电类工程导论小组作业报告——商品推荐搜索引擎

516030910511 田畅达

2018 年 1 月 13 日

摘要

经过一个学期的电类工程导论C的课程学习，我们学到了写网络爬虫，利用网络爬虫搜集互联网上海量的信息。还学到了用Lucene制作搜索引擎。另外，我们还学到了一些图像处理的方法。最后，我们完成了一个搜索引擎的小组作业。我们利用所学知识，完成了一个商品推荐搜索引擎。我们自己写爬虫，从京东商城，苏宁易购上面爬取了12万条商品信息，包括商品标题，价格，商品图片，商品购买链接，商品好评率，商品评价等信息。这个搜索引擎有用户账户系统，不同用户注册后登陆本搜索引擎，我们可以根据他们注册时填写的信息和他们的搜索历史记录为他们推荐商品，帮助他们更快地找到自己需要的商品。同时，我们还利用Keras 机器学习框架制作了图片识别的搜索功能，我们可以实现上传一张图片，把这张图片的内容识别出来，搜索引擎返回用户图片中商品的搜索结果。

目录

1 商品信息爬取

简单的基本方法就不再这里赘述，主要说一说遇到的困难和解决方法。

1.1 价格，好评率，评价动态呈现问题

经过我们的观察，京东，苏宁等购物网站商品的价格，好评率，评价等信息不是静态写在网页中的，而是通过代码请求得到的。因此，一般的爬取方式是不能爬到这些信息的。我们采用的处理方法是利用浏览器的开发者工具，打开network那个功能，刷新网页，看看网页与其他网页的信息交流情况。找到一些请求信息的返回url打开它们，就能发现一些包含了价格信息，一些包含了好评率和评价信息。然后，观察那些url的结构，发现规律。我们发现京东和苏宁的商品都有自己的一个代码，请求价格的url就包含了这些代码，只需要不同商品的代码就可以利用这些url的结构查到相应价格和好评率了。

1.2 封IP问题

我们发现，京东商城爬取的商品数目一旦变多，京东就会封我们的IP地址。我们就用了每隔一段时间自动修改IP地址的方法，并且动用了多台电脑进行爬取，才得到了我们所需的数据量。

2 商品推荐算法

2.1 基于人口统计学的推荐

基于人口统计学的推荐机制是一种最易于实现的推荐方法，它只是简单的根据系统用户的基本信息发现用户的相关程度，然后将相似用户喜爱的其他物品推荐给当前用户。

首先，我们在新用户注册的时候，会统计用户的信息。然后系统会对每个用户都有一个用户Profile的建模，其中包括用户的基本信息，例如用户的年龄，性别，爱好，职业等等；然后，系统会根据用户的Profile 计算用户的相似度，可以看到用户A 的Profile 和用户C 一样，那么系统会认为用户A 和C 是相似用户，在推荐引擎中，可以称他们是“邻居”；最后，基于“邻居”用户群的喜好推荐给当前用户一些物品。

这种基于人口统计学的推荐机制的好处在于：

- 因为不使用当前用户对物品的喜好历史数据，所以对于新用户来讲没有“冷启动（Cold Start）”的问题。
- 这个方法不依赖于物品本身的数据，所以这个方法在不同物品的领域都可以使用，它是领域独立的（domain-independent）。

然后，这个方法的缺点和问题就在于，这种基于用户的基本信息对用户进行分类的方法过于粗糙，尤其是对品味要求较高的领域，比如图书，电影和音乐等领域，无法得到很好的推荐效果。另外一个局限是，这个方法可能涉及到一些与信息发现问题本身无关却比较敏感的信息，比如用户的年龄等，这些用户信息不是很好获取。

2.2 基于内容的推荐

基于内容的推荐是在推荐引擎出现之初应用最为广泛的推荐机制，它的核心思想是根据推荐物品或内容的元数据，发现物品或者内容的相关性，然后基于用户以往的喜好记录，推荐给用户相似的物品。这种推荐系统多用于一些资讯类的应用上，针对文章本身抽取一些tag作为该文章的关键词，继而可以通过这些tag来评价两篇文章的相似度。

