实验一：软件需求的抽取与分类

小组成员：孔令迪 王珊 陈程

成绩分配比例： 33% 33 % 33%

实验仓库：https://github.com/Shannju/NJU-2022-Software-Requirements-Engineering

实验目的

抽取软件需求并进行分类

实验方法

通过其“IDE”标签对stackoverflow进行搜索，找到大量的有关IDE的问答
通过爬虫抓取这些问答，然后对这些问答进行分析，从而获取潜在的需求。

- 将每个问答作为一个需求，直接对问答进行分类。
- 对大量的问答文本进行词云分析，提取出若干关键词作为潜在需求。然后再对这些关键词进行分类（聚类）。

一. 数据获取

1.1 工具

- Python 3.9.7
- Mac OS 12.6
- 芯片：Apple M1 Pro
- Scrapy, Pandas

1.2 数据来源

我们从stack overflow中搜索关键词”ide”后得到许多网页，其中包括多种多样的问题和回答，但是鉴于本次实验的时间和机器内存等有限，且过于靠后的网页的问题和回答往往质量和数量不如排名靠前的网页高，所以采取爬取前50页的问答内容。

1.3 代码

由于无法直接在爬取搜索结果页面信息的同时，爬取该页中每个问答所对应界面的数据的原因，我们进行两次爬取。先爬取搜索”ide”结果页面中每个问答的link并存储，之后再根据link爬取每个问答的信息。

1.3.1 爬取问答网页的link

首先根据搜索”ide”得到的第一页，得到1 ~ 50页的网址信息，然后对每一页进行爬取

```
    def start_requests(self):
        _url = 'https://stackoverflow.com/questions/tagged/ide?tab=newest&page={}&pagesize=15'
        urls=[]
        for i in range(1,50):
            urls.append(_url.format(i))

        for url in urls:
            yield scrapy.Request(url=url,callback=self.parse)
```

对每一页，爬取该页中的所有问题、网址链接即id

```
def parse(self, response):
        qlist=response.xpath('//*[@id="questions"]')

        for i in qlist.xpath('./div'):
            item=Spider1Item()
            item['id']=i.attrib['id']
            item['question']=i.xpath('div[2]/h3/a/text()').extract()
            item['link']=i.xpath('div[2]/h3/a/@href').extract()
            yield item
```

将爬取到的信息以表格形式输出到一个link.csv文件中

```
class Spider1Pipeline:
    def __init__(self):
        self.itemall =DataFrame()

    def process_item(self, item, spider):
        info=DataFrame([item['id'],item['question'][0],item['link'][0]]).T
        info.columns=['id','question','link']
        self.itemall=pd.concat([self.itemall,info])
        self.itemall.to_csv('link.csv',encoding='utf-8')
        return item
```

1.3.2 爬取问答

根据上述爬取的link，依次进行每一页问答的数据的爬取

```
class ItcastSpider(scrapy.Spider):
    name = 'itcast1'

    def start_requests(self):
        urls=[]
        csv_reader = csv.reader(open("link.csv"))
        for line in csv_reader:
            if(line[3][-1]=="\n" or line[3][-1]=="\r"):
                urls.append('https://stackoverflow.com'+line[3][:-1])
            else:
```

```
                urls.append('https://stackoverflow.com'+line[3])
        proxy = ["http://207.236.12.190:80",
                "http://51.75.165.14:80",
                "http://103.172.116.231:80",
                "http://104.148.36.10:80"]
        cnt=0
        for url in urls:
            sleep(0.1)
            yield scrapy.Request(url=url,callback=self.parse,
                                 #  errback=self.errback_httpbin,
                                  dont_filter=True,
                                  meta={"proxy": proxy[cnt]}
                                  )
            cnt=(cnt+1)%4

```

注：鉴于stack overflow等大型网站都会防止恶意爬虫，所以我们使用了4个测试过可用的代理，并且选择每爬取一次休息0.1秒钟

解析爬取到的每一页html文件，找到question和answer，由于一个问题下可能有多个回答、评论，而后续数据分析等操作是对文本的处理，而与answer的个数无关，所以我们把每一个回答都添加到一个string中，设置为总的answer，并用“...”隔开

```

    def parse(self, response):
        sel1 = response.xpath('//*[@class="postcell post-layout--right"]/div[1]')
        item = Spider2Item()
        pro = sel1.xpath(
            'string(.)').extract()[0]
        pro.replace("\r","")
        pro.replace("\n"," ")
        item['question'] = pro
        #多个回答
        sel2s = response.xpath('//*[@class="answercell post-layout--right"]/div[1]')
        sol = sel2s[0].xpath(
            'string(.)').extract()[0]
        if(len(sel2s)) >= 2:
            for i in range(1, len(sel2s)):
                sol = sol + "..."
                sol = sol + sel2s[i].xpath(
                'string(.)').extract()[0]
        sol.replace("\r","")
        sol.replace("\n"," ")
        item['answer'] = sol
        item['link']=response.url
        yield item
```

