软件需求工程实验一 实验报告

1. 小组成员及得分分配

学号	姓名	得分比
181830014	柴致远	20%
181180215	朱景舟	20%
181860061	陆逸麟	20%
181860125	杨震浩	20%
171860588	史文泰	20%

2. 实验目的

- 对于Stack Overflow上的IDE标签进行检索，抽取出用户对于IDE的潜在需求，并对需求进行分类。

3. 实验方法

3.1. 通过爬虫抓取问答

- 首先进入 Stack Overflow 主页，并提取其中问题链接，进入问题页后抓取问题题目、问题描述及回答文本。
    - 注：爬虫自动过滤了投票数为 0 的问题
- 代码及注释如下：

```
# -*- coding:UTF-8 -*-
import requests
import re


def html2plaintext(mystr):  # 去掉html文本中的内容无关部分
    ...
    pass


def html2numstr(mystr):     # 获取html文本中的数字
    ...
    pass


if __name__ == '__main__':
    url_h = 'https://stackoverflow.com/questions/tagged/ide?tab=votes&page='
```

```
    url_t = '&pagesize=50'
    for index in range(70, 71):
        file_path = './results/data' + str(index) + '.txt'
        f = open(file_path, 'w', encoding='utf-8')
        print(index)
        url_m = str(index)
        target = url_h + url_m + url_t
        mainpage = requests.get(target).text

        votes_pattern = '<span class="vote-count-post(.*?)</strong></span>'
        ...

        votes = re.findall(votes_pattern, mainpage, re.S)
        ...

        for (answer, vote, question, suburl) in zip(answers, votes,
questions,suburls):
            answer = html2numstr(answer)
            vote = html2numstr(vote)
            if '[closed]' in question:
                continue
            if int(vote) == 0 or int(answer) == 0:
                continue
            target = 'https://stackoverflow.com/' + suburl
            subpage = requests.get(target).text
            print(target)
            descriptions = re.findall(description_pattern, subpage, re.S)
            anscells = re.findall(anscell_pattern, subpage, re.S)
            ansvotes = re.findall(ansvote_pattern, subpage, re.S)

            anscells = [html2plaintext(ans) for ans in anscells]
            descriptions[0] = html2plaintext(descriptions[0])
            f.write('Title: ' + question + '\n')
            f.write('Description: ' + descriptions[0] + '\n')
            for i in range(0, len(anscells)):
                f.write('Answer ' + str(i + 1) + ': \n')
                f.write(anscells[i])
                f.write('\n')

        f.close()
```

3.2. 整理数据，筛选需求关键词

抓取得到以上的问答内容后，借助python的第三方库NLTK等，筛选其中的关键词，尝试从中提取各类用户对IDE的代表性需求。这里的重点是关键词筛选策略的确定（如，有关IDE的问答中自然地涉及到很多程序代码，需求获取中应该忽略它们），经过小组成员讨论，最终将方案大致确定为：

1. 去除含有【closed】或【duplicate】标签的问答内容、去除数据中人为引入的与问答无关的字符串
2. 舍弃数据中的全部**非英文单词**
3. 保留以上结果中的**名词**（包括单/复数形式）和**动名词**
4. 最后，筛选出出现频次在50左右的词汇

以上即源文件`cloud.py`进行的工作，其代码实现为：

```
from wordcloud import WordCloud
import nltk
from nltk.corpus import wordnet
from textblob import TextBlob
import random

def myfilter(x):
    return x[1] <= 50 and x[1] > 5

if __name__ == '__main__':

    freq = {}
    for idx in range(1,121):
        print(idx)
        text = ''
        filepath = './results/data' + str(idx) + '.txt'
        f = open(filepath, 'r', encoding='utf8').read()
        plain_text = f.lower()
        for line in plain_text.splitlines():
            if 'answer' in line:
                continue
            if '[closed]' in line or '[duplicate]' in line:
                continue
            text += line + ' '
        words = nltk.word_tokenize(text)
        tags = set(['VBG', 'NN', 'NNS'])
        ret = []
        pos_tags = nltk.pos_tag(words)
        for word, pos in pos_tags:
            if pos in tags:
                ret.append(word)
        ret = [word for word in ret if wordnet.synsets(word)]
        for word in ret:
            if word in freq:
                freq[word] += 1
            else:
                freq[word] = 1
        print(len(ret))

    freq = sorted(freq.items(), key=lambda x:x[1], reverse=True)
    top_k = []
    keywords = []
    fp = open('freq.txt', 'w', encoding='utf8')
    for x in freq:
        if myfilter(x):
            for xx in range(0, x[1] + 1):
                keywords.append(x[0])
            fp.write(x[0]+'\t'+str(x[1])+'\n')
    print("Almost done ...")
    random.shuffle(keywords)
    woc = WordCloud().generate(' '.join(keywords))
    image = woc.to_image()
    image.show()
```

下图显示了数据处理结果（部分）：

```
simple  50
domain  50
administrator   50
ios 50
gnu 50
editions    50
office  50
curve   50
nil 50
catch   50
store   50
fan 49
means   49
browsers    49
upgrading   49
displaying  49
runner  49
chance  49
sites   49
pair    49
cloud   49
inspections 48
drag    48
occurrences 48
renaming    48
mine    48
wrap    48
clipboard   48
benefits    48
buffers 48
logic   48
cli 48
considering 48
choices 48
bookmark    48
blackberry  48
apis    47
collapse    47
friend  47
lists   47
nature  47
sequence    47
protocol    47
rule    47
creation    47
boost   47
colour  47
notepad 47
binaries    47
supporting  46
fun 46
```