我们的推荐系统根据我们爬取的商品信息提取商品的关键词，做为tag加入要索引的域内，在向用户推荐商品时，有一项推荐是按tag搜索，以此来实现基于内容的推荐算法。

这种推荐系统的优点在于：

- 易于实现，不需要用户数据因此不存在稀疏性和冷启动问题。
- 基于物品本身特征推荐，因此不存在过度推荐热门的问题。

  然而，缺点在于抽取的特征既要保证准确性又要具有一定的实际意义，否则很难保证推荐结果的相关性。

2.3 基于用户的推荐

我们对每一个用户使用我们的搜索引擎的行为都有记录，包括其搜索历史，访问历史。然后我们根据这些历史记录在我们的数据库中搜索相应tag来达到基于用户的推荐。

2.4 效果展示

图 1: 登录页面

图 2: 起始页面

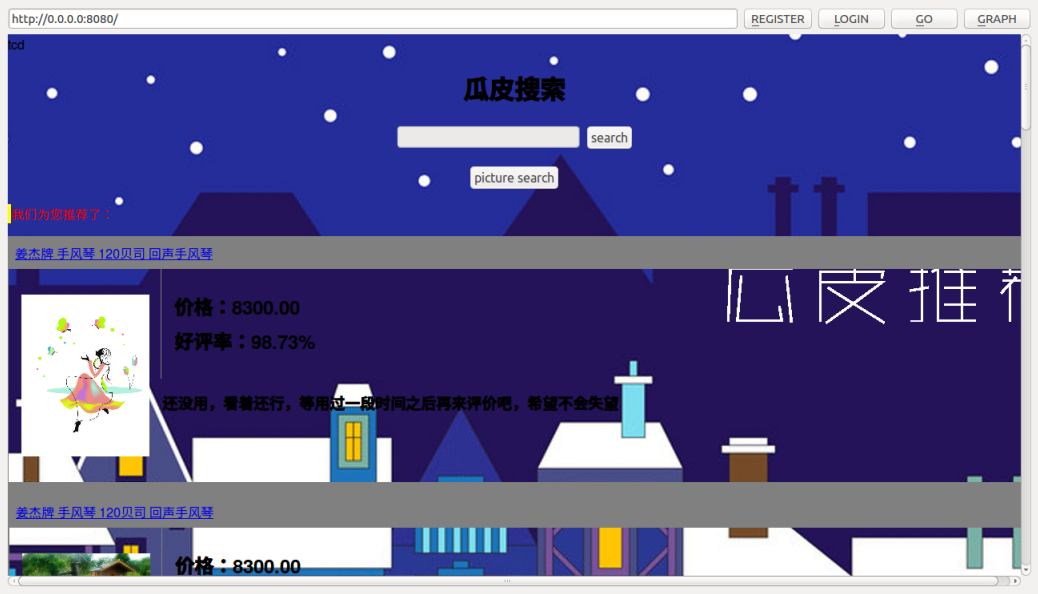


图 3: 搜索结果页面

3 网页构建

借助中期整合的学习经验，利用POST和GET的方法，实现了网页与python之间的连接，利用html网页制作出了简易的网页。刚开始进入的页面为login，提醒用户登录，如果用户登录错误会继续留在这个页面，登录成功会跳转到搜索页面，如果没有账号，会提醒用户注册，注册需要填写用户的一些个人信息，注册成功自动跳转到搜索页。在搜索页页面，我们根据冷启动算法会推荐出用户可能喜欢的产品，并且有注销的按钮，注销后跳回登录页面。搜索页面我们有用户搜索的结果和我们推荐的结果。在其中重要的是我们的程序实现了登录功能和保存登录状态的功能，保存登录利用html的localhost实现储存和注销，功能主要是为了记录用户的信息和用户搜索记录来实现推荐功能，得到结果后利用python的pickle包(可

以将python的数据类型保存在文件里方便读取写入和使用)将信息储存在本地，这样在每次登陆的时候就可以读取用户信息并推荐出产品。使用的url组织有：

图 4: url结构

```
urls = (
    '/', 'start',
    '/re','re',
    '/in', 'index',
    '/login_action', 'login',
    '/registe_action','registe',
    '/voice_action','voice',
    '/s', 'text',
)
```

4 浏览器功能与程序的功能结合

我们想做的功能有登录，图片识别这些功能，就需要能够在网页中运行Python程序。然而，我们不是很会用HTML语言，因此在把我们的网页前端和Python程序后端结合的时候出现了困难。我们不会写一个HTML代码，让我们在网页中点击一个按钮就可以运行一个本地的Python程序。于是我们采用了一个比较好的折中办法，就是用Python写个浏览器，然后把需要与Python程序结合的按钮放到浏览器的功能里面。于是我们使用PyQt4制作了一个搜索引擎。如图

