最终使用余弦距离进行计算：

$$\cos(a, b) = \frac{a \cdot b}{|a| \cdot |b|}$$

(2) 流程图

KNN流程图：



(3) 关键代码

生成word_list

```
#对某个数据集进行单词提取，得到单词集
def create_word_list(word_list, data):
    for sentence in data['Words (split by space)']:
        for word in sentence.split(' '):
            #没有重复的
            if word not in word_list:
                word_list.append(word)
```

合并word_list

```
#个人理解，在计算one-hot之前，应该将训练集与验证集或测试集的单词集进行合并
#生成一个总的单词集
def join_word_list(word_list1, word_list2):
    word_list = list(word_list1)
    for word in word_list2:
        if word not in word_list:
            word_list.append(word)
    return word_list
```

生成one-hot矩阵

```
#处理得到one-hot矩阵
...
```

在生成总的单词集之后，对每组数据进行处理，得到各自的one-hot矩阵
将每个句子中出现的单词，在one-hot中对应标记

```

'''
def create_one_hot(word_list, data):
    word_list_len = len(word_list)
    one_hot = list()
    #这里是一个总的one-hot, 记录了整个数据集
    for sentence in data['Words (split by space)']:
        tmp_one_hot = np.zeros(word_list_len)
        for word in sentence.split(' '):
            tmp_one_hot[word_list.index(word)] = 1
        one_hot.append(tmp_one_hot)
    return np.array(one_hot)

```

TF-IDF

```

for sentence in sentence_list:
    #print(len(sentence))
    tmp_one_hot = list( 0 for i in range(word_list_len) )

    for word in sentence:
        tmp_one_hot[word_list.index(word)] += 1

    #word count---IDF
    for index in range(word_list_len):
        if tmp_one_hot[index] != 0:
            word_count[index] += 1

    #TF
    tmp_one_hot = [float(i) / float(len(sentence)) for i in tmp_one_hot]
    one_hot.append( list(tmp_one_hot) )

TF = np.array(one_hot)
IDF = np.array(word_count)
#get IDF
IDF = np.log2( sentence_list_len/(IDF+1) )
TF_IDF = TF * IDF

```

计算one-hot之间的距离

```

#计算两集合之间的距离
def calculate_distance(distance_list, validation_one_hot, train_one_hot):
    for i in range(len(validation_one_hot)):
        tmp_distance = list()
        for j in range(len(train_one_hot)):
            #distance calculating method
            #tmp_distance.append(math.sqrt(np.sum( (train_one_hot[j] -
validation_one_hot[i])**2 ))) #欧式距离
            #tmp_distance.append(np.linalg.norm(train_one_hot[j] -
validation_one_hot[i])) #欧式距离
            tmp_distance.append( np.dot(train_one_hot[j],
validation_one_hot[i])/(
np.linalg.norm(train_one_hot[j])*np.linalg.norm(validation_one_hot[i]) ) )
#余弦距离
        distance_list.append(list(tmp_distance))

```

分类

```

def get(index_list, validation_index, accuracy_list, k):
    label_dict = {}
    #选择k个数据
    for i in range(0, k):
        ...

        if train_data['label'][index_list[i]] not in label_dict:
            label_dict[train_data['label'][index_list[i]]] = 1
        else:
            label_dict[train_data['label'][index_list[i]]] += 1
        ...

    #对相似度进行加权
    if train_data['label'][index_list[i]] not in label_dict:
        label_dict[train_data['label'][index_list[i]]] =
float(1/(distance_list[validation_index][index_list[i]] + 0.001))
    else:
        label_dict[train_data['label'][index_list[i]]] +=
float(1/(distance_list[validation_index][index_list[i]] + 0.001))

    #predict via K value
    #找到k个预测值中的众数，将它作为预测结果
    prediction = max(label_dict,key=label_dict.get)
    correct_answer = validation_data['label'][validation_index]
    #如果预测结果正确，正确数加一，方便之后计算准确率
    if prediction == correct_answer:
        accuracy_list[k] += 1

```

回归

```

#与分类中相似，取k个数据。
#对k个概率进行平均，得到预测的可能性
def get(index_list, validation_index, probability_list, distance, k):
    predict_probability = np.zeros(len(train_probability_list[0]))
    for i in range(k):
        #存在distance为0的情况，故添加一个很小的值，防止除以0现象。
        predict_probability += (
probability_list[index_list[i]]/(distance_list[validation_index]
[index_list[i]]+0.0001) )
        #归一化
    tmp_sum = sum(predict_probability)
    predict_probability /= tmp_sum
    return predict_probability