之后以每一行为“question,link,answer”形式输出到“qa.txt”文件中方便后续使用

```

class Spider2Pipeline:
```

```
    def __init__(self):
        self.itemall =DataFrame()

    def process_item(self, item, spider):
        que=item['question'].replace("\n","")
        que=item['question'].replace("  ","")
        link=item['link'].replace("\n","")
        link=item['link'].replace("  ","")
        ans=item['answer'].replace("\n","")
        ans=item['answer'].replace("  ","")
        info=DataFrame([que,link,ans]).T
        info.columns=['question','link','answer']
        self.itemall=pd.concat([self.itemall,info])
        self.itemall.to_csv('qa.txt',encoding='utf-8')
        return item

```

注：由于在question,link,answer中会现大量“\n”“\r“不包含信息但是影响格式的字符，所以选择用空字符代替

二. 需求分析

2.1 聚类分析

2.11 工具

- Python 3.7.6
- Windows10
- AMD Radeon显卡
- xiangshi工具包

2.12 数据来源

用爬虫技术获取的数据

2.13 技术和代码

将每个问答作为一个需求，用kmeans算法对文本进行聚类并获得每个聚类中心，也即可代表属于同一类问答的一条问答。随后获取聚类中心问答中潜在的需求。

聚类代码如下，使用了xiangshi工具包，输入为txt文件，其中每个问答占一行。计算文本间的欧几里得距离，自动TFIDF加权，将输入的所有问答聚成k类并获得聚类中心，将聚类中心写入center.txt文件，分析center.txt文件中的问答即可。

```
import xiangshi as xs

fn = 'qa.txt'
doc = []
with open(fn, 'r', encoding='utf-8') as f:
    for line in f.readlines():
        doc.append(line)

res = xs.kmeans(10, doc, WithKeys=True)
fn = 'data_.txt'
fn1 = 'center.txt'
```

```
idx0 = ''
with open(fn, 'w', encoding='utf-8') as f:
    for idx, sentence in res.items():
        if idx != idx0:
            idx0 = idx
            f.write('\n')
            with open(fn1, 'a', encoding='utf-8') as f1:
                f1.write(idx)
                f1.write('\n')
        for i in range(len(sentence)):
            f.write(sentence[i])
            f.write('\n')

```

2.14 结果

获得的聚类中心问答的示例之一如下：

> I'd like to be able to visually see the relationships of the files I require from every file in my project - that's because I'm using redux in my project, but there are probably cases where I introduced anti-pattern behaviour, by running code without dispatching an action creator first, and I'd love to visually check how complicated things are at this point.Example: If A depends on B, and B depends on C, I'd love to visually see A with an arrow connected to B, with an arrow connected to C.Is there any util, or IDE functionality to help me visualise my js code like that?Thank you I've used madge before: npm link with install instructions. If you also install graphviz, you can export an svg of your dependencies with madge --image graph.svg path/to/app.js.See example of one of my more modular repos!

将聚类组数设定为20，可获得20个聚类中心

2.2 词云分析

2.21 工具

- Python 3.8.10
- Windows10
- sklearn pandas nltk pyecharts

2.22 主要内容

- 1、 对数据进行初步整理
- 2、 用正确的方式完成数据清洗
- 3、 对问答文本内容进行合理分词
- 4、 根据分词结果建立词向量完成聚类，根据聚类结果形成词云，对于IDE软件需求进行分析

