D-002

Detecting Reused Contents in Text Documents

Pei Wang† Chuan Xiao‡ Yoshiharu Ishikawa†
† Graduate School of Information Science, Nagoya University
‡ Institute for Advanced Research, Nagoya University

1 Introduction

As digitalization developes, plagiarism is very cheap to be achieved recently. Identifying text reuse enables us to find original sources of facts and track information flow, and thus becomes an important task in text analysis. It is related to plagiarism detection which has been extensively studied. But the major difference is plagiarism detection matches nearly identical documents, while the replication in text reuse is often done partially, i.e., one or more sentences or paragraphs are copied but overall the documents differ significantly. There are two major drawbacks in the existing methods: (1) The segmentation is sensitive to edits, e.g., combining sentences or changing word orders, and hence many meaningful results are missed. (2) The segment matching is time-consuming if we want to find results with minor differences.

We develop a novel method for content reuse detection in text documents. Our basic idea is to use sliding windows. A pair of sliding windows that differ by no more than τ words is regarded as reused contents. We develop a filter-and-refine method to solve this problem, i.e., first find a set of promising candidates and then verify if they satisfy the constraint.

2 Preliminaries

We define a document r as a sequence of tokens from a finite universe $\mathcal{U}$ *. A sliding window x of size w is w consecutive tokens in a document r, denoted as $x \sqsubseteq r$. We also use $r[i\,..\,j]$ to denote a sliding window that starts from the i-the token and ends with the j-th token in r. By regarding each sliding window as a multiset of tokens, the overlap similarity between the two sliding windows x and y is defined as $overlap(x, y) = |x \cap y|$. Multiplicities are considered when computing the intersection of two multisets.

From the perspective of dissimilarity, we define the distance between two sliding windows as $d(x, y) = w - overlap(x, y)$. E.g., the distance between two sliding windows $x =$ `as soon as possible` and $y =$ `yes soon yes as` is $d(x, y) = w - |x \cap y| = 4 - 2 = 2$.

Given a set of data documents R, a query document q, a sliding window size w, our task is to find all sliding window pairs $\langle x, y\rangle$, such that x is from a data document, y is from the query document, and the distance between x and y is no more than a given threshold τ; i.e., $\{\, \langle x, y\rangle \mid x \sqsubseteq r, r \in R, y \sqsubseteq q, d(x, y) \le \tau \,\}$.

For example, consider two data documents "`have no more than`" and "`have more than two`", and a query document "`no less than two`". The windows size is 3 and the threshold is 2. Two results will be returned: $\langle$`no more than`, `no less than`$\rangle$ and $\langle$`more than two`, `less than two`$\rangle$.

3 K-wise Signature Scheme

A naïve method to solve the problem is to enumerate all sliding windows for the data documents and the query document, and compare every window pair to compute their distance. The naïve method exhaustively compare all sliding windows and thus becomes infeasible for large amount of data. One remedy is to consider promising window pairs only by leveraging the prefix filtering principle [1]:

Theorem 1 (Prefix Filtering Principle). Consider two multisets x and y, and their tokens are sorted in a global order $\mathcal{O}$ of the token universe $\mathcal{U}$. Let the prefix of x be the first $\tau+1$ tokens of x. If $d(x, y) \le \tau$, then the prefix of x and the prefix of y must share at least one token.

The prefix filtering principle indicates that if we sort the tokens in the global order $\mathcal{O}$, we only need to consider the pairs of windows that share at least one token in their prefixes. The prefix filtering principle is further extended to the first $\tau + k$ tokens [3, 2] :

Theorem 2 (Extended Prefix Filtering Principle). Let k-prefix of x be the first $\tau+k$ tokens of k. If $d(x, y) \le \tau$, then the k-prefix of x and the k-prefix of y must share at least k token.

Due to the stricter constraint, using k-prefixes exhibits better selectivity than using 1-prefixes. Based on the extended prefix filtering principle, an immediate solution is to build an inverted index for the first $\tau + k$ tokens of all the sliding windows. An inverted index is a data structure that maps a token t to a list of window identifers (called postings list) that contain t. The main problem of this method is that the postings lists of some tokens can be very long because these tokens are frequent in the data documents. Accessing long postings lists incur significant overhead. Next we propose the notion of k-wise signature to to alleviate this problem.

