Data Mining

Homework 2

Due: 12/11/2023, 23:59

Instructions

You must hand in the homework electronically and before the due date and time.

This homework has to be done by each **person individually**.

Handing in: You must hand in the homework by the due date and time by an email to Gianluca (`decarlo@diag.uniroma1.it`) that will contain as attachment **(not links to some file-uploading server!)** a .zip file with your answers. The filename of the attachment should be
`DM_Homework_1__StudentID_StudentName_StudentLastname.zip`;
for example:
`DM_Homework_1__1235711_Robert_Anthony_De_Niro.zip`.
The email subject should be
`[Data Mining] Homework_1 StudentID StudentName StudentLastname`;
For example:
`[Data Mining] Homework_1 1235711 Robert Anthony De Niro`.
After you submit, you will receive an acknowledgement email that your project has been received and at what date and time. If you have not received an acknowledgement email within 2 days after the deadline then contact Gianluca.

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

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

Problem 1. In this exercise we will get some practice in using some Python libraries, for web page retrieval, parsing, and data analysis. We will obtain data about products available on Amazon. 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 to 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 scenario, since the pages to parse are small, but be careful if you want to use it on large XML files.

Write a program that downloads and parses product data from `https://www.amazon.it/` based on a given keyword. You can search for a product page using the URL `https://www.amazon.it/s?k=KEYWORD&page=X`, where 'KEYWORD' is the product you want to search for, and 'X' is the page number. For this exercise, use the keyword *gpu* to find various products such as GPUs, cables, GPU stands, etc.

Save the products in a tab-separated value (TSV) file, where each product occupies one line with the following details:

- Product description
- Price
- Prime product status
- URL of the product page

- Number of stars of the product
- Number of reviews

To prevent being blocked by Amazon, introduce a delay (use 'sys.sleep()') between different downloads of Amazon pages.

After collecting information, the Data Scientists have to know what dataset they are dealing with, so let's start with an Exploratory Data Analysis (EDA). What can you say about our datasets? Please summarise its main characteristics with visual and tabular methods.

Let's explore the dataset by finding simple insights regarding the products:

1. Price Ranges: Determine the price range of products within different categories.
2. Customer Reviews: Analyze customer reviews to find the products with the highest ratings
3. Primeness: is there any relation with the product being 'Prime' with its price/rating?
4. Plot the top 10 products in term of ratings
5. Plot the top 10 products in term of price

Problem 2. The next step is to build a search-engine index. First, you need to build an inverted index, and store it in a file. Build an index that allows to perform proximity queries using the cosine-similarity measure. Then build also a query-processing part, which, given some terms, will bring the most related products. For this version of the search engine, narrow the interest on the description of each product. It means that you will evaluate queries only with respect to the product's description. For each most related product computed by your search engine, return all its information stored. If your search engine returns poor results try by increasing the number of crawled pages.

Problem 3. 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 all the document 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 on the products previously crawled. We want to find products that are near duplicates, i.e. that have similar descriptions. We will say that two products 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:

(a) Find the near-duplicates among all the products of Problem 1 using LSH. Use the description.

(b) Find the near-duplicates among all the products of Problem 1 by comparing them with each other.

(c) Report the number of duplicates found in both cases, and the size of the intersection.

(d) 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.

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 4. Implement Problem 3 using Apache Spark, considering the previously downloaded web pages.

Please submit the code and a report containing examples of queries, and screenshots of the results for both short and long queries, including full product descriptions.