2.23 核心实现

文本分词&&数据预处理

使用nltk分词器对爬取的问答进行分词。

对于分词结果，在去除标点符号， 词形还原，去除英文常用停用词的基础上，针对计算机常用词汇构建专业名词停用词表。

> 为本项目构建的专业名词停用词表"english stopwords.txt"。包含了编程常用名词：如file/code/project/work/program，语言关键字：如int/use/include等一系列词汇。用以保障筛选结构的信息有效程度。

```
def deal_eng(data):
    with open("english stopwords.txt", encoding='utf-8') as f:
        stopwords_list = f.read()
```

```
    cutwords1 = word_tokenize(data)  # 分词
    # print (cutwords1)

    interpunctuations = [',', ' ', '.', ':', ';', '?', '(', ')', '[', ']', '&', '!', '*', '@', '#',
                         '$',
'%','//','=','{','}','...','``','i','1','2','3',"''","'s",'>','<','0',"'",'-','+','--','...','/','use']  # 定义符号列表
    cutwords2 = [word for word in cutwords1 if word not in interpunctuations]  # 去除标点符
    # print(cutwords2)
    cutwords25 = [word for word in cutwords2 if word not in stopwords_list] # 去除标点符号
    stops = set(stopwords.words("english"))
    cutwords3 = [word for word in cutwords25 if word not in stops]  # 判断分词在不在停用词列表内
    # print(cutwords3)
    tagged_words = pos_tag(cutwords3)
    wnl = WordNetLemmatizer()
    cutwords4 = []
    for tag in tagged_words:
        wordnet_pos = get_wordnet_pos(tag[1]) or wordnet.NOUN
        cutwords4.append(wnl.lemmatize(tag[0], pos=wordnet_pos))  # 词形还原
    return cutwords4
```

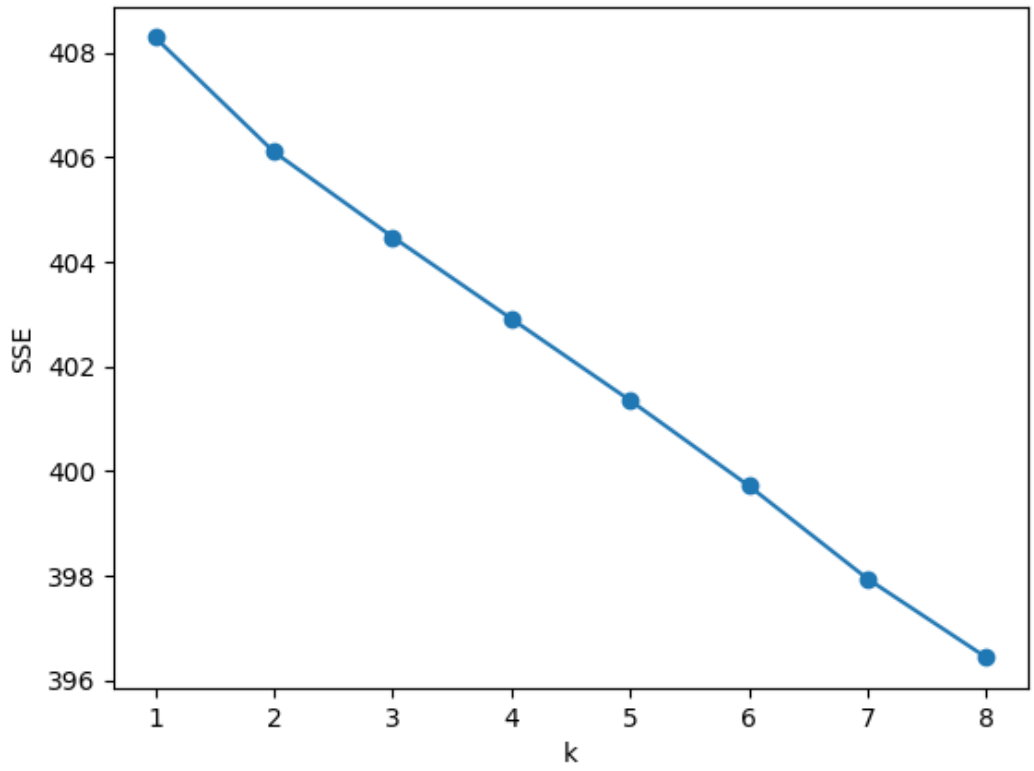
聚类+词云

> K-means 聚类分析算法又称群分析，它是研究（样品或指标）分类问题的一种多元统计方法，所谓类就是指相似元素的集合。

(1)首先分词后的内容进行词向量的转化

```
#词向量转换
count_vect = CountVectorizer()
X = count_vect.fit_transform(df['danmu_text'])

#tf-idf
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X)

```

(2)用SSE画出K值的图，用来选择最优的K值



(3)创建kmeans模型，传入词向量进行训练

```
# K均值聚类
model_kmeans = KMeans(n_clusters=8,random_state=1)  # 创建聚类模型对象
model_kmeans.fit(X_tfidf)  # 训练模型

# 聚类结果
cluster_labels = model_kmeans.labels_  # 聚类标签结果
print(cluster_labels)
```

