Task 1 Report

Task

This task is to identify pairs of similar articles using Locality Sensitive Hashing (LSH) based on cosine similarity.

Parameters

`n_hashes` : determines the number of rows (or hash functions) in the MinHash signature matrix. A higher number of hash functions increases the precision of similarity estimation but will increase computational cost. After testing, with 100 hash functions, the MinHash signatures provide a reliable approximation of document similarity without excessive computational overhead

`band_size`: LSH divides the MinHash signature matrix into bands of `band_size` rows each. The number of bands `n_bands = n_hashes / band_size`

- Larger `band_size` will cause higher recall but increases false positives.
- Smaller `band_size` will cause fewer false positives but may miss some true positives.

With `n_hashes = 100` and `band_size = 20`, there are 5 bands. After testing, this setup balances the trade-off between precision and recall.

`shingle_size`: Shingles are overlapping word sequences of length `shingle_size`. Smaller shingle sizes may lead to excessive matches and noise, while larger sizes (e.g., 3 or more)may miss similarities. By testing, 2 is the best choice.

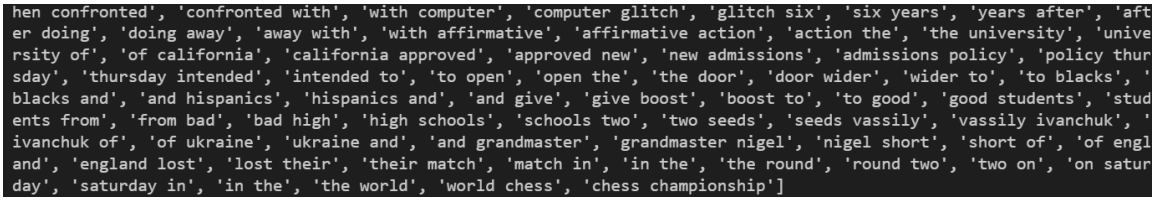
1. Data pre-processing

The pre-processing step cleans the text to prepares the dataset for analysis:

- Removing punctuation, short words and numbers, and converting the words to lowercase
- Each line in the dataset is parsed to separate the `t_id` and article content by `line.split(" ", 1)`.
    - For example, in `all_articles.txt`, content: **t8470 IAEA governors met here Monday to discuss how to improve detection of clandestine…** are spilt into `t_id = t8470` and `content = IAEA governors met here Monday to discuss how to improve detection of clandestine…`

2. Generating shingles

- Articles are divided into overlapping sequences of words (i.e. shingles) of a shingle size (e.g. `shingle_size = 2`).
- Take a sentence in `all_articles.txt` for instance, **IAEA governors met here Monday to discuss...** will divided into words-based shingles: `IAEA governors`, `governors met`, `met here`, `here Monday`, `Monday to`, `to discuss`..

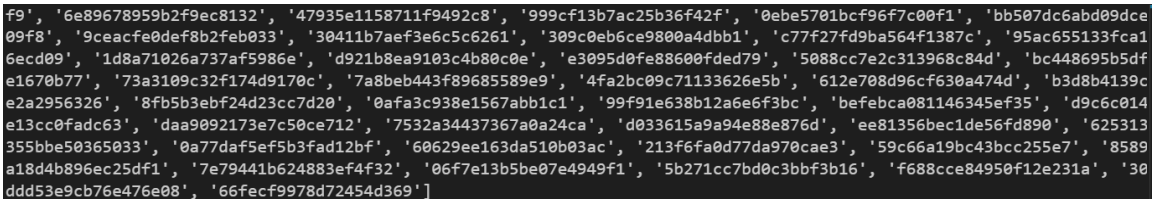


```
hen confronted', 'confronted with', 'with computer', 'computer glitch', 'glitch six', 'six years', 'years after', 'aft
er doing', 'doing away', 'away with', 'with affirmative', 'affirmative action', 'action the', 'the university', 'unive
rsity of', 'of california', 'california approved', 'approved new', 'new admissions', 'admissions policy', 'policy thur
sday', 'thursday intended', 'intended to', 'to open', 'open the', 'the door', 'door wider', 'wider to', 'to blacks', '
blacks and', 'and hispanics', 'hispanics and', 'and give', 'give boost', 'boost to', 'to good', 'good students', 'stud
ents from', 'from bad', 'bad high', 'high schools', 'schools two', 'two seeds', 'seeds vassily', 'vassily ivanchuk', '
ivanchuk of', 'of ukraine', 'ukraine and', 'and grandmaster', 'grandmaster nigel', 'nigel short', 'short of', 'of engl
and', 'england lost', 'lost their', 'their match', 'match in', 'in the', 'the round', 'round two', 'two on', 'on satur
day', 'saturday in', 'in the', 'the world', 'world chess', 'chess championship']
```

