EXERCISE1

A:练习1的a主要在于实现函数simple\_tokenize(string)这个函数主要功能是将输入的字符串分割成一系列子串。自然想到split()方法，题目要求不区分大小写，并且子串为小写，因此先将字符串转换为小写再调用split()。仔细观察题目下的代码，发现要求该函数能忽视“！”，并且能以“/”和“.”为分隔符。这是就需要正则表达式，如果匹配到字符串中有”!”,就调用strip(‘!’)删除。如果字符串中没有空格，就以“/”和“.”为分隔符。

def simple\_tokenize(string):

if " " in string:

return string.lower().split()

elif re.match('!+',string)!=None:

string1= string.strip('!')

string2=string1.replace("/",".")

return string2.lower().split(".")

B：b中引入了停用词的概念，停用词是对理解文件的内容或意义没有帮助的公用词。实现函数tokenize(string)，并在所生成的子串中排除stopwords，题目所需的stopwords由stopwords.txt给出。首先要将文件读入，并将读入的字符串转换成可用的一个个子串。即所给的停用词。

sto=open('stopwords.txt','r').read()

sto1=sto.split()

对于输入函数tokenize(string)的字符串，除了a中的工作，还要比较每一个子串是否是停用词，如果不是才可以加入结果列表中。部分代码为：

for c in string2:

if c not in sto1:

result.append(c)

C：c开始使用两个小数据集，每个数据集建立一个关于tokens的字典，ID作为字典的键，函数tokenize的输出作为值，需要tokenize的字符串包括name, description, manufacturer 但是不包括price ，还需要计算这两个数据集中总共有多少tokens,那个Amazon record拥有最多的tokens。

首先写了两个函数getdic(name)和getdtoken(dic)来帮助建立字典，函数getdic (name)将文件名字输入，读取文件，将id作为键，name,description，manufacturer 作为值。

def getdic(name):

csvfile = file(name, 'rb')

i=-1

dic={}

line=csv.reader(csvfile)

for x in line:

i=i+1

if i>0:

dic[x[0]]=x[1]+" "+x[2]+" "+x[3]

csvfile.close()

return dic

函数getdtoken(dic)用来将dic中值所对应的字符串用tokenize变为一系列子串

def getdtoken(dic):

dtoken={}

for x in dic:

token=tokenize(dic[x])

dtoken.setdefault(x,token)

return dtoken

利用上述两个函数分别求出am\_d，am\_dtoken，go\_d,go\_dtoken，接着使用了字典dtime来记录ID值及所对应的tokens数。以am\_dtime为例：

am\_dtime={}

for x in am\_dtoken:

com1=am\_dtoken[x]

time=len(com1)

am\_dtime.setdefault(x,time)

同理求出go\_dtime

total\_tokens=str(sum(go\_dtime[k] for k in go\_dtime)+sum(am\_dtime[t] for t in am\_dtime))

这样就得到了两个数据集总共拥有的tokens.

对于求Amazon record拥有最多的tokens，首先想到了sort方法。由于sort是list的方法，先将dictionary转换成list再进行排序，这样就可以得到biggest\_record。

EXERCISE2

首先引入概念TF-IDF

TF：与出现的频率有关，相同的token在不同的文档中有不同的权重。local weights

IDF：逆频率，总是相同的频率，global weights

A: 实现函数tf(tokens)，输入一个token list 练习一中b输出的那种，输出一个字典，键是token，值是TF值。求TF值需要总token数及每个token出现的次数。

def tf(tokens):

tfd={}

i=0

tfd=tfd.fromkeys(tokens)

#print tfd

n=len(tokens)

for x in tfd:

for y in tokens:

if x==y:

i=i+1

tfd[x]=round(float(i)/float(n),4)

i=0

return tfd

B: 实现函数idfs(corpus)，输入是一个数据集合，是一个键为ID，值为token list的字典，输出也是一个字典，键是token，值是idf值。求在两个小数据集中共有多少个不同的token。对于每一个token的idf值，需要N——文档总数，和n——包含这个token的文档数。

def idfs(corpus):

lis=[]

idfsd={}

N=len(corpus)

n=0

for x in corpus:

lis=lis+(corpus[x])

s= list((set(lis)))

idfsd=idfsd.fromkeys(s)

for x in idfsd:

for y in corpus:

if x in corpus[y]:

n=n+1

idfsd[x]=round(float(N)/float(n),4)

n=0

return idfsd

至于求在两个小数据集中共有多少个不同的token，就是该函数返回的这个字典有多少项。

idfsd= idfs(combined\_d)

unique\_tokens= len(idfsd)

C：在b中求出的字典中找到10个最小的IDF值的token，看他们在实体解析中是否真的有用。下面是百度到的一种将字典值对转换成元组集合并排序的方法，

从中找出10个最小的。

idf\_tokens=sorted(idfsd.iteritems(), key=lambda d:d[1], reverse = False )

for x in range (10):

small\_idf\_tokens.append(idf\_tokens[x])

我得到10个token为：software，use，new，complete，features，easy，create，system，windows，1. 这十个词对实体解析有没有用呢？有的看起来有用，有的貌似没有。

D：

E：实现函数tfidf(tokens, idfs)，函数的输入是token list 和idfs输出的是字典（键是token，值是token的idf值），函数的输出是一个字典，键是token，值是token的tfidf值。计算ID为'b000hkgj8k'的token和权重。TFIDF实际就是TF\*IDF,将对应的TF和IDF值相乘就好。

def tfidf(tokens,idfs):

tf\_tokens=tf(tokens)

dic={}

dic=dic.fromkeys(tokens)

for i in tf\_tokens:

dic[i]=tf\_tokens[i]\*idfs[i]

return dic

EXERCISE3

首先引入余弦相似度的概念。大致来说就是计算两个文件向量之间的角度，权重作为tokens的量纲，文件向量角度小，就是相似，反之，文件向量角度大，就是不相似。

A：实现函数cosine\_similarity(string1, string2, idfs)输入是两个字符串和idfs输出的字典（键是token，值是token的idf值），输出是两个字符串的余弦相似度。