(4)把聚类生成的结果与question&&answer对应

```
labels=pd.DataFrame(cluster_labels,columns=['标签'])
shuju=pd.concat([df['question'],labels],axis=1)
print(shuju)
```

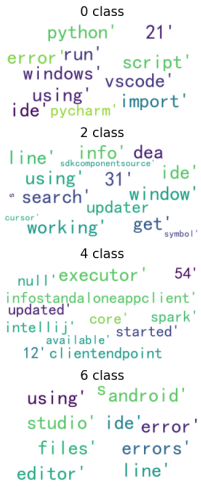
(5)查看部分类别的词汇

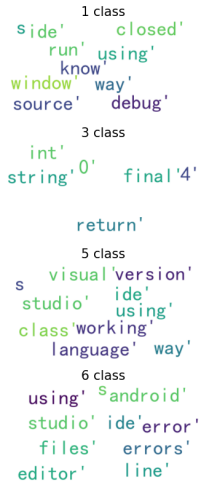
部分2类词汇表如下，可以作为分析参考

```
['virtual', 'emulator', 'default', 'dock', 'ide', 'instead', 'floating',
'window', 'thus', 'changed', 'view', 'mode', 'float', 'later', 'window',
'seems', 'scrolling', 'longer', 'works', 'properly', 'setup', 'believe',
'bug', 'going', 'based', 'experience', 'getting', 'whenever', 'scroll',
'using', 'mouse', 'virtual', 'device', 'disappear', 'running', 'prompt',
'user', 'device', 'already', 'running', 'emulator', 'go', 'tab', 'either',
'.', 'idea', 'switch', 'contextual', 'help', 'covers', 'line', 'working',
'covers', 'auto-complete', 'super', 'annoying', '.', 'somewhat', 'recently',
'using', '``', 'delete', 'word', 'end', 'functionality', 'intellij',
```

(6)把聚类产生的结果，然后统计词频，可以画出词云图，方便进行分析

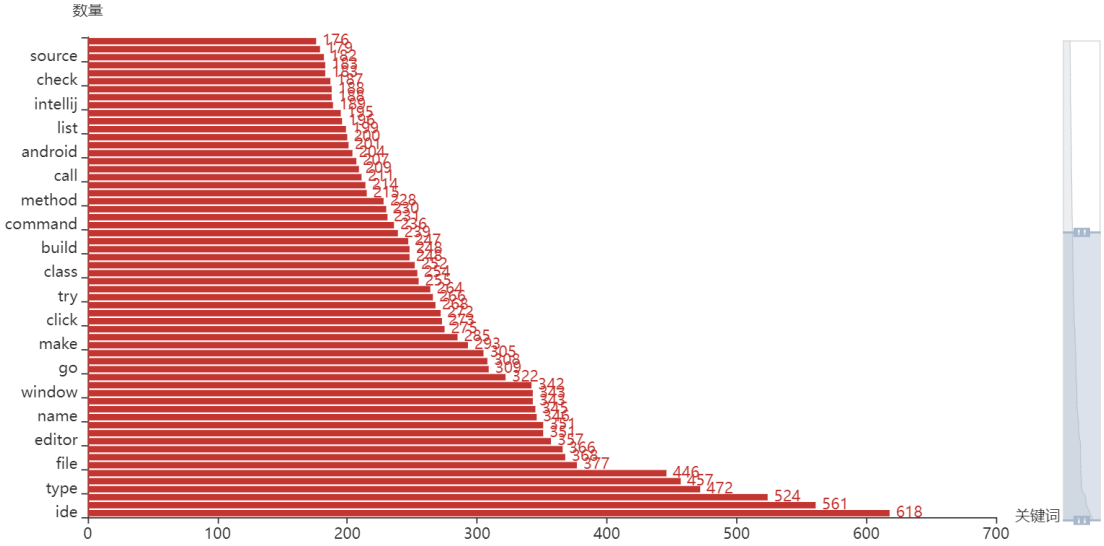
```
def render_cloud(word_counts_top100):
    word1 = WordCloud(init_opts=opts.InitOpts(width='1350px', height='750px', theme=ThemeType.MACARONS))

    word1.add('词频', data_pair=word_counts_top100,

              word_size_range=[15, 108], textstyle_opts=opts.TextStyleOpts(font_family='cursive'),

              shape=SymbolType.DIAMOND)

    word1.set_global_opts(title_opts=opts.TitleOpts('评论云图'),

                          toolbox_opts=opts.ToolboxOpts(is_show=True, orient='vertical'),

                          tooltip_opts=opts.TooltipOpts(is_show=True, background_color='red', border_color='yellow'))

    # 渲染在html页面上

    word1.render("评论云图.html")

def render_bar(word_counts_top50):

    x=[i[0] for i in word_counts_top50]
    y = [i[1] for i in word_counts_top50]
    print (x)

    keywords_count_bar = (
        Bar()
            .add_xaxis(x)
            .add_yaxis("",y)
    # .add_yaxis(“”， list(keywords_count_dict.values())) 可以形成两个柱状图在一个坐标系中
    .reversal_axis()  # 反转坐标轴
        .set_series_opts(label_opts=opts.LabelOpts(position="right"))  # 柱状图坐标上数字显示，不写默认为right
        .set_global_opts(
        title_opts=opts.TitleOpts(title="title"),  # 标题
        yaxis_opts=opts.AxisOpts(name="数量"),  # y轴
        xaxis_opts=opts.AxisOpts(name="关键词"),  # x轴
        datazoom_opts=opts.DataZoomOpts(type_="slider",orient='vertical')
    )
    )
    keywords_count_bar.render('title-word-count-bar.html')