```

回归找最优K

```

#找到最优的k
'''
输入之前的最优k,最优的相关系数，还有当前测试的k
计算相关系数，与之前相关系数进行比较。
'''
def get_best_k(min_corr, best_k, k):
    predict_probability_list = list()
    for validation_index in range( len(distance_list) ):
        index_list = np.argsort(distance_list[validation_index])
        #这里采用余弦距离，故需要将大小顺序调整。
        index_list = index_list[::-1]
        predict_probability_list.append(get(index_list, validation_index,
train_probability_list, distance_list[validation_index], k))
    cov_list = list()
    predict_probability_list = np.array(predict_probability_list)

    tmp = 0
    #对概率向量进行处理
    for i in range(len(predict_probability_list[0])):
        a = predict_probability_list[:,i]
        b = validation_probability_list[:,i]
        #计算相关系数
        tmp += np.corrcoef(a,b)[0][1]
    tmp = tmp / len(predict_probability_list[0])
    if tmp > min_corr:
        min_corr = tmp
        best_k = k
    return min_corr, best_k

```

(4) 创新点&优化

因为基本按照ppt所给的内容进行实验，没有重要的优化点。

1. 将欧式距离改为余弦距离。
2. classification中对相似度进行加权

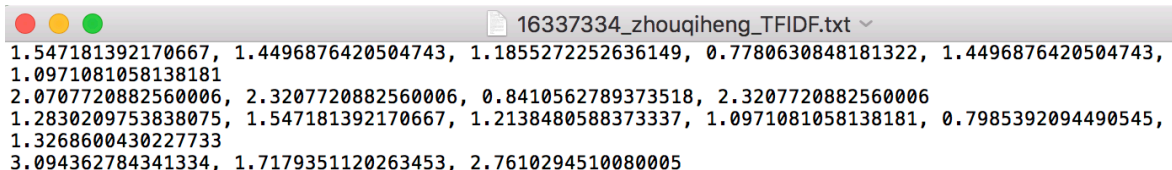
$$power = \frac{1}{distance_i + d}$$

对于相似度越高的数据组，权值越高。

```
label_dict[train_data['label'][index_list[i]]] =  
float(1/(distance_list[validation_index][index_list[i]] + 0.001))
```

(5) 实验结果展示

- TFIDF部分结果：



1.547181392170667, 1.4496876420504743, 1.1855272252636149, 0.7780630848181322, 1.4496876420504743,
1.0971081058138181
2.0707720882560006, 2.3207720882560006, 0.8410562789373518, 2.3207720882560006
1.2830209753838075, 1.547181392170667, 1.2138480588373337, 1.0971081058138181, 0.7985392094490545,
1.3268600430227733
3.094362784341334, 1.7179351120263453, 2.7610294510080005

- KNN-classification部分结果：

	Words (split	label
0	senator carl	joy
1	who is princ	joy
2	prestige has	joy
3	study female	joy
4	no e book fo	joy
5	blair apolog	sad
6	vegetables r	joy
7	afghan force	sad
8	skip the sho	surprise

- KNN-regression部分结果：

	anger	disgust	fear	joy	sad	surprise
0	0.04894596	0.0895189	0.09345908	0.05943826	0.16911292	0.53952487
1	0.0279464	0.179091	0.10306522	0.37003231	0.16540455	0.15446051
2	0	0	0.01534726	0.56548543	0.23780775	0.18135956
3	0	0.05396281	0	0.53689991	0	0.40913728
4	0.03482246	0.02176077	0	0.47399241	0.04354767	0.42587668
5	0.04006275	0.14147216	0.13101915	0.11360349	0.23369995	0.3401425
6	0	0	0.1142879	0.47506389	0.00826756	0.40238065
7	0.08756348	0	0.48096273	0.0312519	0.26627537	0.13394652
8	0	0.10217681	0	0.12334405	0	0.77447915
9	0.08213376	0.09856974	0.24352745	0.32229495	0.08989077	0.16358333

(6) 评测指标展示

- KNN-classification: k=13,
 - 改进前准确率 = 0.4052221349512136
 - 改进后准确率 = 0.4212218649517685
- KNN-regression: k=3时, 相关系数 = 0.2854616615436468