DataMining ID2222 - Homework 1
Finding Similar Items: Textually Similar Documents

Sherly Sherly and Anna Martignano

November 24, 2019

1 Our Solution

In this section, we will describe the approach that was taken to complete the tasks. Section 1.1 describes the pre-processing method taken for all the text data involved in this project. In Section 1.2, 1.3 and 1.4 We will be describing the three tasks: Shingling, MinHashing and Locality-Sensitive Hashing respectively.

1.1 Dataset and Cleaning

The dataset used for this assignment consists of a collection of ten books of the author Joseph Conrad(1857 -1924), a famous Polish writer regarded as one of the greatest modern writers of England even if his mother language was not English. These books has been published online at this website which is maintained by the *Project Gutenberg*, the aim of this project is to make available the largest collection of eBooks.

The following cleaning operations has been performed as part of the Shingling class:

- Conversion to lower case.
- Substitution of the new line command with space command.
- Punctuation removal keeping only letters, numbers, and single spaces.
- Removal of consecutive spaces, only one space among words is kept.
- Substitution of the spaces with the underscores.

1.2 Shingling Class

A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the document. In this project, the tokens are defined by characters and hence a k-shingle is a token of k characters. The code snippet below describes the implementation of the shingling method.

```
class Shingling:
    def __init__(self, k):
        <function>
        """
        To initialize this class is required the k-grams
        parameter in order to specify the dimension
        of the shingles.
        """

    def _clean(self, doc):
        <function>
        """
        Cleaning of the documents as per Section 1.1
        """

    def _tokenize(self, doc):
        <function>
        """
        Tokenizing the document into k-grams
        Constructs the shingles based on the k-characters
        of the document received as an input and return
        the sets of the hashed shingles
        """

    def _hash(self, shingle):
        <function>
        """
        Encodes the input shingle with a 32bit
        integer representation of it
        """

    def generate_shingles(self, doc):
        <function>
        ""
```

```
        Wraps around other functions to clean
        then tokenize the document
        """

    def generate_shingles_for_docs(self, docs):
        <function>
        """
        Iterate through all documents and
        generate shingles
        """

    @staticmethod
    def compare_sets(s1, s2):
        <function>
        """
        Generates the Jaccard Similarity between
        two sets of Shingles
        """
```

The methods **generate_shingles** and **generate_shingles_for_docs** are pretty similar, both they first call the clean method and then the tokenize one. The only difference is that generate_shingles_for_docs instead of returning the computed set of hashed shingles it stores the results in a dictionary internal with respect to the class. The computation of the Jaccard Similarity of sets has been implemented as an internal static method of the Shingling Class called **compare_sets**, it computes the Jaccard Similarity of the two sets A and B if both of them are not empty using the following formula $\frac{A \cap B}{A \cup B}$. We have also implemented the Jaccard Similarity as an external function to assess the performance of this computation, this function generate a similarity matrix of the input sets of hashed shingles.

1.3 MinHashing Class

The method min hashing takes in shingles and generates signatures based on predefined number of hashes by users. The code below shows the implementation of MinHashing method.

```
class MinHashing:

    def __init__(self, n, max_shingle_ID = 2**32-1):
        self.n = n # number of hashes
```

```
        """
        Initialization with the number of hashes to be used
        to compute the signature
        """

    def generate_coeffs(self):
        <function>
        """
        Generate the coefficients a and b n times where n is the
        number of hashes initialized. This method creates the
        coefficients list. It is important to ensure that all
        the n coefficients are different, i.e. there should be
        distinct values for the coefficients list of a and
        for the coefficients list of b.
        """

    def _minHash_function(self, pos, x):
        <function>
        """
        Computes the following MinHash function: (ax+b) mod c,
        in which a and b are randomly generated coefficients
        and c is the next prime bigger than 2**32 1  since the
        previous class Shingling create sets of 32 bit integers.
        """

    def generate_signature(self, shingle_set):
        <function>
        """
        Creates the signatures of the input set ofshingles
        calling the internal minHash function n times.
        """

    def generate_doc_signatures(self, shingles):
        <function>
        """
        Generates the signature for all shingles.
        Returns a dict: {k: v} where k is the doc id
        and v is the signature
        """

    @staticmethod
```

```
    def compare_signatures(s1, s2):
        <function>
        """
        Compares  pairwise  the elements of the two input
        signatures and it returns the fraction of the equal
        elements over the total signatures length. This
        comparison is made only if the signatures have
        the same length.
        """
```

The comparison between signatures to determine distance is not exactly the Jaccard similarity. For two sets S1 and S2, the probability Pr[hmin(S1) = hmin(S2)] is the fraction of the minHash functions in which they agree i.e. the number of rows in the signature matrix with the same values in S1 and S2 columns divided by the total number of rows in the signature k. This probability is an estimate of the Jaccard similarity of the two corresponding sets.

$$Pr[hmin(S1) = hmin(S2)] \approx Sim(S1, S2) \tag{1}$$

1.4 LSH Class

```
class LSH:

    def __init__(self, band_size, row_size, threshold):
        self.band_size = band_size
        self.threshold = threshold
        self.row_size = row_size
        """
        The parameters required for a matrix M is the number
        of bands b, the row  size  r  and  a  similarity
        threshold  t to  filter  the  candidate  pairs.
        """

    def get_lsh(self, signature):
        <function>
        """
        Create  an  hash  of  the  signatures
        for  each  signature  band
        """

    def get_lsh_for_docs(self, signatures):
```

```
        <function>
        """
        Create a LSH internal dictionary of the given input
        signatures dictionary, the keys remain the same but
        the values is the ordered list of buckets per
        each bands.
        """

    def generate_candidate_pairs(self):
        <function>
        """
        Uses a sliding count to perform the comparison,
        in this way each document is com-pared with all
        the other documents only once. For each pair of
        documents, it is count how many times they belong
        to the same bucket, and if this count is greater
        than the threshold multiplied by the total
        number of bands, this pair is stored in a
        dictionary of candidate pairs.
        """
```

For performance reasons, in the computation to generate the candidate pairs, we have decided not to compute the similarity fraction for each pair of documents, i.e. if $count/b > t$, but simply multiply the threshold by the number of bands only once and then simply compare it to the count, i.e. if $count > t \times b$.

2 Results and Scalability Performance

Based on the three methods implemented: shingling, min-hashing and LSH, we observed that the bottleneck lies in the min-hashing step. The table below denotes the time taken for each of the steps.

Method	Time
Shingling	2.72 s
Min-Hashing	160 s
LSH	0.1 s

Our dataset consists of text from books which has a lot of characters. As such, there are many shingles being generated and in the min-hashing algorithm, we will be performing the hashing to signatures n times according to the parameter n that we supplied.

3 How to run the code

To develop our application, we have used Jupyter Notebook. Therefore, it is sufficient to launch the Jupyter Notebook environment from the conda shell (3.x) and simply run all the code cells in a consecutive order. Please check that the input documents are in the same file_path as specified by the load function, which is the conradbooks folder and it must be positioned at the same file_path level of the .ipynb notebook.