3.3. 对关键词进行分类

- 人工分类：
- **使用聚类算法**：
  - 采用主题模型(`Topic Model`)和**LDA**算法：主题模型是以**非监督学习**的方式对文档的**隐含语义结构**进行**聚类**的统计模型。主题模型认为在词与文档之间没有直接的联系，它们应当还有一个维度将它们串联起来，主题模型将这个维度称为**主题**。每个文档都应该对应着一个或多个的主题，而每个主题都会有对应的词分布，通过主题，就可以得到每个文档的词分布。依据这一原理，就可以得到主题模型的一个核心公式：

$$p(w_i|d_j) = \sum_{k=1}^{K} p(w_i|t_k) \times p(t_k|d_j)$$

    在一个已知的数据集中，每个词和文档对应的$p(w_i|d_j)$都是已知的。而主题模型就是根据这个已知的信息，通过计算$p(w_i|t_k)$和$p(t_k|d_j)$的值，从而得到主题的词分布和文档的主题分布信息。而要得到这个分布信息，现在常用的方法就是**LSA**和**LDA**。其中LSA主要是采用**SVD**的方法进行暴力破解，而LDA则是通过**贝叶斯学派**的方法对分布信息进行拟合。
  - LDA算法假设文档中主题的先验分布和主题中词的先验分布都服从狄利克雷分布。通过**LDA**算法，我们得到了文档对主题的分布和主题对词的分布，接下来就是要利用这些信息来对关键词进行抽取。在我们得到主题对词的分布后，也据此得到词对主题的分布。接下来，就可以通过这个分布信息计算文档与词的相似性，继而得到文档最相似的词列表，最后就可以得到文档的关键词。LDA的训练是根据现有的数据集生成文档-主题分布矩阵和主题-词分布矩阵，`Gensim`中有实现好的训练方法，直接调用即可。
  - 从前述爬取的数据中选择部分数据文件，提取出其中的`Description`部分，并存放到一个文件`text.txt`中，每个`Description`一条。对于`text.txt`进行停词删除，聚类分析，并取相关度前**10**的单词，得到大致需求。

- 代码如下：

```
if __name__ == '__main__':
    stop = set(stopwords.words('english'))
    # for w in ['!', ',', '.', '?', 'Android', 'Studio', 'studio',
'How', 'Eclipse', 'IntelliJ', 'IDEA',
    #           'PyCharm', 'JetBrains', 'Xcode']:
    #     stop.add(w)
    exclude = set(string.punctuation)
    lemma = WordNetLemmatizer()
    # 读入训练样本
    file_object = open('text.txt', encoding='utf-8')
    line = file_object.readline()
    doc_list = []
    while line:
        doc_list.append(line)
        line = file_object.readline()
    # print(text)
    doc_clean = [clean(line).split() for line in doc_list]

    dictionary = corpora.Dictionary(doc_clean)
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
    Lda = models.ldamodel.LdaModel
    ldamodel = Lda(doc_term_matrix, num_topics=10, id2word=dictionary,
passes=50)
    print(ldamodel.print_topics(num_topics=10, num_words=1))
```

4. 实验结果及效果分析

- 人工分类：
- 使用聚类算法：
  - 余弦值前10大：

```
C:\Users\TamakiRinko\AppData\Local\Programs\Python\Python37\python.exe D:/Myfolder/Desktop/StackOverflow/TopicModel.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\TAMAKI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.653 seconds.
Prefix dict has been built successfully.
LSI模型结果:
shortcut/one/method/split/phpstorm/file/delete/block/line/studio/
LDA模型结果:
and/existing/console/importing/one/list/jump/next/within/module/
```

  - `Topic`中取**2**个关键字前10相关：

```
C:\Users\TamakiRinko\AppData\Local\Programs\Python\Python37\python.exe D:/Myfolder/Desktop/StackOverflow/TopicModel.py
[(0, '0.044*"eclipse" + 0.030*"line"'), (1, '0.029*"line" + 0.029*"shortcut"'), (2, '0.037*"project" + 0.025*"studio"'), (3, '0.039*"code" + 0.020*"would"'),
 (4, '0.043*"file" + 0.015*"ide"'), (5, '0.030*"eclipse" + 0.023*"find"'), (6, '0.058*"shortcut" + 0.035*"way"'), (7, '0.033*"eclipse" + 0.025*"using"'), (8,
 '0.058*"one" + 0.039*"designer"'), (9, '0.045*"lisp" + 0.030*"im"')]
```

  - `Topic`中取**1**个关键字前10相关：

```
C:\Users\TamakiRinko\AppData\Local\Programs\Python\Python37\python.exe D:/Myfolder/Desktop/StackOverflow/TopicModel.py
[(0, '0.032*"eclipse"'), (1, '0.028*"would"'), (2, '0.052*"shortcut"'), (3, '0.053*"way"'), (4, '0.024*"eclipse"'), (5, '0.036*"using"'), (6, '0.044*"line"'),
 (7, '0.029*"eclipse"'), (8, '0.034*"project"'), (9, '0.024*"eclipse"')]
```

  - 由于每个需求的长度较短，且训练样本数量较少，最终聚类结果并不明显，但不难可以看出`eclipse`和`android studio`和`visual studio`是需求较大的`ide`，且`shortcut`快捷键，`line`代码行问题拥有较大的用户需求。

5. 结论

经过分析，可以得出如下结论：

- `eclipse`和`android studio`和`visual studio`是大多数用户常用的`ide`，大部分用户的问题都集中在这些编辑环境中，此外`phpstorm`也有可观数量的用户使用；
- `delete`、`shortcut`等`ide`中常见的操作是很多用户有问题的地方，可以编写这些操作的详细说明书；
- 很多用户对 `importing` (python 中的一个关键字) 并不了解；
- `line` 代码行问题有较大量的用户需求。