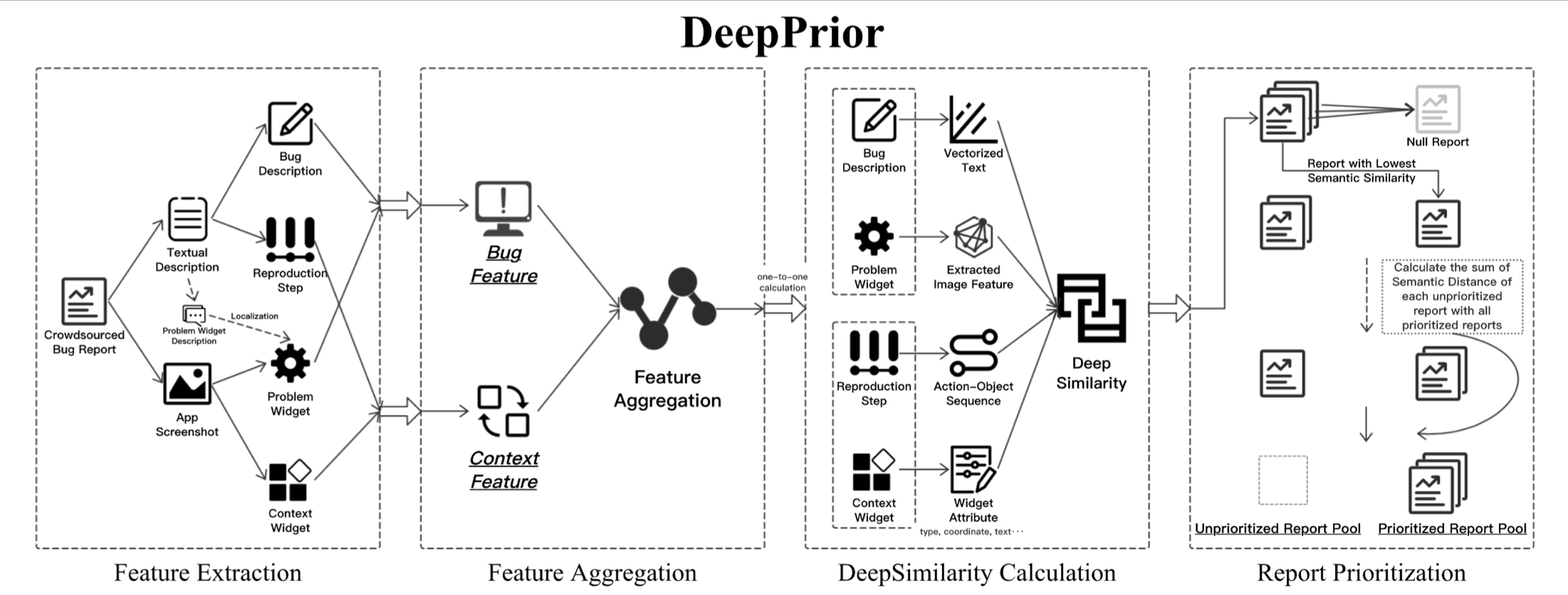
**Author: 191820133 陆昱宽**

**DeepPrior**



Volatile组在迭代三的算法参考了论文*Prioritize Crowdsourced Test Reports via Deep Screenshot Understanding*, 因此先介绍DeepPrior算法。

DeepPrior把众测报告分为Bug描述和Bug上下文两部分， 对应特征为Buf Feature和Context Feature, 每个报告都由这两个特征表示， 根据这两个特征来计算报告的相似度，并进行报告推荐

具体思路是:

报告从形式上可以分为图片和文本两部分， 图片可分为展现了Bug的图片区域和Bug周围的正常区域（即Bug图片上下文区域）；文本可分为对Bug的描述和对Bug的复现步骤。 我们将Bug图片区域和对Bug的文字描述作为Bug Feature, Bug图片上下文区域和对Bug的复现步骤作为Context Feature， 将BFT和CFT提取出来，就得到了报告的向量表示

