Data Mining for Business Analytics

Exercise 2: Locality-sensitive Hashing

Task 2.1 Computational exercise (4 p)

Consider the following matrix:

Element	S1	S2	S3	S4
0	1	1	0	1
1	0	1	0	0
2	0	0	0	1
3	1	0	1	0
4	0	1	1	1
5	0	0	1	0

Compute the minhash signature for each column using the following hash functions: $h_1(x) = 2x + 1 \; mod \; 6$; $h_2(x) = 3x + 2 \; mod \; 6$; $h_3(x) = 5x + 2 \; mod \; 6$.

Which of these hash functions are true permutations?

How close are the estimated Jaccard similarities for the six pairs of columns to the true Jaccard similarities?

Task 2.2 Computational exercise (4 p)

Consider the S-curve of equation: $1 - (1 - s^r)^b$

Evaluate the curve for the following values of r and b:

- $r = 3$ and $b = 10$
- $r = 7$ and $b = 20$
- $r = 5$ and $b = 40$

Plot the results to your report.
For each of the above pairs find the value of Jaccard similarity for which the probability of becoming a candidate is 0,5. How close is this to the estimate of $\left(1/_b\right)^{1/_r}$?

Task 2.3 (6 p)

Upload text files from the zip file in Moodle.

2.3.1 Shingling

The code below generates ngrams of input text files. It removes spaces and all non-letter characters (preserving only letters a-z) and then generates ngrams by shingling.

Test the code with different *shingle length* values. Include first rows of the test results in your report (do not copy the whole result!). Based on the course material, how does shingle length affect similarity analysis?

Sometimes it is useful to preserve white spaces in shingles. You can preserve white spaces (the \s character) by using the condition `[^a-z]\s+` in the argument of the *re.sub* function. Test this and include first rows of the result in your report.

```
import pyspark
import re

sc = pyspark.SparkContext("local", "Similarity")

def make_shingles(text, shingle_len):
    text = text.lower()
    text = re.sub(r'[^a-z]+', '',text )
    for i in range( 0, len(text)-shingle_len-1 ):
        yield text[i:i+shingle_len]

shingle_length = 6

files = sc.wholeTextFiles('*.txt')
files = files.map(lambda p: (p[0].split("/")[-1], list(make_shingles( p[1], shingle_length) ) ) )
res = files.collect()
print(files.take(1))
```

2.3.2 Calculating hashes

Modify the *make_shingles* function to produce hashes by replacing the last line of the function by:

```
yield string_hash(text[i:i+shingle_len],shingle_len, Ashingle, Pshingle)
```

Add then the following code before the *make_shingles* function definition:

```
from numpy import uint32
from numpy import uint64
import random
import numpy

def generate_random_hash_params_A_P():
    A = random.getrandbits(64)
    P = random.getrandbits(64)
    while A>=P:
        A = random.getrandbits(64)
        P = random.getrandbits(64)
    return A , P

Ashingle, Pshingle = generate_random_hash_params_A_P()

def string_hash(shingle,slen,A,P):
    tmp = uint64(ord(shingle[0]))
    for i in range(1, slen):
        tmp = (tmp*uint64(A) + uint64(ord(shingle[i]))) % uint64(P)
    return uint32(tmp&uint64(0xFFFFFFFF))
```

Run the code. What do you get as an output? In what case does hashing compress the data?

2.3.3 MinHashing

Minhashing captures the whole documents into *N* (the number of hash functions used) minhash codes. Include the following function definitions and code to your script:

```
def uint32_hash(stringhashes,A):
    for i in range(0, len(stringhashes)):
        stringhashes[i] = uint32((uint64(A)*uint64(stringhashes[i])) >> uint64(32))
    return stringhashes

def min_hash(hashes,minhash_A):
    minhashes = []
    for i in range(0,len(minhash_A)):
        minhashes.append(min(uint32_hash(hashes,minhash_A[i])))
    return minhashes


num_minhashes = 15
minhash_A = []
for i in range(0,num_minhashes):
    minhash_A.append( random.getrandbits(64) )
```

Replace the print function by the following code to print out the minhashes:

```
minhashes = files.map(lambda p: (p[0],min_hash(p[1],minhash_A)))
minhashes.cache() #cache result for performance

res = minhashes.collect()
for x in res:
    print(x)
```

Run the code. What do you get as the output?

2.3.4 Jaccard similarity

The following code estimates the Jaccard similarity based on the minhashes:

```
for x in res:
    jaccrow = minhashes.map(lambda p: (p[0],sum(numpy.array(p[1])-
numpy.array(x[1])==0)/num_minhashes)).collect()
    for y in jaccrow:
        print(x[0].ljust(20) + " vs. " + y[0].ljust(20) + " similarity: " + str(y[1]*100) + " %")
```

Test the code.
The more minhash functions we have, the better is the Jaccard similarity estimate. You can calculate the number of minhashes required to obtain certain error rate of the Jaccard similarity estimate by:

$$NrOfMinhashes = \left({}^{1}\!/_{ErrorRate}\right)^2$$

Run the code for the number of minhashes that correspond to error rates of 0.3, 0.2, 0.1, 0.05 (running the code for error rate 0.05 takes several minutes). What is the number of minhashes requires for these error rates? How do the similarities change? Select 5 pairs of documents and report their respective similarities for these error rates.

Select the number of minhashes as 50. Run the code with two different shingle lengths and either containing white spaces or not (i.e., 4 cases in total). Report the similarities for the selected 5 pairs of documents.

Analyze the results for the two previous steps.

Please return your exercise in pdf format to Moodle with your name, student number and the title of the exercise clearly visible on the first page.