图 5: 浏览器的注册页面

5 图片识别

我们想达到用户上传一张图片，我们就能根据这个图片的一些特征得知这个图片是个什么东西。然后在搜索引擎中搜索这个信息，就可以达到图片识别搜索商品的方法了。经过多方学习，我们选择了Keras机器学习框架。结合ImageNet上的图片识别神经网络权重，我们搭建了一个图片识别神经网络，通过这个神经网络，我们就可以实现上传一张图片我们就能得到这个图片是个什么东西，并且把从图片得到的信息放进搜索引擎中搜索。得到图片搜索的商品结果。

图 6: 图片识别功能

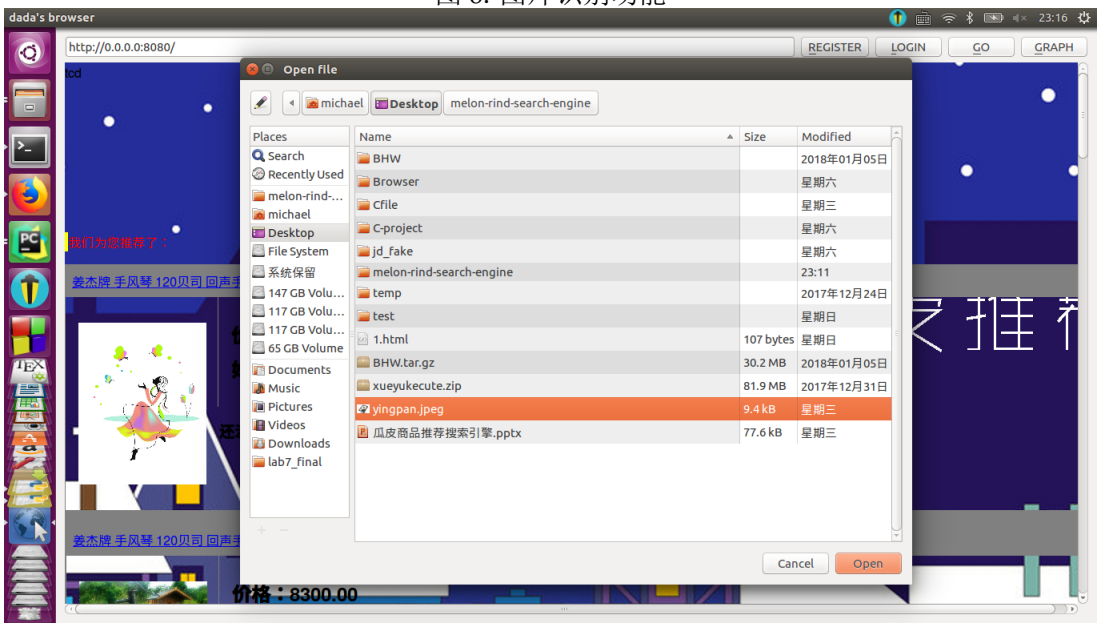
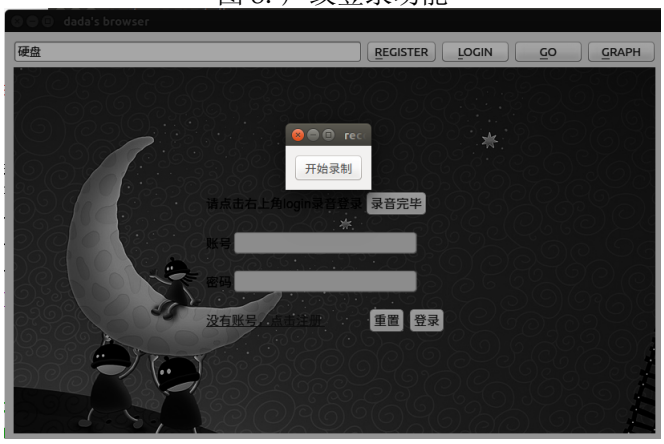


图 7: 图片识别搜索结果

6 声纹识别的登录方式

我们为了方便用户登录，特意制作了声纹识别登录方式。用户注册时，需要录制自己的声纹密码。再登录时，只需要录下自己的声音即可登录。我们的声纹识别使用了mfcc的方法，对每一条录音建立一个vq码本。对登录的声音进行同样的操作，对比得到的vq码本和库中的密码本的差距最小的就是匹配者。效果如图：