对于两篇报告，只要分别计算其BFT和CFT的相似度( , ), 再加权和，就得到了两篇报告的整体相似度

DeepPrior计算报告间两两的相似度，得到报告的相似度矩阵，这就实现了报告的相似度计算。

在报告推荐环节，许多推荐算法直接将相似度作为推荐程度的逆，即报告越不相似，推荐程度就越高， 这个方法的缺陷在于推荐结果依赖于所给的报告，而**报告推荐的目的是尽量将描述新bug的报告推荐给用户， 而不是将新报告推荐给用户，即推荐结果应该与bug有关而不是与当前报告有关**。DeepPrior采用了新的推荐方式， 通过引入“Null Report”， 作为报告比较的基准， 使得推荐结果不依赖于所给的具体报告

具体步骤为:

1. 特征提取：对图片，提取出Bug图片区域和Bug周围的正常区域； 对文本，提取出对Bug的描述和对Bug的复现步骤
2. 特征聚合： 把Bug图片区域和对Bug的文字描述聚合为Bug Feature, Bug图片上下文区域和对Bug的复现步骤聚合Context Feature
   * 注意，这里的“特征”都不是向量，不能直接拿来计算。 比如BugFeature就是图片和文字，而不是它们的向量表示
3. 相似度计算：对于两篇报告，分别计算其BFT和CFT的相似度，再将两个相似度加权和，作为两报告的整体相似度。 对所有报告两两进行相似度计算，生成报告相似度矩阵
   * BFT和CFT也都不是向量，其相似度计算，其实还是将图片和文本相似度分别计算，再加权和
4. 报告推荐：根据报告的相似度，并引入空报告，进行报告推荐

**特征提取**

**文字特征提取**

用CNN将文字区分为Bug Description和 Reproduction Steps

Bug Description是文本， 而Reproduction Steps是一个步骤序列，每个步骤都是一段文本

**图片特征提取**

移动应用的界面都由多个widget组成

1. 通过CV方法，将图片分为多个widget， 每个widget又由多个原子widget组合成
2. 对于每个widget，通过CNN，将其表示为一个多维向量，每一维代表一类原子widget，该维的值就是该widget所含有的该原子widget的数量， 比如一个widget含有3个Button， 2个Pane，它对应的向量就是{.3, 2, 0,0, ... }
   * 这里采用CNN作为分类，以识别widget的某个子组件是不是一个原子widget（是不是Button，是不是Pane，是不是Text Box.... ）
   * 所有widget都给出其向量表示，但这个向量表示只是用于简单地区分该widget的组件构成（该Widget有几个按钮、几个下拉框）， 并不能深刻反映组件的内容差异（组件的颜色、位置，文字内容... ）， 对于Context Widget, 我们认为它们的内容差异是可接受的，只要计算其构成即可； 但对于Bug Widget，其内容差异是很重要的，不能用这个低维向量来表示，因此还是要将其转化为高维的图像特征
3. 对于每个widget，识别其是否为Bug Widget（即表现了bug的图片区域）， 如果不是，则为Context Widget（没有表现bug的图片区域）
   * 如果该widget上有文字，就用ocr提取出来，和Bug Description计算相似度，如果相似，则判定为Bug Widget
   * 如果widget上没有文字，则送入CNN，判定其是否为Bug Widget

