Finding Similar News Articles Project

This phase of the project focuses on using the Locality-Sensitive Hashing (LSH) method for finding similar items. You will work with a dataset of BBC news articles, and assuming that similar news articles talk about the same event, the aim is to use the LSH method to find these similar articles.

1 Data

The data can be found in the .zip file you are given, under the "code/data/bbc" folder. This folder contains 2225 .txt documents from the BBC news website corresponding to stories from 2004-2005.

2 Code

You are given a Python script in which the main methods of reading the data, reading the input parameters, and giving the results are implemented.
The methods that are implemented are described as following:

- **read_parameters():** Reads the input parameters from the "default_parameters.ini" parameter file. The input parameters are k, permutations, r, buckets, data, naive, t and they are described in Table 1. The method stores the input parameters in the

Parameter Name	Type	Default	Description
k	int	5	the number of words for the k-shingles of each document
permutations	int	20	the number of permutations for simulation of MinHash
r	int	2	the number or rows per band at the LSH
buckets	int	10	the number of buckets of the LSH
data	string	bbc	the **directory** of the input data, should be under "/data"
naive	boolean	false	parameter that controls the run of the naive method
t	float	0.6	similarity threshold

Table 1: Input Parameters

"parameters_dictionary" dictionary and they can be accessed as "parameters_dictionary['parameter_name']".

- **read_data(data_path):** Reads the input data that are in the "data_path" directory. The documents in the input data are stored in the "document_list" dictionary, where the key of the dictionary is the document id (integer of the name of the file) and the value is the document text. The documents then are sorted with their document ids.
- **naive():** If the parameter "naive" is "true" the naive() method is run. It calculates the similarities of all the combinations of documents and stores the similarities in the triangular array "naive_similarity_matrix".
- **jaccard(doc1, doc2):** Calculates the Jaccard similarity of two documents that are represented as sets of words.
- **get_triangle_index(i, j, length):** Calculates the triangle index for the triangular array used in the "naive()" method.

You will need to use and extend this code in order to find news articles that talk about the same event.

2.1 Important notes before you start

- Please use Python 3.8 or newer to develop you solution.
- Make sure that the code given is running on your environment **before** you start your own implementation.
- Make sure that the dataset is in a directory in the "/data" directory of your project.
- There are some default empty methods in the code where you need to implement your tasks. You can change the methods and add more if you need, but **don't change** the way the code is reading the inputs and giving the outputs.
- For testing your code while you implement it, you can create a subset of the data that you can verify if the implementation is going well. There is a small dataset given in "/data/test" if you want to use.
- If you create your own dataset, or use another existing one, make sure the documents are in a directory and have sequential integer names starting from 1.
- Since you will be dealing with string variables, use case-insensitive comparison if needed.

3 Tasks

Your task is to implement the LSH method in the code given, and experiment with the parameters of the method by following the tasks below:

1. Implement the document representations using k-Shingles (**using words**) for the documents in the data. Use k as an input parameter.
2. Using the k-Shingles from the previous task, create the signature sets for each document.

3. Using the k-Shingles Signatures sets from the previous task, implement the Permutation Simulation algorithm of the MinHash Signature Matrix of the documents. Create random a, b and p coefficients, with p being a prime number. Use the number of permutations as input parameters.

4. Using the MinHash Signature Matrix from the previous task, implement the LSH method to create a list of candidate document pairs that are possibly similar. Use the number of rows r and the number of buckets as input parameters.

5. For the candidate document pairs from the previous task, calculate the document signature sets similarity using the fraction of the hash functions which they agree, i.e.
$$similarity(d_1, d_2) = \frac{\#(h_i(d_1) == h_i(d_2))}{permutations}$$

6. After calculating the similarity for the candidate document pairs from the previous task, count how many document pairs have similarity over the threshold input parameter t and print their ids. Also, count how many of the pairs are false positive and false negative using the naive method for the accurate similarity.

4 Report

In your report briefly describe the following:

1. Using parameter $k = [1, 5, 10]$, and the default values for the rest of the parameters, how does the parameter k affect the total number of shingles? Is having more shingles better for identifying as many as possible document pairs that have similarity over the threshold, and why?

2. Using $permutations = [10, 20, 50]$, and the default values for the rest of the parameters, how does the number of permutations affect the number of false positive and negative results? Are more permutations giving more or less of false positives and negatives, and why?

3. Using $r = [2, 5, 10]$, and the default values for the rest of the parameters, what is the optimal number of rows so that we do the least amount of comparisons?

4. Using $buckets = [10, 20, 30]$, and the default values for the rest of the parameters, how is the number of buckets affecting the number of the candidate document pairs? Are more buckets giving less false positives and negatives, and why?

5. Using the naive() method, parameter $naive = true$, count how many document pairs are checked for similarity for the naive() and how many for the LSH. How efficient is the LSH method compared to the simple naive method both on terms of document pairs and on running time? Why?

6. (Optional) Are there any other parameter combinations that give better results? Are the parameters data-related?

5 Delivery

You should deliver a ZIP file containing:

- the report in PDF format $(50\% points)$
- the source code in .py file(s) $(50\% points)$

Your project will be evaluated taking into consideration the following points:

- if the code runs using the command: “`python3 lsh.py`”
- if the code doesn’t include external libraries for the tasks.
- if the code doesn’t have bugs.
- if the code gives the correct results.
- if the answers in the report have **justification**, i.e. answering the “why” questions.