Data Mining

Homework 2

Due: 12/11/2017, 23:59.

Instructions—Read carefully!

You must hand in the homeworks electronically and before the due date and time.
Handing in: You must hand in the homeworks by the due date and time by an email to Mara (`sorella@dis.uniroma1.it`) that will contain as attachment (not links!) a .zip or .tar.gz file with all your answers and subject
`[Data Mining class] Homework #`
where `#` is the homework number. In the text you must also mention the your name and student id (matricola). After you submit, you will receive an acknowledgement email that your homework has been received and at what date and time. If you have not received an acknowledgement email within 2 days after you submit then contact Mara.

The solutions for the theoretical exercises must contain your answers either typed up or hand written clearly and scanned.

The solutions for the programming assignments must contain the source code, instructions to run it, and the output generated (to the screen or to files).

For information about collaboration, and about being late check the web page.

Most of the questions are not very hard but require time and thought. **You are advised to start as early as possible, to work in groups (but the writeup should be yours, as per the collaboration policy), and to ask Mara (first) or Aris in case of questions.**

Problem 1. We will run locality-sensitive hashing (LSH) on a set of announcements that we will obtain from the kijiji web site. In this exercise we do the first step, which is downloading and parsing the web pages with the announcements.

For downloading the web pages you may use the package `Requests` or the package `urllib2`. To parse the page you can either use regular expressions through the package `re` (it is anyway a good idea become familiar with regular expressions), or, probably better, use an HTML/XML parser. The `Beautiful Soup` package is a good one but it loads the whole file in memory. This is fine for this problem, since the pages to parse are small, but be careful if you want to use it on large XML files; for those ones check the `lxml` library and the tutorial at

`http://www.ibm.com/developerworks/xml/library/x-hiperfparse/`

Write a program that will download from `http://www.kijiji.it` and parse all the job positions in Italy about *Informatica/Grafica/Web*. Download regular and top announcements, but not sponsored ads. Save in a tab-separated value (TSV) file, for every job (one line per job), the *title*, *short description* (from the job summary page), *the location*, the *publication date* of the job announcement, the *URL link* to its web page, and the *full description* from the ad page. **Because you will make a lot of calls to the kijiji site, make sure that you have a delay (use: `sys.sleep()`) between different downloads of kijiji pages, with a random waiting time between different calls, to avoid being blocked.**

Note: For everything but the *full description* it is enough to parse the page of results. For the *full description* you need to visit each ad page individually. This requires time (because of the waiting

between calls). If you see you cannot do it, use the *short description* in the Problem 2.

Problem 2. Here we are asking to implement nearest-neighbor search for text documents. You have to implement shingling, minwise hashing, and locality-sensitive hashing. We split it into several parts:

1. Implement a class that, given a document, creates its set of character shingles of some length k. Then represent the document as the set of the hashes of the shingles, for some hash function.
2. Implement a class, that given a collection of sets of objects (e.g., strings, or numbers), creates a minwise hashing based signature for each set.
3. Implement a class that implements the locally sensitive hashing (LSH) technique, so that, given a collection of minwise hash signatures of a set of documents, it finds the all the documents pairs that are near each other.

To test the LSH algorithm, also implement a class that given the shingles of each of the documents, finds the nearest neighbors by comparing all the shingle sets with each other.

We will apply the algorithm to solve the problem that companies such as kijiji face when companies or individuals post many copies of the same announcement—usully they want to block announcements that are near duplicates, otherwise users keep putting up announcements for the same job to show up in the top. We will work on the announcements for apartments of Problem 1.

We want to find announcements that are near duplicates. We will say that two announcements are near duplicates if the Jaccard coefficient of their shingle sets is at least 80%. We will use shingles of length 10 characters. Find values for r and b (see Section 3.4 in the book) that can give us the desired behavior. To plot the graph that gives the probability as a function of the similarity for different values of r and b you can use, for example, Wolfram Alpha.

To apply the algorithm you have the following tasks:

1. Find the near-duplicates among all the announcements of Problem 1 using LSH. Use the *full description*, if you have it, otherwise the *short description*.
2. Find the near-duplicates among all the announcements of Problem 1 by comparing them with each other.
3. Report the number of duplicates found in both cases, and the size of the intersection.
4. Report the time required to compute the near duplicates in either case.

You will need a way to create a family of hash functions. One way is to use a hash function and a code similar to the following.

```
# Implement a family of hash functions. It hashes strings and takes an
# integer to define the member of the family.
# Return a hash function parametrized by i
import hashlib
def hashFamily(i):
  resultSize = 8        # how many bytes we want back
  maxLen = 20           # how long can our i be (in decimal)
  salt = str(i).zfill(maxLen)[-maxLen:]
  def hashMember(x):
    return hashlib.sha1(x + salt).digest()[-resultSize:]
  return hashMember
```

Note that this code is an overkill because we use a cryptographic hash function, which can be very slow, even though it is not needed to be as secure. However, for the necessities of the homework we will use it to avoid having to install some external hash library.

Problem 3. In this question we will see how we can use minhashing signatures created for the Jaccard similarity of two sets, to estimate the Hamming similarity of two sets. The approach that we present is not the most efficient but it is relatively simple.

First, recall that the Hamming distance between two sets $A, B \subseteq U$ is the size of their symmetring difference: $|(A \setminus B) \cup (B \setminus A)|$. The Hamming similarity between S and T is defined as

$$S_H = 1 - \frac{|(A \setminus B) \cup (B \setminus A)|}{n},$$

where n is the total number of elements in the universe: $n = |U|$.

Let $h_1, \ldots, h_\ell$ be a set of hash functions with each $h_i : U \mapsto \{0, \ldots, k-1\}$, such that each h_i maps each element $x \in U$ to an item uniformly distributed in $\{0, \ldots, k-1\}$, independently of the other $\ell - 1$ hash functions, and independently of where it maps other elements $y \in U$. We assume that $n \gg k$.

Assume that we have minhash signatures for A and B such as those we did in the class. Namely, for a set A we have a signature

$$\text{sig}(A) = (v_1(A), v_2(A), \ldots, v_\ell(A)),$$

where you can assume that ℓ is a sufficiently large integer, and for each $i \in \{1, \ldots, \ell\}$ we have that

$$v_i(A) = \min_{x \in A} h_i(x).$$

However, you do not have access to the real sets A and B, you only know their signatures.

We divide the question of estimating the Hamming similarity into three parts.

1. Show that given a signature $\text{sig}(A)$, you can estimate $|A|$. (**Hint:** Can you express $|A|$ as the Jaccard similarity between two sets?)
2. Show how you can estimate $|A \cup B|$.
3. Estimate the similarity $S_H(A,B)$, using the signatures $\text{sig}(A)$ and $\text{sig}(B)$.