A k-wise signature is a combination of k tokens. Based on the above observations, for each window we generate all k-combinations of the tokens in its k-prefix. If two windows satisfy the distance constraint, they share at least one identical signature.

*We tokenize documents into English words in our examples, but the proposed method is independent of tokenization scheme.

For example, given a window's 3-prefix $\{a, b, c, d, e\}$, $\{abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde\}$ are generated as its 3-wise signatures. If its distance to a query window is no larger than τ, there must be at least one identical 3-wise signature between the two windows.

To generate k-wise signature for each sliding window, an intuitive baseline is to materialize all sliding windows, compute signatures, and build inverted index for the signatures. However, this method suffers from the following two drawbacks: (1) Materializing all the sliding windows is time-consuming. (2) Consecutive sliding windows share many k-wise signatures. Common signatures are generated multiple times.

We propose a dynamic index generation method to fix above problems. The intuition is that instead of mapping k-wise signatures to a list of windows in the inverted index, we map each signature to an interval $[a, b]$, indicating that every window with starting position in the range of $[a, b]$ has this signature. For every data document, we get the first window by reading the first w tokens into a buffer, and sort them by the global order and mark its window ID as an interval starting point. k-wise signatures are generated. An interval with left bound 1 is created for each of these k-wise signatures. When we move to the next window, the first token of the previous window is deleted from the buffer and its rank before deletion is returned. Then we insert the last token of the current window into the buffer and its rank is returned as well. If either rank is in within $\tau + k$, we close the relevant signatures' intervals by setting their right bounds, and open new intervals for new signatures.

4 Candidate Generation

We now have every signature's inverted list which is composed of several corresponding window intervals. We would like the candidate IDs to be in form of (queryWindowID, candidate intervals) for the smooth of verification. We must note that by simply traversing the query's inverted lists and push back candidate window intervals, there would be many duplicate elements. We here trim the candidate lists for each query window ID: combine the overlap intervals and keep them in order. For example, for query window ID 2, we may get a trimmed list: (10, 15), (18, 20), (25, 40), windows of these intervals are query window 2's candidate windows. However for (10, 15) and (18, 20), it may save computations if we combine them as one interval(10, 20), namely adding in some false positives on purpose. We explain this trick in next section.

5 Candidate Verification

We explain the method to verify candidates efficiently with hash map data structure in this section.

We continue with above example. For query window 2, suppose its candidate intervals are (10, 15), (18, 20), (25, 40). In intialization phase, build a hash map for query window 2 whose key is token and value is token's appearance times in current window(save a query map copy). Then traverse data window 10 one by one. If current token is not in query map, then set this token's appearance as -1. Otherwise if the token's appearance is not bigger than 0, then minus 1. If the token's appearance is bigger than 0, this means query window and data window has similar token and plus 1 for the counter. We finally get an initial counter number sim(query window 2, data window 10). To make use of the continuity of sliding windows, we don't have to start from scratch again for data window 11. We have a look at the kicked out token t_1 (the first token in window 10) and newcoming token t_2 (the last token in window 11), minus one for $map[t_1]$ and plus one for $map[t_2]$, rewrite the counter only if $map[t_1] > 0$ and $map[t_2] \geq 0$. Excute this operation until interval ends (namely data window 15 here). As for new candidate window interval (18, 20), since a copy of original query window map has been saved, we can repeat above operations for this new interval smoothly. Note here data window 15 and 18 is actually very near and it is potentially not cost-effective enough. The cost of initializing a new interval is $2w$ (traverse the starting window + copy the original query map), and the cost of processing a token in an interval is 2. So only when window interval gap is no larger than w, we use the above method. Otherwise we open a new hash map for verification.

6 Conclusion

In this paper, we proposed an effective exact content detection method which is derived from the state-of-art prefix-filterting and verification framework. It works efficiently when window size is not too large and threshold is not too loose. For each window, we have to computer $\binom{k+\tau}{k}$ number of signatures, which is too big to afford when window gets long or prefix becomes long. We leave it as a future problem to support these cases.

7 Acknowledgements

This research is supported by KAKENHI(25280039).

Reference

[1] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.

[2] Jiannan Wang, Guoliang Li, and Jianhua Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *ACM SIGMOD*, pages 85–96, 2012.

[3] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient similarity joins for near-duplicate detection. *ACM TODS*, 36(3), 2011.