تمرین شماره 2 کاوش دادگان انبوه | دکتر هراتی‌زاده

پیدا کردن اسناد مشابه

۱. تبدیل هر سند به مجموعه‌های شینگلی

اولین مرحله در شناسایی اسناد تبیدل رشته‌های حرفی به شینگل‌هایی با طول k است برای این کار تابع extract\_kshingles پیاده‌سازی شد. ما در این‌جا شینگل‌های تکراری را درنظر نگرفته‌ایم. با توجه به توصیه کتاب مقدار k برابر ۹ در نظر گرفته شد.

def extract\_kshingles(doc, k):

* + """
  + doc - string that should be divided to shingles.
  + k - length of shingles.
  + Returns a set of k-shingles
  + """
  + shingles = set()
  + if (k<=len(doc)):
  + for i in range(0, (len(doc)-k+1)):
  + shingles.add(doc[i:i+k])
  + return shingles

با توجه به اینکه بیشتر شینگل‌ها در سند وجود ندارند جهت استفاده بهینه‌تر از حافظه و راحتی مقایسه‌ی شینگل‌ها آن‌ها توسط تابع CityHash32 به اعداد ۳۲بیتی(۴ بایتی) هش گردیدند. این عمل توسط تابع hash\_kshingles انجام میگیرد.

def hash\_kshingles(raw\_shingles):

* + """
  + raw\_shingles - set of raw string k-shingles
  + Returns a set of hashed k-shingles
  + """
  + hashed\_shingles = set()
  + for rs in raw\_shingles:
  + # Hash the shingle to a 32-bit integer.
  + hs = CityHash32(rs)
  + hashed\_shingles.add(hs)
  + return hashed\_shingles
* ادامه‌ی قسمت‌ها برخلاف این قسمت که به صورت توابع جداگانه پیاده سازی شده‌اند. در داخل کلاس CharacteristicMatrix اجرا شده‌اند.

۲. ساختن ماتریسی که مشخص کند هر سند دارای چه شینگل‌هاییست. (مقادیر hash شینگل‌ها را باید استفاده کنید.)

هر سند توسط متد add\_doc به کلاس اضافه می‌شود این متد اسناد را با کمک توابعی که قبلا تعریف شد به لیستی از هش سینگل‌ها تبدیل کرده و این مقدار را در دیکشنری doc\_shingles در کلیدی برابر با نام سند ذخیره می‌کند. همچنین این متد تمامی شینگل‌های اسناد را به مجموعه‌ی total\_shingles\_set اضافه می‌کند.

def add\_doc(self, doc\_name, doc\_content):

"""

doc\_name - name of the file.

doc\_content - content string.

Adds documents to doc\_shingles and updates total\_shingles\_set

"""

raw\_shingles = extract\_kshingles(doc\_content, self.k)

hashed\_shingles = hash\_kshingles(raw\_shingles)

# Uniting hashed shingles with current total\_shingles\_set

self.total\_shingles\_set |= hashed\_shingles

self.doc\_shingles[doc\_name] = list(hashed\_shingles)

برای ساخت characteristic matrix باید متد create\_charactersitic\_matrix از کلاس فراخوانی شود.این متد ابتدا مجموعه‌ی همه‌ی شینگل‌ها را که قبلا ساخته شده به لیستی مرتب از آن‌ها تبدیل می‌کند. سپس به ازای هر سند شروع به بررسی عضویت آن شینگل کرده و ماتریس characteristic\_matrix را میسازد.

def create\_charactersitic\_matrix(self):

"""

Creates characteristic matrix for added documnets

"""

self.total\_shingles\_list = list(self.total\_shingles\_set)

for doc in self.doc\_shingles:

#making vector of zeros

self.characteristic\_matrix[doc] = [0] \* len(self.total\_shingles\_list)

for shingle in self.doc\_shingles[doc]:

i = self.total\_shingles\_list.index(shingle)

self.characteristic\_matrix[doc][i] = 1

return self.characteristic\_matrix

۳. ساختن ماتریس minhash signature با استفاده از ماتریس قبلی.

برای پیاده‌سازی این متد باید جایگشت‌ها به صورت توابع هش پیاده شوند به همین خاطر متد hash\_row\_number ایجاد شده است. آرگومان hash\_id در این متد به صورت یک seed عمل میکند و با اختصاص هر عدد تابع هش دیگری ایجاد می‌شود.

def hash\_row\_number(self, hash\_id, x):

"""

returns permutation's hash

"""

hash\_range = len(self.total\_shingles\_list)

if (hash\_id ==0):

return x

else:

return (x\*hash\_id + hash\_id - 1) % hash\_range

return x

متد create\_minhash\_signature با استفاده از الگوریتم شرح داده شده در کتاب اقدام به ساخت minhash\_signature\_matrix میکند. جهت این کار ابتدا با توجه به آرگومان r که نشانگر توابع هش یا سطرهاست و مقدار اولیه که برابر شماره آخرین سطر characteristic matrix است(این مقدار اولیه مشابه مقدار ∞

در کتاب است) ماتریس minhash\_signature ساخته می‌شود.سپس به ازای هر سطر و به ازای هر تابع هش در هر سند مقدار هش ردیف سطر characteristic matrix محاسبه می‌شود اگر مقدار ۰ در آن مکان بود هیچ کاری انجام نمی‌شود در غیر این صورت مقدار آن‌خانه در minhash\_signature\_matrix با مینیمم مقدار فعلی آن خانه در minhash\_signature\_matrix و هش کنونی محاسبه شده برای سطر جابه‌جا می‌شود.

def create\_minhash\_signature(self, r):

"""

r - number of hash functions h1, h2, h3, h4,...,hr.

