Plagiarism checker

Textindexierung

Daniel Hoske Stefan Walzer Geoffroy Deneuville

7th of July 2013

Goal

Goal:

- Software for analysing whether a document is a plagiarism
- Visualization of results for manual testing
- Answer 1-to-n queries: Given a large set of reference documents D and a query document q, does q plagiarise one of the documents in D?

Goal

Goal:

- Software for analysing whether a document is a plagiarism
- Visualization of results for manual testing
- Answer 1-to-n queries: Given a large set of reference documents D and a query document q, does q plagiarise one of the documents in D?

Approach:

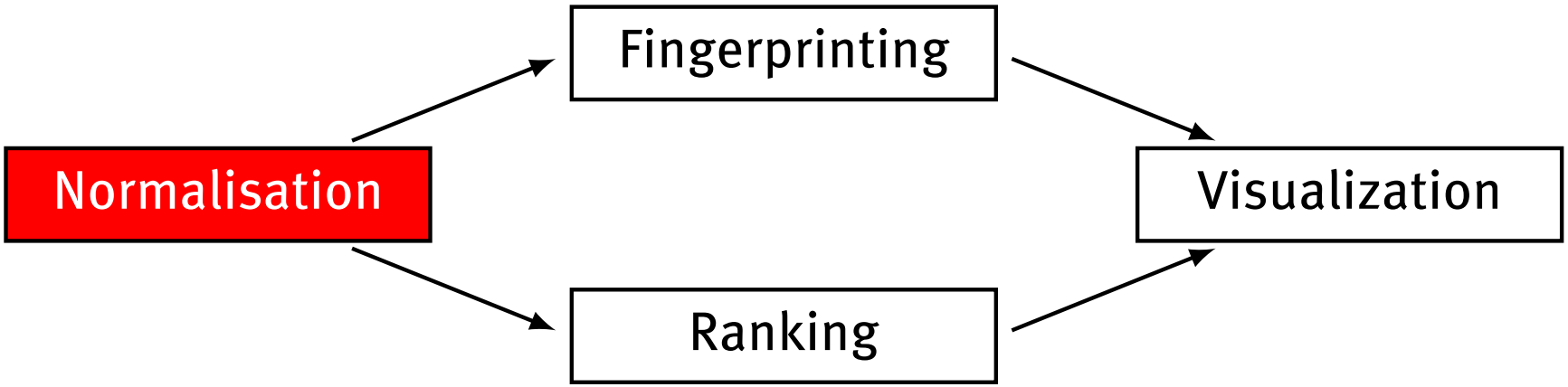
- Compute similarity score of q with every document $d \in D$ s.th. higher scores signify a higher chance of plagiarism

$\Rightarrow$ Can only preselect likely plagiarisms for manual checking

Contents

Contents

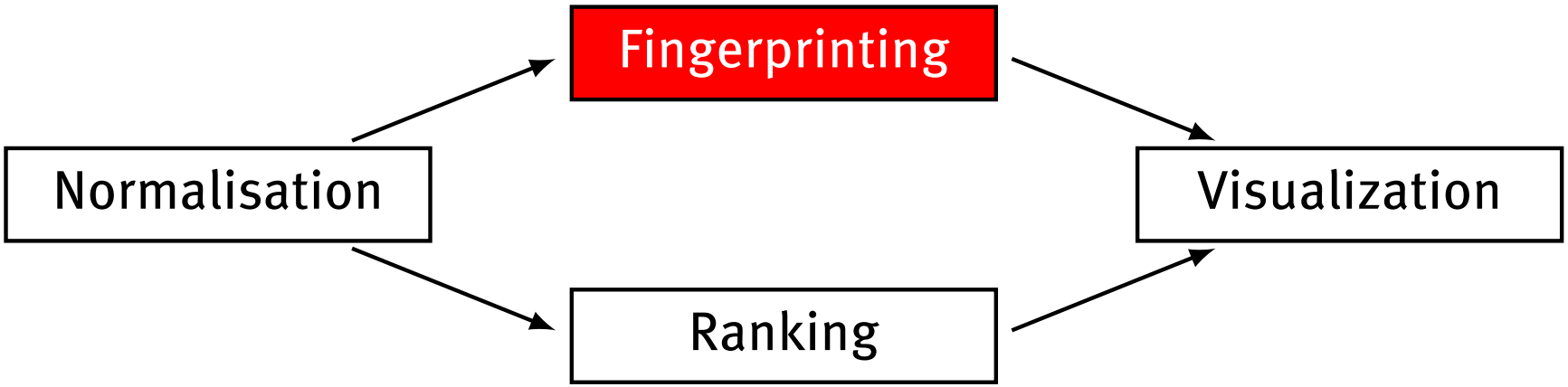
Overview



Parse text into words and remove stop words

Based on Hoad and Zobel, 2003.