- Then each shingle is hashed using the MD5 algorithm to create a compact representation. MD5 mathematical function:

$$MD5(x) = H(A, B, C, D, message_{blocks})$$

- MD5 can produces a fixed-size output regardless of the input size, so shingles of varying sizes are reduced to a uniform hash value, which simplifies indexing and comparisons. Also, the same input always results in the same hash.



```
f9', '6e89678959b2f9ec8132', '47935e1158711f9492c8', '999cf13b7ac25b36f42f', '0ebe5701bcf96f7c00f1', 'bb507dc6abd09dce
09f8', '9ceacfe0def8b2feb033', '30411b7aef3e6c5c6261', '309c0eb6ce9800a4dbb1', 'c77f27fd9ba564f1387c', '95ac655133fca1
6ecd09', '1d8a71026a737af5986e', 'd921b8ea9103c4b80c0e', 'e3095d0fe88600fded79', '5088cc7e2c313968c84d', 'bc448695b5df
e1670b77', '73a3109c32f174d9170c', '7a8beb443f89685589e9', '4fa2bc09c71133626e5b', '612e708d96cf630a474d', 'b3d8b4139c
e2a2956326', '8fb5b3ebf24d23cc7d20', '0afa3c938e1567abb1c1', '99f91e638b12a6e6f3bc', 'befebca081146345ef35', 'd9c6c014
e13cc0fadc63', 'daa9092173e7c50ce712', '7532a34437367a0a24ca', 'd033615a9a94e88e876d', 'ee81356bec1de56fd890', '625313
355bbe50365033', '0a77daf5ef5b3fad12bf', '60629ee163da510b03ac', '213f6fa0d77da970cae3', '59c66a19bc43bcc255e7', '8589
a18d4b896ec25df1', '7e79441b624883ef4f32', '06f7e13b5be07e4949f1', '5b271cc7bd0c3bbf3b16', 'f688cce84950f12e231a', '30
ddd53e9cb76e476e08', '66fecf9978d72454d369']
```

3. Generating MinHash Signature

- A MinHash signature matrix is computed using `n_hashes = 100` hash functions. This step reduces the dimensionality of the shingles and preserving similarity relationships.
- The random coefficients for the hash functions are generated deterministically for reproducibility.
- **Initialization:**
  - `a` and `b`: Random coefficients for hash functions for the variety in hashing.

    - `p`: A large prime number for modulus.
    - `minhash_matrix`: A 2D matrix initialized to `inf`, with dimensions `(n_hashes x num_docs)`.
- `shingles_indices` maps each shingle to its unique index in `shingles_dict`.
- For each document, the index of its shingles are retrieved using `shingles_indices`.
- **Applying Hash Functions:**
    - For each shingle index in the document, hash values are computed using:

      $$h(x) = (a \cdot x + b) \% p$$

      where x is the shingle index, a and b are random coefficients, and p is a large prime number.
    - This operation is vectorized using `np.outer(a, indices)` for efficiency.
- For each hash function, the minimum hash value across all shingles in the document is stored in the matrix.

4. Locality Sensitive Hashing (LSH)

- The MinHash signature matrix is divided into bands by band size (e.g. `band_size = 20`), where each band contains `band_size` rows.
- Then, each band is hashed into buckets.
- For each document, extract its MinHash values for the current band (`band_slice`).
- Represent these values as a tuple (`band_key`), which acts as the hash key for the band.
- `band_hashes` is a dictionary where each key is a band index, and the values are sub-dictionaries. Each sub-dictionary maps `band_key` to a list of document index (i.e. `doc_idx`) that share the same `band_key`.
- Output a nested dictionary structure that maps bands and keys to document indices.
- Articles with similar signatures in at least one band are grouped as candidate pairs.