函数dotprod(a,b)是计算a,b两个向量相乘的值，输入的a,b是两个字典

def dotprod(a,b):

dot=0

for x in a:

if x not in b:

b.setdefault(x,0)

for x in b:

if x not in a:

a.setdefault(x,0)

for x in a:

dot=dot+a[x]\*b[x]

return dot

函数norm(a)是计算|a|的值，输入的a是个字典。

def norm(a):

norma=0

for x in a:

norma=norma+a[x]\*a[x]

return math.sqrt(norma)

函数cossim(a,b)是计算a,b的余弦值，输入的a,b是两个字典。

def cossim(a,b):

if (norm(a)\*norm(b))==0:

return 0

else :

return dotprod(a,b)/(norm(a)\*norm(b))

函数cosine\_similarity(string1, string2, idfs)是将字符串转换成token子串，然后计算这两个子串的tfidf权重，得到两个字典，接着计算它们的余弦。

def cosine\_similarity(string1,string2,idfs):

s1=exercise1.tokenize(string1)

s2=exercise1.tokenize(string2)

dic1=exercise2.tfidf(s1,idfs)

dic2=exercise2.tfidf(s2,idfs)

return cossim(dic1,dic2)

B：现在我们终于可以尝试实体分析了，对谷歌和亚马逊的小数据集中的每一项记录计算它们的余弦相似度。建立一个字典，键是谷歌和亚马逊ID的元组对，值是它们的余弦相似度。这里我使用了函数cos\_sim(dic1,dic2,idfs)，输入是两个字典，字典的键是ID，值是字符串，（注意这里不是token串）和idfs输出的字典（键是token，值是token的idf值）。函数的输出是题目所需要的字典。

def cos\_sim(dic1,dic2,idfs):

similarities={}

for x in dic1:

for y in dic2:

string1=str(dic1[x])

string2=str(dic2[y])

sim=cosine\_similarity(string1,string2,idfs)

string=(x,y)

similarities.setdefault(string,sim)

return similarities

C：使用黄金标准数据来回答一些问题，小数据集中有多少重复对？重复对的平均余弦相似度，不重复对的呢！从识别重复对的质量上看，余弦相似度做到好不好。读取黄金标注数据文件的方法同之前，读取之后，之前B中的字典键与黄金标准数据一一比较即可得到重复对数。total是重复对的总余弦相似度，total2是非重复对的总余弦相似度。

avg\_sim\_dups=total/true\_drups

avg\_sim\_non=total2/(len(similarities)-true\_drups)

EXERCISE4

能感觉处理Python在上面处理时速度较慢，为了减少开销！使用倒排索引。倒排索引就是根据属性的值来查找记录。

A：工作比较简单，开始使用完整的数据，计算完整数据的token串，计算完整数据的IDFS值。预计算TF-IDF weights,表现为一个字典，键是ID，值也是一个字典（键是token，值是权重）。预计算norm，表现也为一个字典，键是ID，值是其对应的norm值。

B：给完整的数据建立倒排索引。键是token，值是有这个token的ID。先找出所有的token，建立一个空字典，再遍历一遍forward\_index，对于每一个token在新字典对应的值中加上该ID

def invert\_index(forward\_index):

inlist=[]

for x in forward\_index:

for y in forward\_index[x]:

inlist.append(y)

s= list((set(inlist)))

ind={}

ind=ind.fromkeys(s,[])

for x in forward\_index:

for y in forward\_index[x]:

lis=ind[y]

lis.append(x)

return ind

C：用到了一个新东西，迭代器。需要得到一个字典，键是两组数据的ID元组对，值是具有的相同的token。用到了Ifilter，product两种创建迭代器的方法，代码如下：

common\_tokens={}

from itertools import \*

for i in ifilter(lambda x:x in gogle\_inv,amazon\_inv):

for l in product(amazon\_inv[i],gogle\_inv[i]):

if l in common\_tokens:

lis=[]

lis=common\_tokens[l]

lis.append(i)

else :

alis=[]

alis.append(i)

common\_tokens.setdefault(l,alis)

print (len(common\_tokens))

D：完成函数fast\_cosine\_similarity(a\_rec,g\_rec,tokens)，得到一个字典，键是两个数据的元组对，值是余弦相似度。不过这里用到了上面的倒排索引。

def fast\_cosine\_similarity(a\_rec,g\_rec,tokens):

sims={}

for x in tokens:

s=0

for y in tokens[x]:

s=s+(a\_rec[x[0]][y]\*g\_rec[x[1]][y])

norm=0

g\_norms=norms(g\_rec)

a\_norms=norms(a\_rec)

norm=a\_norms[x[0]]\*g\_norms[x[1]]

sims.setdefault(x,s/norm)

return sims

EXERCISE 5

Fmeasure=2(precision\*recall)/(precision+recall)

Precision(查准率)

Recall(查全率)

TP 查询到的相关数据

TN 未查询的不相关数据

FP 查询到的不相关数据

FN 未查询到的相关数据

Precision=TP/(TP+FP)

Recall=TP/(TP+FN)

有个阈值，余弦相似度大于这个值就可以算相关，小于就算不相关，搞清楚这些概念代码就不难了，图形部分未完全弄好。

注意：代码应该不是完全正确的，仅供参考，不对的地方多多指正.