Overview

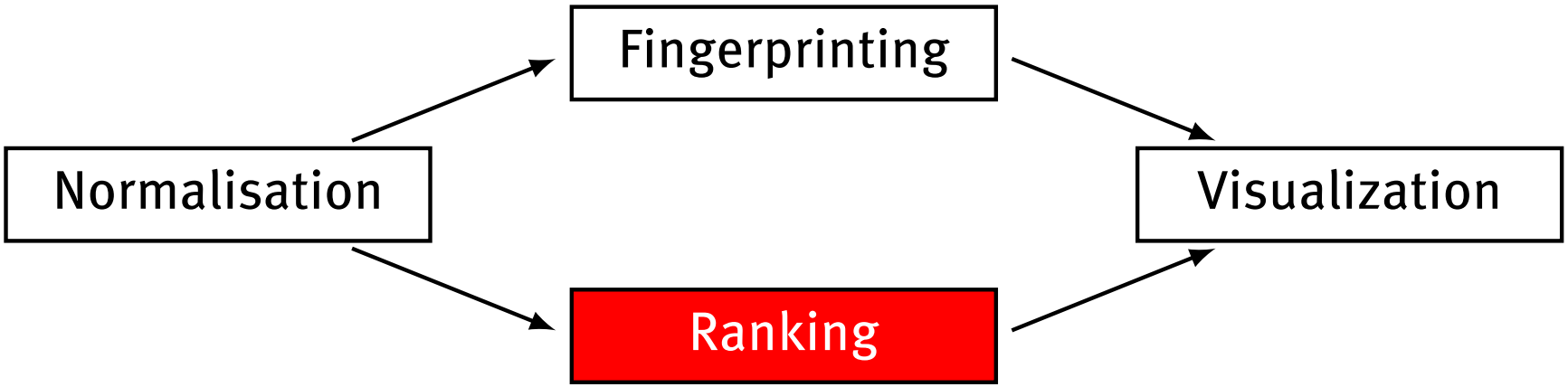


Compact description (fingerprint) of query that can be compared to the fingerprints of the references

---

Based on Hoad and Zobel, 2003.

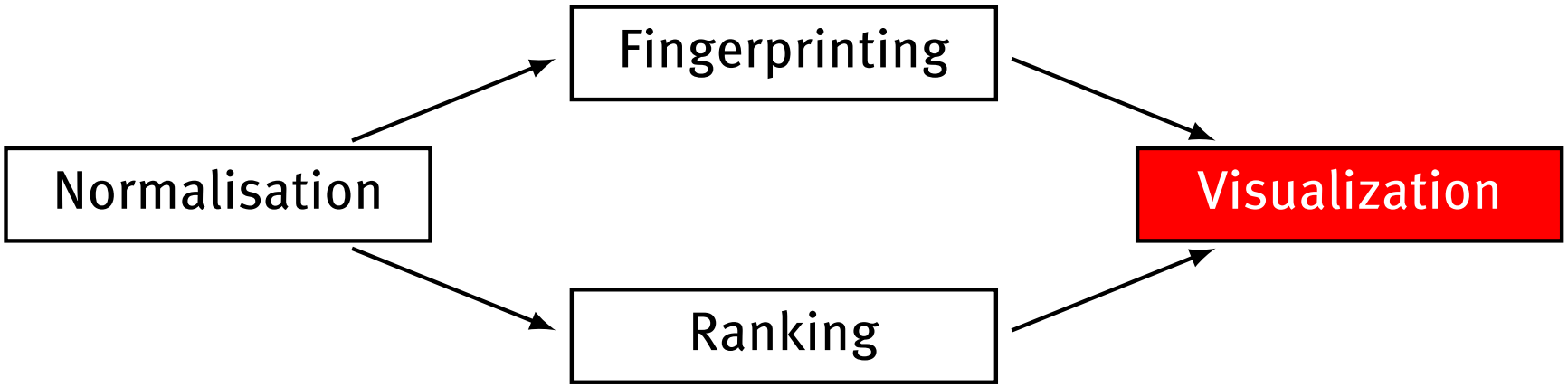
Overview



Similarity measures based on word frequencies in the texts

Based on Hoad and Zobel, 2003.

