# 文本检索大作业报告-王田雨2000012903

本次大作业耗时3天完成，选择了推荐的英文数据集，我的作业流程大致如下。

## 完成server-client框架

```
while True:
    new_socket,client_info = server.accept()
    p = Thread(target = self.serve,args = (new_socket,client_info))
    p.start()
```

实现多线程处理。

```
def wrap(data):
    return bytes(json.dumps(data).encode('utf-8'))
```

借助json定义wrap函数来封装传输数据

```
def serve(self,new_socket,client_info):
    '''处理客户端发送来的数据'''
    print("client{} is connected".format(client_info))
    raw_data = new_socket.recv(1024)
    while raw_data:
        data = json.loads(raw_data)
        print('data is {}'.format(data))
        new_socket.send(wrap(self.find(data)))
        raw_data = new_socket.recv(1024)
    new_socket.close()
    print("client{} is unconnected".format(client_info))
```

进行server-client交互

## 进行数据处理

```
self.data = pd.read_csv('data/all_news.csv')
self.doc = pd.read_csv('data/all_news.csv')
self.stop_words = set(stopwords.words('english'))
self.remove = str.maketrans('','',string.punctuation)
self.lemmatizer = WordNetLemmatizer()
```

进行一些初始化操作

```
def ccut(self,x):
    '''正则化、去除无关'''
    for i in range(1,2):
        x[i] = re.sub('[^A-Za-z]+', ' ', x[i]).lower()
        x[i] = x[i].translate(self.remove)
        x[i] = word_tokenize(x[i])
        ls = []
        for w in x[i]:
            if w not in self.stop_words:
                ls.append(self.lemmatizer.lemmatize(w))
        x[i] = ls
    return x
def getdt(self,x):
    '''get word count dt'''
    for i in range(1,2):
        for word in x[i]:
            if word in self.dt:
                self.dt[word] += 1
            else:
                self.dt[word] = 1
    return x
```

```
self.data = self.data.apply(self.ccut,axis = 1)
self.dt = {}
self.data = self.data.apply(self.getdt,axis = 1)
```

对数据进行apply，得到（去标点，停用词后）分好的词组，以及统计数据集整体词频

```
for x in list(self.dt.keys()):
    if self.dt[x] < 20:
        del self.dt[x]
print('dict.length is {}'.format(len(self.dt)))
```

筛掉低频词语，完成词典构建

## 检索排序

```
self.data['setwords'] = self.data.apply(self.getset,axis = 1)
self.pos = {}
ls = list(self.dt.keys())
for i in range(len(ls)):
    self.pos[ls[i]] = i
    self.dt[ls[i]] = self.getidf(ls[i])
    #print(self.dt[x])
self.data['TF-IDFvec'] = self.data.apply(self.getTF_IDF,axis = 1)
```

```
def getTF_IDF(self,x):
    '''get tf-idf'''
    dt = {}
    for word in x[1]:
        if word in dt:
            dt[word] += 1
        else:
            dt[word] = 1
    ls = []
    for word in list(self.dt.keys()):
        if word in dt:
            ls.append(dt[word] / len(x[1]) * self.dt[word])
        else:
            ls.append(0)
    a = np.array(ls).astype('float')
    a = a.reshape((1,len(ls)))
   # print(a.shape)
    return a
```

求取文章tf-idf向量

```
def evaluate(self):
    '''实现聚类，计算purity'''
    print('local server is evaluating...')
    #print(self.data['TF-IDFvec'].shape)
    vec = np.hstack(self.data['TF-IDFvec']).reshape((len(self.data),len(self.dt)))
    vec = preprocessing.scale(vec)
    km = KMeans(n_clusters=5,random_state=666).fit(vec)
    pred = km.predict(vec)
    orig = []
    s = set()
    for i in range(len(vec)):
        s.add(self.data['topic'][i])
        orig.append(len(s) - 1)
    g = np.zeros((10,10))
    for i in range(len(pred)):
        g[pred[i]][orig[i]] += 1
    purity = np.sum([np.max(g[i]) for i in range(len(g))]) / len(pred)
    print("the purity is {}".format(purity))
```

实现聚类评估，purity在0.6-0.7之间

```
  ls = []
  v = np.zeros(len(self.dt))
  for w in terms:
      if w in self.pos:
          v[self.pos[w]] = (terms.count(w)) / len(terms) * self.getidf(w)
  for i in article:
      ls.append((i,self.cos(self.data['TF-IDFvec'][i],v)))
  ls.sort(key = lambda x : -x[1])
  ret = []
  for x in ls[:10]:
      ret.append((self.doc['title'][x[0]],self.doc['body'][x[0]]))
  return ret
```

将相似词筛选后的文章集合按照与输入terms tf-idf向量的cos similarity排序，选择最相似的十条返回

# 相似词

```
self.writevocab()
self.writesimilar()
#self.data.to_csv('cut.csv')
```

```
def writevocab(self):
    with open('vocab.txt','w') as f:
        for w in self.dt.keys():
            f.write("{}\n".format(w))
def writesimilar(self):
    with open('synonym.txt','w') as f:
        ls = list(self.dt.keys())
        for i in tqdm(range(len(ls))):
            f.write('{} : {}\n'.format(ls[i],self.getsimilarword(ls[i])))
```

相关写操作，完成vocab.txt，synonym.txt。

```
def getsimilarword(self,w):
    '''在词典中找到相似词'''
    ls = []
    for x in self.dt.keys():
        if fuzz.ratio(x,w) >= 80:
            ls.append(x)
    return ls
```

使用fuzzywuzzy中的fuzz进行简单相似词匹配，这里的阈值采用80

```
similarterms = []
for w in terms:
    print("x is {} sim is \n".format(w))
    print(self.getsimilarword(w))
    similarterms += self.getsimilarword(w)
article = []
```

```
def include(self,data,terms):
    '''判断文章x是否含有搜索的关键字以及相似词'''
    for w in terms:
        if w in data['setwords']:
            return True
    return False
```

依据相似词拓展搜索集合，这里采用如果文章A包含词条t的相似词，那么将A加入待选集合