用户推荐算法说明

为了节省众包测试的代价，需要找到与待测任务相似度比较高的众包工人，并将这些工人推荐给发包方。为了研究众包工人与任务之间的相似度，首先需要提取众包工人与待测任务的描述性词。其中众包工人的描述性词由众包工人提交的报告获得，待测任务的描述性词由任务本身的描述获得。

首先，对文本进行分词，去除停用词，并且统计词频，构建词向量。

```
public static List<String> segmentWord(String str) {
        if (str == null || str.length() == 0) {
            return new ArrayList<>();
        }

        List<String> words = segmenter.sentenceProcess(str);
//        System.out.println(words);
        return removeStopWords(words);
    }

public static List<String> removeStopWords(List<String> words) {
    ArrayList<String> remainWords = new ArrayList<>();
    for (int i = 0; i < words.size(); i++) {
        if (words.get(i) == null || words.get(i).trim().equals(""))
            continue;
        else if (stopWordList.contains(words.get(i)))  // 去除停用词
            continue;
        else
            remainWords.add(words.get(i).trim());
        }
        return remainWords;
    }
```

其次，对用户的词向量以及任务的词向量计算余弦相似度。

```
public static Double cosinSimilarity(HashMap<String, Integer> vector1,
HashMap<String, Integer> vector2) {
    if (vector1 == null || vector2 == null || vector1.size() == 0 ||
vector2.size() == 0) {
        return 0.0;
    }
    HashSet<String> totalTermList = new HashSet<String>();

    Set<String> keySet1 = vector1.keySet();
    Set<String> keySet2 = vector2.keySet();
    totalTermList.addAll(keySet1);
    totalTermList.addAll(keySet2);

    int v1sum = 0, v2sum = 0, multiply = 0;
    for (String term : totalTermList) {
        int v1 = 0, v2 = 0;
        if (vector1.containsKey(term))
            v1 = vector1.get(term);
        if (vector2.containsKey(term))
```

```
            v2 = vector2.get(term);

        v1sum += v1 * v1;
        v2sum += v2 * v2;
        multiply += v1 * v2;
    }

    double sim = (1.0 * multiply) / (Math.sqrt(1.0 * v1sum) * Math.sqrt(1.0 *
v2sum));
    return sim;
}
```

对众包工人按照与任务从高到低的相似度进行排序，取出前20高的用户用作重排序。因为如果直接选择与任务相似度最高的用户，可能导致发现的bug都是重复的，无法发挥众包测试的工人多样性。于是我们以与任务相似度最高的用户为待返回结果，每次从剩下的用户中选择与中心余弦相似度最小的用户，加入到待返回结果，知道凑足10人。这样既考虑了众包工人与任务的相似度，也考虑了众包工人的多样性。部分代码如下：

```
public List<Integer> findFarthest10(List<UserDescriptor> userDescriptorListTop) {
        List<UserDescriptor> chosen = new ArrayList<>();
        if (userDescriptorListTop.size() <= 10) {
            chosen.addAll(userDescriptorListTop);
        } else {
            UserDescriptor top1 = userDescriptorListTop.get(0);     // 选取与任务最
匹配的用户
            userDescriptorListTop.remove(top1);
            chosen.add(top1);
            while (chosen.size() != 10) {
                UserDescriptor userDescriptor = findFarthest(chosen,
userDescriptorListTop);
                if (userDescriptor == null)
                    break;
                userDescriptorListTop.remove(userDescriptor);
                chosen.add(userDescriptor);
            }
        }
        List<Integer> res = new ArrayList<>();
        for (UserDescriptor userDescriptor : userDescriptorListTop) {
            res.add(userDescriptor.getUserid());
        }
        return res;
    }
```

预测任务的BUG数说明

论文`iSENSE_Completion-Aware_Crowdtesting_Management`中共提出了四种模型来预测系统的bug数。

		Crowdworker's detection capability	
		Identical	Different
Bug detection probability	Identical	M0 (*M0*)	Mt (*MtCH*)
	Different	Mh (*MhJK, MhCH*)	Mth (*Mth*)

按照时序关系对缺陷报告进行采样，并进行分组，识别缺陷bug是否含有bug，并且bug是否重复，然后构造lookup table，以每组缺陷报告为行，以每一个新发现的bug为列，如果该组缺陷报告含有对应列的bug，则记为1，否则记为0。样例如下：

Table I: Example of bug arrival lookup table

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	...
Sample #1	1	1	1	0	0	0	0	0	0	0	0	0	
Sample #2	0	0	1	1	0	0	0	0	0	0	0	0	
Sample #3	0	0	1	0	1	0	0	0	0	0	0	0	
Sample #4	0	0	0	1	1	1	1	1	0	0	0	0	
Sample #5	0	0	1	1	0	0	0	1	1	1	1	0	
Sample #6	1	0	1	0	1	0	0	0	0	0	0	1	
Sample #7	...												

其中（sample #1， #1）记为`1`，表示第一组权限报告发现了bug#1。

我选择了自重模型中的Mh模型，并参考实现了MhCH预测算法。该算法所采用的公式如下：

$$N = D + \frac{f_1^2}{2f_2} \tag{4}$$

or

$$N = D + \frac{[\frac{f_1^2}{2f_2}][1 - \frac{2f_2}{tf_1}]}{1 - \frac{3f_3}{tf_2}}, \text{ if } tf_1 > 2f_2, tf_2 > 3f_3, 3f_1f_2 > 2f_2^2 \tag{5}$$

具体字段的含义说明如下：

字段	含义
N	预测的总bug数
D	已经发现的实际bug数
t	分组样本数
fk	被捕获了k次的bug的数量

利用此模型预测待测软件的bug数，如果发现已找到的bug数已经占预测总bug数百分比达到某个阈值（例如95%），系统会自动通知该任务的发包方，并且允许发包方提前关闭此任务。