**特征聚合**

将Problem Widget和Bug Description合并为BFT, Context Widget和Reproduction Steps合并为CFT

这一步事实上没有做什么工作，因为图片和文字都不是向量形式，把他们聚合到一起也只是逻辑上的放在一起，没有在实际上将其聚合（比如向量合并等）

**相似度计算**

对于两篇Bug，计算其BFT和CFT的相似度，最后加权和作为总体相似度

**BFT**

**Problem Widget**

虽然

对于Problem Widget，计算其图像相似度，方法是用SIFT转成向量，用FLANN计算向量相似度

* SIFT的优点是可以处理不同大小、位置、旋转度的图片

**Bug Description**

使用word2vec计算其文本相似度

**CFT**

**Context Widget**

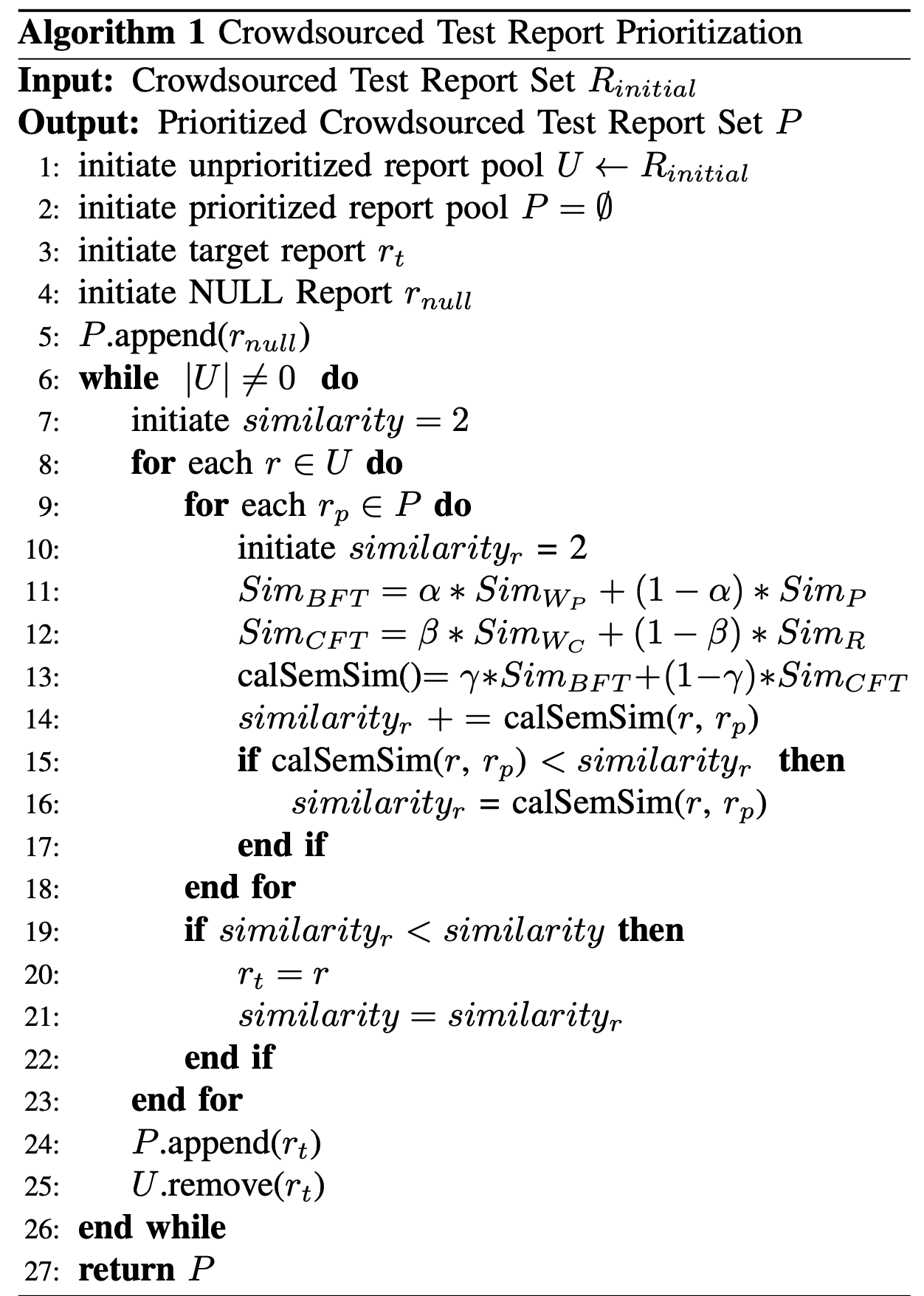
使用之前计算好的低维向量来表示Context Widget， 则Context Widget的相似度就是其向量的欧式范数