图 8: 声纹登录功能



7 总结与展望

这次的小组实验，我们充分利用了本学期学习的知识，并且结合自己的创意制作出了一个比较实用的商品推荐搜索系统，得到了很大的收获。在未来对我们的作品的完善工作中，我们准备再多爬取一些数据，并且尝试使用我们现在因受制于数据量而无法使用的基于关联规则的商品推荐算法。

8 源代码

Input Python source:

```
# -*- coding: utf-8 -*-
import web
from web import form
import urllib2
#import cchardet
import os
import sys
import pickle
from SearchFiles import *
global user_name
from compare import *
from recommend import *
user_name=None
urls = (
    '/', 'start',
    '/re','re',
    '/in', 'index',
    '/login_action', 'login',
    '/registe_action','registe',
    '/voice_action','voice',
    '/s', 'text',
)
```

```
render = web.template.render('templates') # your templates

def sort_by_weight(origion_list):
    #print origion_list
    tfidf_list = []
    tag_list = []
    price_list = []
    wellrate_list = []
    for i in range(len(origion_list)-1,0,-1):
        tfidf_list.append(i)
    for item in origion_list[1:]:
        #print item
        try:
            item[6]
            tag_list.append(1)
        except IndexError:
            tag_list.append(0)
        #print item[2]
        try:
            price_list.append(float(item[2]))
        except Exception:
            price_list.append(float(100.0))
        wellrate_list.append(float(item[4].strip().strip('%')))
    weighted_list = []
    tf_weight = 10
    tag_weight = 6000
    price_weight = -20.0 / max(price_list)
    wellrate_weight = 60
    for i in range(len(origion_list)-1):
        s = 0
        s += (tf_weight*tfidf_list[i])
        s += (tag_weight*tag_list[i])
        s += (price_weight*price_list[i])
        s += (wellrate_weight*(wellrate_list[i]-97.5))
        weighted_list.append(s)
    res_dic = {}
    for i in range(len(origion_list)-1):
        res_dic[weighted_list[i]] = origion_list.index(origion_list[i])
    res = []
    for i in sorted(res_dic.keys(),reverse=True):
        res.append(origion_list[res_dic[i]+1])
    res = [origion_list[0]] + res

    return res



def func_index(command):
```

```
    STORE_DIR = "index"
    vm_env.attachCurrentThread()
    #base_dir = os.path.dirname(os.path.abspath(sys.argv[0]))
    directory = SimpleFSDirectory(File(STORE_DIR))
    searcher = IndexSearcher(DirectoryReader.open(directory))
    analyzer = StandardAnalyzer(Version.LUCENE_CURRENT)
    return sort_by_weight(run(searcher, analyzer,command,'contents'))




def func_tags(command):
    STORE_DIR = "tags"
    vm_env.attachCurrentThread()
    #base_dir = os.path.dirname(os.path.abspath(sys.argv[0]))
    directory = SimpleFSDirectory(File(STORE_DIR))
    searcher = IndexSearcher(DirectoryReader.open(directory))
    analyzer = StandardAnalyzer(Version.LUCENE_CURRENT)
    return sort_by_weight(run(searcher, analyzer,command,'tags'))

class start:
    def GET(self):
        return render.login()
class re:
    def GET(self):
        return render.registe()

class login:
    def GET(self):
        global user_name
        vm_env.attachCurrentThread()
        name=voice_recognize('key')
        judge=True
        user_name=name
        kword_1=recommend1(name)
        kword_2,kword_3=recommend2(name)
        kword_4=recommend3(name)
        ans_1=func_index(kword_1)
        ans_2=func_tags(kword_2)
        ans_3=func_tags(kword_3)
        ans_4=func_index(kword_4)
        ans=[]
        for i in range(1,4):
            ans.append(ans_4[i])
        for i in range(1,4):
            ans.append(ans_1[i])
            ans.append(ans_2[i])
```

```
119             ans.append(ans_3[i])
120         return render.formtest(name,judge,ans)
121     def POST(self):
122         global user_name
123         vm_env.attachCurrentThread()
124         user_data = web.input()
125         name= user_data.username
126         password=user_data.userpassword
127         read_file=open("customer_message.pkl",'rb')
128         usermessage=pickle.load(read_file)
129         read_file.close()
130         judge=False
131         if usermessage[name][0]==password:
132             judge=True
133             user_name=name
134             kword_1=recommend1(name)
135             kword_2,kword_3=recommend2(name)
136             kword_4=recommend3(name)
137             ans_1=func_index(kword_1)
138             ans_2=func_tags(kword_2)
139             ans_3=func_tags(kword_3)
140             ans_4=func_index(kword_4)
141             ans=[]