Overview



Visual report for manually checking the relevance of the results

---

Based on Hoad and Zobel, 2003.

Ranking

Variants of frequency based measures between q and d:

1	$\frac{1}{1+\lvert f_d - f_q\rvert} \cdot \sum_{t \in q \cap d} \frac{\ln(N/f_t)}{1+\lvert f_{d,t} - f_{q,t}\rvert}$
2	$\frac{1}{1+\ln(1+\lvert f_d - f_q\rvert)} \cdot \sum_{t \in q \cap d} \frac{\ln(1+N/f_t)}{1+\lvert f_{d,t} - f_{q,t}\rvert}$
3	$\frac{1}{1+\ln(1+\lvert f_d - f_q\rvert)} \cdot \sum_{t \in q \cap d} \frac{\ln(1+N/f_t)\cdot(f_{d,t}+f_{q,t})}{1+\lvert f_{d,t} - f_{q,t}\rvert}$
4	$\frac{1}{1+\ln(1+\lvert f_d - f_q\rvert)} \cdot \sum_{t \in q \cap d} \frac{\ln(N/f_t)}{1+\lvert f_{d,t} - f_{q,t}\rvert}$
5	$\frac{1}{1+\ln(1+\lvert f_d - f_q\rvert)} \cdot \sum_{t \in q \cap d} \frac{N/f_t}{1+\lvert f_{d,t} - f_{q,t}\rvert}$

Glossary:

- D document collection
- q query document
- d reference document
- f_d number of words in reference d
- f_q number of words in query q
- N number of documents in D
- f_t number of documents in D containing word t
- $f_{q,t}$ number of occurrences of word t in query q
- $f_{d,t}$ number of occurrences of word t in reference d

Ranking: Variant 1

$$\underbrace{\frac{1}{1+|f_d - f_q|}}_{\text{normalization}} \sum_{t \in q \cap d} \underbrace{\frac{\ln(N/f_t)}{1+|f_{d,t} - f_{q,t}|}}_{\text{per-word contribution}}$$

- Plagiarism should have similar word frequencies
- Rare words more useful as discriminator
- Globally plagiarised text should be of similar length

Ranking: Variant 1 vs. Variant 5

$$\frac{1}{1+|f_d - f_q|} \cdot \sum_{t \in q \cap d} \frac{\ln(N/f_t)}{1+|f_{d,t} - f_{q,t}|}$$

vs.

$$\frac{1}{1+\ln(1+|f_d - f_q|)} \cdot \sum_{t \in q \cap d} \frac{N/f_t}{1+|f_{d,t} - f_{q,t}|}$$

- Rare words more important
- Differences in text lengths less important

Fingerprinting

- Fingerprints of a document are sets of integers (minutiae) representing the document
- Comparing fingerprints allows detection of (local) plagiarisms
- Generation: select substring of some fixed length and apply hash-function to it

$\Rightarrow$ Number of matching minutiae is score

Fingerprinting

- Fingerprints of a document are sets of integers (minutiae) representing the document
- Comparing fingerprints allows detection of (local) plagiarisms
- Generation: select substring of some fixed length and apply hash-function to it

$\Rightarrow$ Number of matching minutiae is score

- Design decisions:
  - Hash-function
  - Size of selected substrings
  - Strategy for picking substrings

Contents

Technology overview

Languages (& other technologies):

- Simple command line tool in C++ (& Boost)
- Refinement of visualization with JavaScript (& jQuery)

Usage:

- Input: File containing file names of queries, file containing file names of references
- Output: Report as HTML document per query as well as similarity scores on the console (if enabled)

Normalization

- Iterate with regular expression `[\w]([\w']|(\s*-\s*))+` over text to find words
- Turn to lower case, filter out stop words (read from fixed file)
- Remember start and stop indexes of matches

```
struct Normalized {
    string name;
    vector<string> words;
    vector<int> indexes, end_indexes;
};
```

Similarity measures

- Generic interface:

```
class Algorithm {
public:
    virtual void add_reference(const Normalized& ref) = 0;
    virtual void visualize(ostream& os, /* ... */) const = 0;
    virtual vector<double> compute(const Normalized& query) const = 0;
};
```