Creates minhash signature matrix

"""

#Initializing minhash signature matrix by inf

for doc in self.characteristic\_matrix: #for each column

self.minhash\_signature\_matrix[doc] = [len(self.total\_shingles\_list)] \* r

for i, shingle in enumerate(self.total\_shingles\_list):

print("row = {}".format(i))

for hash\_id in range(r):

h\_i = self.hash\_row\_number(hash\_id, i)

for doc in self.characteristic\_matrix:

if(self.characteristic\_matrix[doc][i] != 0):

self.minhash\_signature\_matrix[doc][hash\_id] = min(self.minhash\_signature\_matrix[doc][hash\_id],

h\_i)

یافتن موارد کاندید شباهت با استفاده از توابع LSH

ماتریکس minhash\_signature\_matrix به b بند که هر کدام band\_size سطر دارند تقسیم میشوند. هر بردار اعداد یک سند در این بند هش می‌گردد و نام آن سند در باکت مربوطه قرار می‌گیرد.

def get\_similar\_docs\_by\_lsh(self, band\_size):

"""

band\_size - number of rows in each band.

Returns dictionary of buckets that docs mapped into.

"""

for doc in self.minhash\_signature\_matrix:

i=1

for chunk in chunks(self.minhash\_signature\_matrix[doc], band\_size):

bucket = CityHash32("".join(str(x) for x in chunk)+str(i))

i = i+1

self.lsh\_buckets[bucket].append(doc)

return self.lsh\_buckets

در پیاده سازی این سند از تابع chunks استفاده گردیده است، این تابع یک generator برمیگرداند که تکه‌های لیست دریافتی(آرگومان اول) را با طول آرگومان دوم برمیگرداند. پیاده سازی این تابع به صورت زیر است:

def chunks(l, n):

"""

l - list to be chunked.

n - size of chunks.

Generates n-sized chunks from a list

"""

for i in range(0, len(l), n):

yield l[i:i+n]

۵. چک کردن کاندید‌ها و بررسی مقدار شباهتشان.

متن برنامه اصلی

def main():

cm = CharacteristicMatrix()

file\_names = os.listdir(PATH\_TO\_DOCS)

for file\_name in file\_names:

with open(os.path.join(PATH\_TO\_DOCS, file\_name), 'r') as doc:

content = doc.read()

cm.add\_doc(str(file\_name), content)

cm.create\_charactersitic\_matrix()

cm.create\_minhash\_signature(200)

buckets = cm.get\_similar\_docs\_by\_lsh(100)

similars = {}

for k in buckets:

if(len(buckets[k])>1):

similars[k] = buckets[k]

print("buckets = {}".format(similars))

if (\_\_name\_\_ == "\_\_main\_\_"):

main()

توابع زیر جهت بررسی مقدار شباهت :

def sim(doc\_x, doc\_y, k):

"""

x, y - documnets.

k - length of shingles.

Returns jaccard similarity of two strings (0:0.5)

"""

kshingles\_x = extract\_kshingles(doc\_x, k)

kshingles\_y = extract\_kshingles(doc\_y, k)

hashed\_shingles\_x = list(hash\_kshingles(kshingles\_x))

hashed\_shingles\_y = list(hash\_kshingles(kshingles\_y))

union = hashed\_shingles\_x + hashed\_shingles\_y

intersection = []

for x in hashed\_shingles\_x:

if x in hashed\_shingles\_y:

intersection.append(x)

hashed\_shingles\_y.remove(x)

jaccard\_similarity = len(intersection) / len(union)

return jaccard\_similarity

def sim\_files(file\_x, file\_y, k):

"""

file\_x, file\_y - documnets files.

k - length of shingles.

Returns jaccard similarity of two fils (0:0.5)

"""

doc\_x, doc\_y = "", ""

with open(os.path.join(PATH\_TO\_DOCS, file\_x), 'r') as doc:

doc\_x = doc.read()

with open(os.path.join(PATH\_TO\_DOCS, file\_y), 'r') as doc:

doc\_y = doc.read()

return sim(doc\_x, doc\_y, k)

خروجی برنامه

buckets = {2191515575: ['17', '19'], 3414974309: ['17', '19'], 2380508325: ['1', '16', '15'], 3503431405: ['1', '16', '15']}