**Reproduction Steps**

使用DTW算法计算其步骤序列的相似度

**报告推荐**

由于如前所述的原因，DeepPrior通过引入空报告，来实现不受具体报告影响的报告推荐。 具体的推荐过程如论文所述:



**算法思考**

**相比DeepPrior的删减**

由于CNN的实现复杂，且需要大量的数据进行训练。 （而且个人感觉，对CNN的编程也没什么意义。。。），我在实现算法时修改了所有CNN相关的部分

DeepPrior中用到CNN的部分有：

* “将Context Widget进行分类并转化成低维向量”这一步需要CNN来进行图像分类。 我废除了该步骤，即不将Context Widget作为一个特征，也不对它进行后续的相似度计算（当然，可以用普通的图像相似度算法来计算Context Widget， 但由于Bug Widget就是这么做的， 如果Context Widget也这么计算，其结果就和直接将所有widget计算图像相似度一样，意义不大，因此直接省略了该步）
* “将文本转化成Bug Description和Bug Description”这一步需要CNN来进行文本到步骤序列的转化。 我的修改是使用分词模型来催文本进行分词，形成词序列，对词序列计算其相似度。
  + 这里的CNN有两个功能：将文本区分为Bug Description和Bug Description，并将Bug Description转化为步骤序列。 由于我们在前端界面设计时就将Bug的描述和缺陷复现步骤作为两个独立的区域让用户输入，因此不需要第一个功能，但第二个功能依然需要CNN实现

**相比DeepPrior的创新**

DeepPrior的问题有二：

1. 它没有将报告转化成一个统一的向量表示， 图片和文本的关系依然是各算各的，最终相似度也是二者的加权和，这缺乏可解释性，只是从结果上能说明该算法“似乎有效”。
2. 问题1也意味着报告**无法聚类**， 因为聚类需要线性空间中的对象，而报告没有能够在线性空间中表示，其唯一的数值特征就是其相似度，这是个一维特征，而且是相对的，没有独立意义。 即，DeepPrior只能算出某篇报告对其他报告的相似度，但没法计算/表示出该报告本身

由于论文本身的整体思路，就是将文本和图片分开处理，我也没办法解决问题1，没办法给出报告整体的向量表示。 但是我提出了一个办法来改善问题1，思路如下：

1. 可以将BFT和CFT作为报告的两个特征，即报告的两维，给出报告的“近似”向量表示。之所以说近似，是因为BFT和CFT本身也不是向量或者标量，而是图片和文字， 也没有“值”。 BFT和CFT在计算相似度的时候被转化成“相似度”这个数值
2. 那么，为了给BFT和CFT一个数值上的表示，可以选择一个报告作为基准报告，其他报告和它进行相似度计算，并用BFT相似度和CFT相似度来作为BFT和CFT这二维的值。这样就实现了报告的（二维）向量化
3. 这就引入了新的问题，由于报告的向量表示依赖于基准报告，该怎么选择基准报告？ 我的想法有二：
   1. 采用DeepPrior报告推荐中“空报告”的思想，用空报告作为基准，这样就使得向量表示是稳定的，不依赖其他报告
   2. 随即选择一篇报告作为基准报告
4. 虽然方法二有点傻，但我认为方法1是不可行的。 因为空报告的文字都为空，图片只是一个固定大小的色块，其他报告与其计算相似度，其结果都是一样的，其相似度也不能体现报告的特征。 因此我选择方法二

最终，我得到了报告的二维向量表示，二维分别是BFT和CFT， 由于得到了向量表示，也就能够聚类了

报告的向量表示：

report\_id,x,y

11,0.17047329607499934,0.4152

25,0.5002089775561097,0.8234

50,0.35525924861641534,0.8617

14,0.10322453323377659,0.41900000000000004

15,0.5,0.5

可以看到，report\_id = 11的报告，其向量为[0.17047329607499934,0.4152]