5. Filtering Candidate Pair

- Use `CountVectorizer` to construct a sparse document-term matrix for the articles.
- Extract the content of the documents corresponding to the candidate pairs.
- Compute the cosine similarity for all candidate pairs:

$$sim(X, Y) = \frac{X \cdot Y}{||X||\ ||Y||}$$

  - Only pairs with a similarity score greater than `0.8` are retained.

6. Find Out Similar Document

- After using `filter_similar_pairs` , convert document indices in the final result back to document IDs

7. Output

- Similar article pairs are written to `result.txt` with their `t_id`.

Results and Evaluation

Results

The program outputs all similar pairs into a file (`result.txt`) for review. There are total 80 similar pairs. The result can be view attachment 1.

Runtime

We have run the code for 5 times, and the total time are: 4.76 seconds, 4.62 seconds, 4.89 seconds, 5.09 seconds and 4.76 seconds. The average time is 4.825 seconds.

```
Total unique shingles: 718043
Document-term matrix shape: (100, 10000)
Optimized MinHash computation time: 1.55 seconds
Total get_band_hashes time: 0.17 seconds
Number of candidate pairs: 90
Total filter_similar_pairs time: 0.71 seconds
Total get_similar_docs time: 4.34 seconds
Results written to result.txt.
Total time: 4.76 seconds
(MLAI) (base) winght@WinghtdeMacBook-Pro Task-1
da3/envs/MLAI/bin/python "/Users/winght/Desktop/
ject/FTEC4005-project-3/FTEC4005-project/Task-1
Total unique shingles: 718043
Document-term matrix shape: (100, 10000)
Optimized MinHash computation time: 1.44 seconds
Total get_band_hashes time: 0.17 seconds
Number of candidate pairs: 80
Total filter_similar_pairs time: 0.70 seconds
Total get_similar_docs time: 4.22 seconds
Results written to result.txt.
Total time: 4.62 seconds
(MLAI) (base) winght@WinghtdeMacBook-Pro Task-1
da3/envs/MLAI/bin/python "/Users/winght/Desktop/
ject/FTEC4005-project-3/FTEC4005-project/Task-1
Total unique shingles: 718043
Document-term matrix shape: (100, 10000)
Optimized MinHash computation time: 1.47 seconds
Total get_band_hashes time: 0.18 seconds
Number of candidate pairs: 80
Total filter_similar_pairs time: 0.70 seconds
Total get_similar_docs time: 4.48 seconds
Results written to result.txt.
Total time: 4.89 seconds
(MLAI) (base) winght@WinghtdeMacBook-Pro Task-1
da3/envs/MLAI/bin/python "/Users/winght/Desktop/
ject/FTEC4005-project-3/FTEC4005-project/Task-1
Total unique shingles: 718043
Document-term matrix shape: (100, 10000)
Optimized MinHash computation time: 1.72 seconds
Total get_band_hashes time: 0.17 seconds
Number of candidate pairs: 80
Total filter_similar_pairs time: 0.71 seconds
Total get_similar_docs time: 4.67 seconds
Results written to result.txt.
Total time: 5.09 seconds
(MLAI) (base) winght@WinghtdeMacBook-Pro Task-1
da3/envs/MLAI/bin/python "/Users/winght/Desktop/
ject/FTEC4005-project-3/FTEC4005-project/Task-1
Total unique shingles: 718043
Document-term matrix shape: (100, 10000)
Optimized MinHash computation time: 1.44 seconds
Total get_band_hashes time: 0.17 seconds
Number of candidate pairs: 80
Total filter_similar_pairs time: 0.69 seconds
Total get_similar_docs time: 4.34 seconds
Results written to result.txt.
Total time: 4.76 seconds
```

Conclusion

This implementation of LSH for identifying similar articles is designed to be both fast and accurate. By using hashing techniques and cosine similarity, the method effectively scales to large datasets while ensuring precision.

Attachment

1. result.txt