142             for i in range(1,4):
143                 ans.append(ans_4[i])
144 
145             for i in range(1,4):
146                 ans.append(ans_3[i])
147                 ans.append(ans_1[i])
148                 ans.append(ans_2[i])
149 
150             return render.formtest(name,judge,ans)
151         else:
152             return render.login()
153 
154 class registe:
155     def GET(self):
156         global user_name
157         vm_env.attachCurrentThread()
158         user_data = web.input()
159         name= user_data.username
160         user_name=name
161         password=user_data.userpassword
162         gender=user_data.gender
163         age=user_data.age
164         hobby=user_data.hobby
165         job=user_data.hobby
166         infile = open('/home/michael/Desktop/Browser/usrname.txt', 'w')
```

```
167         infile.write(user_name)
168         infile.close()
169         read_file=open("customer_message.pkl",'rb')
170         usermessage=pickle.load(read_file)
171         read_file.close()
172         usermessage[name]=[password,gender,age,hobby,job]
173         write_file=open("customer_message.pkl",'wb')
174         pickle.dump(usermessage,write_file)
175         write_file.close()
176         return render.voice(name)
177 class voice:
178     def  GET(self):
179         judge=True
180         infile = open('/home/michael/Desktop/Browser/usrname.txt', 'r')
181         name=infile.readline()
182         kword_1=recommend1(name)
183         kword_2,kword_3=recommend2(name)
184         ans_1=func_index(kword_1)
185         ans_2=func_tags(kword_2)
186         ans_3=func_tags(kword_3)
187         ans=[]
188         for i in range(1,5):
189             ans.append(ans_3[i])
190         for i in range(1,5):
191             ans.append(ans_1[i])
192             ans.append(ans_2[i])
193         return render.formtest(name,judge,ans)
194 class index:
195     def GET(self):
196         return render.formtest()
197 class text:
198     def GET(self):
199         name=user_name
200         kword_1=recommend1(name)
201         kword_2,kword_3=recommend2(name)
202         ans_1=func_index(kword_1)
203         ans_2=func_tags(kword_2)
204         ans_3=func_tags(kword_3)
205         ans=[]
206         for i in range(1,5):
207             ans.append(ans_1[i])
208             ans.append(ans_2[i])
209             ans.append(ans_3[i])
210         user_data = web.input()
211         if user_data.keyword:
212             a = func_index(user_data.keyword)
213             ans_4=func_tags(user_data.keyword)
214             ans_4=ans_4[1:5]
```

```
215             read_file=open("good_message.pkl",'rb')
216             goodmessage=pickle.load(read_file)
217             read_file.close()
218             if user_name in goodmessage:
219                 goodmessage[user_name].append(user_data.keyword)
220             else:
221                 goodmessage[user_name]=[user_data.keyword]
222             write_file=open("good_message.pkl",'wb')
223             pickle.dump(goodmessage,write_file)
224             write_file.close()
225         else:
226             a=['']
227             ans_4=['']
228         return render.result(a,ans,ans_4)
229     def POST(self):
230         name=user_name
231         kword_1=recommend1(name)
232         kword_2,kword_3=recommend2(name)
233         ans_1=func_index(kword_1)
234         ans_2=func_tags(kword_2)
235         ans_3=func_tags(kword_3)
236         ans=[]
237         for i in range(1,5):
238             ans.append(ans_1[i])
239             ans.append(ans_2[i])
240             ans.append(ans_3[i])
241         infile=open("graph_result.txt",'r')
242         word=infile.readline().decode('utf-8')
243         infile.close()
244         read_file=open("good_message.pkl",'rb')
245         goodmessage=pickle.load(read_file)
246         read_file.close()
247         if user_name in goodmessage:
248                 goodmessage[user_name].append(word)
249         else:
250                 goodmessage[user_name]=[word]
251         write_file=open("good_message.pkl",'wb')
252         pickle.dump(goodmessage,write_file)
253         write_file.close()
254         a = func_index(word)
255         ans_4=func_tags(word)
256         ans_4=ans_4[1:5]
257         return render.result(a,ans,ans_4)
258
259
260
261
262 if __name__ == "__main__":
```