**算法实现**

我根据DeepPrior的思想实现了Volatile组的报告算法，主要包括报告推荐、报告相似度计算和报告聚类

**相似度计算**

相似度矩阵，其中-1,11,25,50,14,15表示的是对应的report\_id,  -1代表的是空报告

可以看到，矩阵的第三列，也就是report\_id = 11的那列，代表该报告相比其他报告的相似度

index,-1,11,25,50,14,15

-1,1.0,0.5149,0.4589,0.4201,0.2936,0.6554

11,0.5149,1.0,0.2646,0.3842,0.328,0.2439

25,0.4589,0.2646,1.0,0.4772,0.226,0.5972

50,0.4201,0.3842,0.4772,1.0,0.3692,0.5072

14,0.2936,0.328,0.226,0.3692,1.0,0.198

15,0.6554,0.2439,0.5972,0.5072,0.198,1.0

**得到相似度矩阵**

采用DeepPrior的思想得到报告相似度矩阵:

def cal\_sim\_matrix(report\_file\_path, text\_feature\_list, widget\_category\_list, problem\_widget\_path\_list,

report\_sim\_matrix\_file\_path, report\_vec\_file\_path):

'''

# calculate and record sim between every two reports in report list

计算相似度矩阵和报告的相似度向量，分别存入对应文件

:param report\_file\_path: report数据所在文件

:param text\_feature\_list:

:param widget\_category\_list:

:param problem\_widget\_path\_list:

:param report\_sim\_matrix\_file\_path: 相似度矩阵存入该文件

:param report\_vec\_file\_path: 报告转化成的向量存入该文件

:return:

'''

reports = pd.read\_csv(report\_file\_path)

report\_ids = list(reports[ReportEnum.REPORT\_ID]) # 报告id的列表

# get max and min value of distance (used for distance normalization)

max\_sift\_fea, max\_problem\_dis, min\_problem\_dis, max\_procedure\_dis, min\_procedure\_dis, max\_category\_dis, \

min\_category\_dis = cal\_normalize\_dis(report\_ids, problem\_widget\_path\_list, text\_feature\_list, widget\_category\_list)

file = open(report\_sim\_matrix\_file\_path, 'w', encoding='utf-8', newline='')

csv\_writer = csv.writer(file)

header = ['index', '-1']

for report\_id in report\_ids:

header.append(str(report\_id))

csv\_writer.writerow(header) # 矩阵的header， 形如 index,-1, [report\_id], [report\_id], [report\_id]

sim\_matrix = np.zeros((len(report\_ids) + 1, len(report\_ids) + 1))

sim\_matrix[0][0] = 1

for i in range(0, len(report\_ids)):

# cal normalized sim between each report and blank report

# 空报告的report\_id为-1

sim = cal\_report\_similarity(text\_feature\_list[i], widget\_category\_list[i],

problem\_widget\_path\_list[i], None, initial\_widget\_categories, '',

report\_ids[i], -1, max\_sift\_fea, max\_problem\_dis, min\_problem\_dis,

max\_procedure\_dis,

min\_procedure\_dis, max\_category\_dis, min\_category\_dis)

# sim between report i and report j is equal to sim between report j and report i.

sim\_matrix[i + 1][0] = sim

sim\_matrix[0][i + 1] = sim

# cal sim between every two report in report list

# 由于第一行和第一列是和空报告做对比，该位置被占据。

# 假设是第零和第一篇report做对比，其结果应该存在相似矩阵的[ 0+1, 1+1 ]

for i in range(0, len(report\_ids)):

for j in range(i, len(report\_ids)):

last\_report\_id\_idx = len(report\_ids) - 1

if i == j:

sim\_matrix[i + 1][j + 1] = 1

# 最后一篇报告的相似度，肯定是1

if i == last\_report\_id\_idx:

save\_totally\_same\_report\_vec(report\_ids[i], report\_vec\_file\_path)

else:

sim = cal\_report\_similarity(text\_feature\_list[i], widget\_category\_list[i], problem\_widget\_path\_list[i],

text\_feature\_list[j], widget\_category\_list[j], problem\_widget\_path\_list[j],

report\_ids[i], report\_ids[j], max\_sift\_fea, max\_problem\_dis,

min\_problem\_dis,

max\_procedure\_dis, min\_procedure\_dis, max\_category\_dis, min\_category\_dis)

if j == last\_report\_id\_idx:

cal\_sim\_vec(text\_feature\_list[i], widget\_category\_list[i], problem\_widget\_path\_list[i],

text\_feature\_list[j], widget\_category\_list[j], problem\_widget\_path\_list[j],

report\_ids[i], report\_ids[j], max\_sift\_fea, max\_problem\_dis, min\_problem\_dis,

max\_procedure\_dis, min\_procedure\_dis, max\_category\_dis, min\_category\_dis,

report\_vec\_file\_path)

sim\_matrix[i + 1][j + 1] = sim

sim\_matrix[j + 1][i + 1] = sim

# record sim result in .csv file

for i in range(0, len(report\_ids) + 1):

index = 0

if i == 0:

index = -1

else:

index = report\_ids[i - 1]

sim = [str(index)]

for j in range(0, len(report\_ids) + 1):

sim.append(str(sim\_matrix[i][j]))

csv\_writer.writerow(sim)

file.close()

**根据矩阵得到相似度**

取矩阵的某一列，就是某一报告相比其他报告的相似度（代码很简单，不演示了）

**推荐**

使用DeepPrior的思想，加入空报告，进行报告推荐，具体流程见上文的截图

**聚类**

由于已经得到了报告的二维向量表示，将向量进行Kmeans聚类即得到结果

def cluster(self, report\_vec\_file\_path):

'''

返回值：

:param report\_vec\_file\_path:

:return: [ { "report\_id": 10,"coordinate": [0.23, 0.87],"cluster\_id": 19 }, ... ]

'''

with open(report\_vec\_file\_path, 'r') as csvfile:

reader = csv.reader(csvfile)

header = next(reader) # 跳过矩阵头

report\_id\_list = []

data = []

for line in reader:

report\_id\_list.append(int(line[0])) # report\_id

line = line[1:] # 去除第一个元素（report\_id）

ele\_float\_list = []

for ele\_str in line:

ele\_float\_list.append(float(ele\_str)) # 将元素从str转成float

data.append(ele\_float\_list)

print(header)

print(data)

X = np.array(data)

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=0).fit(X)

# 输出及聚类后的每个样本点的标签（即类别），预测新的样本点所属类别

label\_list = kmeans.labels\_

label\_list = label\_list.tolist() # 将np.array类型转化为list类型，使其元素可以序列化

print(kmeans.labels\_)

print(kmeans.predict([[0, 0], [4, 4], [2, 1]]))

res\_list = []

for i in range(len(report\_id\_list)):

item = {}

item["report\_id"] = report\_id\_list[i]

item["coordinate"] = [ data[i][0], data[i][1] ]

item["cluster\_id"] = label\_list[i]

# print(type(label\_list[i]))

res\_list.append(item)

print(res\_list)

return res\_list

**算法遇到的困难**

算法实现时遇到的困难是算法的速度，因为要对图片进行下载，ocr，特征提取和相似度计算，算法的速度很慢。 我的解决方案是，将算法拆成两部分： 计算出中间结果； 根据中间结果进行具体算法计算

* 计算出中间结果： 中间结果包括报告的相似度矩阵和报告的向量表示。 性能瓶颈主要在这一步，有了中间结果，后续进行具体算法（聚类、相似度、推荐）速度就很快了， 因此可以把它们序列化存起来，给具体的算法使用
  + 当然，如果某一任务下的任何报告有任何改动，该任务下的报告就需要重新计算其中间结果
* 根据中间结果进行具体算法计算：有了相似度矩阵和向量，再进行推荐、聚类等等就不存在速度问题，这些算法都可以自由调用