```





2.4 数据可视化

除了前10个词云聚类之外，本项目使用pyecharts构建柱状图和词云，以便数据可视化。



完整图表见仓库[title-word-count-bar.html]

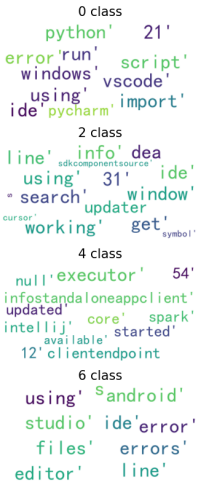
完整图表见仓库[评论云图.html]

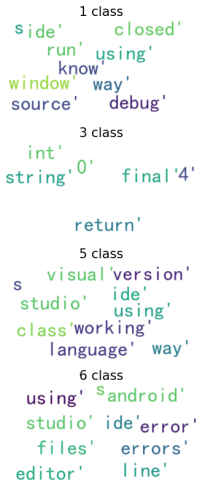
三. 需求总结

3.1 分析方法

- 将每个问答作为一个需求，直接对问答进行分类 （见2.1 聚类分析）
- 对大量的问答文本进行词云分析，提取出若干关键词作为潜在需求。然后再对这些关键词进行分类（聚类）。（见2.3 词云分析）

3.2 分析过程





参考生成的前十类问答聚类，可知

class0：

针对python Ide的问题较多，考虑到python应用的广泛以及上手的简易性，可以分析出，python作为针对新手的语言，使用人数多，使用者倾向于上网求助也多。针对python IDE的需求应该是新手友好的。

class1：

针对debug存在需求，用户对于窗口有一个直观的感受，窗口的出现和消失以及debug的错误定位有相关需求。

class2：

对于版本管理存在需求

class4：

关键词execute，对于编译运行存在需求

class5：

关键词language，对于语言管理存在需求

3.2 分析结果

从以上两种方法中，分析IDE可有如下需求：

（1）功能需求：

- 1.代码编辑
  - 1）自动补全：变量名、类型名在编辑时，有自动补全的可选项
  - 2）缩进处理：单行或多行的缩进和取消缩进
- 2. 调试
  - 1）模块化调试：可单独调试某一函数
  - 2）错误定位：报错行以及warnings行的定位功能，可在每条报错和warning间切换
- 3. 编译
  - 1）工具库：包含更完整的工具库，并能正确定位到所需的工具库
  - 2）完善编译器的语法解析功能，如减少无缘报错的情况发生
  - 3）统一缓存的实现，使得同样的代码在不同IDE中有相同的运行结果
- 4. 其他
  - 1）提供可视化工具，如可视化项目用到的文件的联系、类之间的关联等；
  - 2）提供更简洁的代码风格配置功能；

（2）非功能需求

- 1. 兼容性
  - 1）可在跨系统方面更加包容和完善，如在Windows系统上使用bash
  - 2）可支持打开多种格式文件，也可支持将文件导出为多种格式；
- 2. 快捷键
  - 如在错误定位中引入快捷键，也可自行定义