```
    vm_env=lucene.initVM(vmargs=['-Djava.awt.headless=true'])
    app = web.application(urls, globals())
    app.run()
    print user_name
```

```
#!/usr/bin/env python

INDEX_DIR = "IndexFiles.index"

import sys, os, lucene

from java.io import File
from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.index import DirectoryReader
from org.apache.lucene.queryparser.classic import QueryParser
from org.apache.lucene.store import SimpleFSDirectory
from org.apache.lucene.search import IndexSearcher
from org.apache.lucene.util import Version

"""
This script is loosely based on the Lucene (java implementation) demo class
org.apache.lucene.demo.SearchFiles.  It will prompt for a search query, then it
will search the Lucene index in the current directory called 'index' for the
search query entered against the 'contents' field.  It will then display the
'path' and 'name' fields for each of the hits it finds in the index.  Note that
search.close() is currently commented out because it causes a stack overflow in
some cases.
"""


def run(searcher, analyzer,keyword,way):
    while True:
        try:
            command = keyword.encode('utf8')
        except UnicodeDecodeError:
            command = keyword
        if command == '':
            return []
        if way=='contents':
            query = QueryParser(Version.LUCENE_CURRENT, "contents",analyzer).parse(
                command)
        elif way=='tags':
            query = QueryParser(Version.LUCENE_CURRENT, "tag",analyzer).parse(command)
        scoreDocs = searcher.search(query, 50).scoreDocs
        result=[]
        result.append(command)
        for scoreDoc in scoreDocs:
            doc = searcher.doc(scoreDoc.doc)
```

```
            item=[]
            item.append(doc.get('title'))
            item.append(doc.get('url'))
            item.append(doc.get('price'))
            item.append(doc.get('imgurl'))
            item.append(doc.get('wellrate'))
            item.append(doc.get('comment'))
            item.append(doc.get('tag'))
            #print doc.get('comment').encode('utf8')
            result.append(item)
    #print result

        return result
```

```
from keras.applications.mobilenet import MobileNet
from keras.preprocessing import image
from keras.applications.mobilenet import preprocess_input, decode_predictions
import numpy as np
import translate as tr

def graph_reco(path):
    model = MobileNet(weights='imagenet')

    img_path = path
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    preds = model.predict(x)
    predict_list = decode_predictions(preds, top=3)[0]
    to_trans = []
    for i in predict_list:
        to_trans.append(' '.join(str(i[1]).split('_')))
    translator = tr.Youdao_translate()
    res = []
    for i in to_trans:
        res.append(translator.get_translation(i))

    return res
```

```
#coding=utf8
import pickle
import math

def get_vector_book(usr):
    infile = open('customer_message.pkl', 'rb')
    message = pickle.load(infile)
```

```
    infile.close()
    v = []
    for i in message:
        if i != usr:
            temp = []
            for k,j in enumerate(message[i]):
                if k == 1:
                    if j.encode('utf8') == '鑾:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣temp.append(0)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣else:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣temp.append(1)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣k␣==␣2:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣h,l␣=␣j.split('-')
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣s␣=␣(int(h)+int(l))␣*␣1.0␣/␣2
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣temp.append(s)
␣␣␣␣␣␣␣␣␣␣␣␣v.append(temp)
␣␣␣␣return␣v


def␣get_dis(v1,␣v2):
␣␣␣␣dis␣=␣0
␣␣␣␣for␣i␣in␣range(len(v1)):
␣␣␣␣␣␣␣␣dis␣+=␣(v1[i]␣-␣v2[i])␣**␣2
␣␣␣␣return␣math.sqrt(dis)


def␣find_a_nearest(usr):
␣␣␣␣v_book␣=␣get_vector_book(usr)
␣␣␣␣infile␣=␣open('customer_message.pkl',␣'rb')
␣␣␣␣message␣=␣pickle.load(infile)
␣␣␣␣infile.close()
␣␣␣␣v␣=␣[]
␣␣␣␣print␣message[usr][1]
␣␣␣␣v.append(0␣if␣message[usr][1].encode('utf8')␣==␣'鑾 else 1)
    h,l = message[usr][2].split('-')
    s = (int(h)+int(l))*1.0/2
    v.append(s)

    l = []
    for i in range(len(v_book)):
        l.append(get_dis(v,v_book[i]))
    print l

find_a_nearest('sry')
```