- Implemented by `Fingerprinting` and `Ranking`

Ranking

- Save dictionary in inverted index (own Trie implementation)
- Per word: List of pairs (reference id, frequency in reference)

```
typedef Trie<vector<pair<int, int>>> Freqs;
```

Ranking

- Save dictionary in inverted index (own Trie implementation)
- Per word: List of pairs (reference id, frequency in reference)

```
typedef Trie<vector<pair<int, int>>> Freqs;
```

- Per query: compute scores for all references at the same time
- Normalisation: divide by score of query with itself

Ranking

- Save dictionary in inverted index (own Trie implementation)
- Per word: List of pairs (reference id, frequency in reference)

```
typedef Trie<vector<pair<int, int>>> Freqs;
```

- Per query: compute scores for all references at the same time
- Normalisation: divide by score of query with itself
- Generic similarity measure with exchangeable functions:

$$\text{Normalization}(\dots) \sum_{t \in d \cap q} \text{PerWord}(\dots)$$

```
typedef function<double(int, int)> NormalisationFunc;     /* f_d, f_q */
typedef function<double(int, int, int, int)> PerWordFunc; /* N, f_t, f_dt, f_qt */
```

Fingerprinting: Design choices

Hash of string $c_1 \cdots c_n$: (Ramakrishna & Zobel, 1997)

- Hash of prefix $c_1 \cdots c_i$

$$h(c_i) := h(c_{i-1}) \oplus \left(c_i + h(c_{i-1}) \ll 6 + h(c_{i-1}) \gg 2\right)$$

- Fast, deterministic, quite uniform

Fingerprinting: Design choices

Hash of string $c_1 \cdots c_n$: (Ramakrishna & Zobel, 1997)

- Hash of prefix $c_1 \cdots c_i$

$$h(c_i) := h(c_{i-1}) \oplus \left(c_i + h(c_{i-1}) \ll 6 + h(c_{i-1}) \gg 2\right)$$

- Fast, deterministic, quite uniform

Substring length: 2–4 words

Fingerprinting: Design choices

Hash of string $c_1 \cdots c_n$: (Ramakrishna & Zobel, 1997)

- Hash of prefix $c_1 \cdots c_i$

$$h(c_i) := h(c_{i-1}) \oplus \left(c_i + h(c_{i-1}) \ll 6 + h(c_{i-1}) \gg 2\right)$$

- Fast, deterministic, quite uniform

Substring length: 2–4 words

Substring selection:

- All substrings (full fingerprinting)
- Random subsequences

Fingerprinting: Implementation

- Just store minutiae of references in hash table
- Per minutia: list of reference ids

```
typedef unordered_map<int, vector<int>> RefHashes;
```

Fingerprinting: Implementation

- Just store minutiae of references in hash table
- Per minutia: list of reference ids

```
typedef unordered_map<int, vector<int>> RefHashes;
```

- Computation: at the same time for every reference
- Normalisation: divide by number of query minutiae

Fingerprinting: Implementation

- Just store minutiae of references in hash table
- Per minutia: list of reference ids

```
typedef unordered_map<int, vector<int>> RefHashes;
```

- Computation: at the same time for every reference
- Normalisation: divide by number of query minutiae
- Substring selection can be exchanged:

```
typedef vector<string>::const_iterator WordIt;
typedef function<vector<WordIt>(const vector<string>&)> SubsequenceFunction;
```

Visualization

- Ranking: mark words from highest to smallest contribution until sum reaches constant $\times$ total sum
- Fingerprint: mark substrings with same minutia ($\rightsquigarrow$)
- Problem: overlapping substrings not directly representable as tree structure

Visualization

- Ranking: mark words from highest to smallest contribution until sum reaches constant $\times$ total sum
- Fingerprint: mark substrings with same minutia ($\rightsquigarrow$)
- Problem: overlapping substrings not directly representable as tree structure
- JavaScript:
  - Selection of similarity measure & compared reference
  - Summary table
  - Colour-coded values
  - Highlighting of matching marked words/substrings

Contents

Demo

Live demonstration

Contents

Problems

- Ill defined goal:
  - No standard benchmarks or test data
  - No right or wrong answers
  - What does a score of 0.0032134 mean?
  - Even for humans the problem is hard

Problems

- Ill defined goal:
  - No standard benchmarks or test data
  - No right or wrong answers
  - What does a score of 0.0032134 mean?
  - Even for humans the problem is hard
- Similarity of two texts depends on collection:
  - Do we even want that?
  - Are our tests therefore useless?
  - What is a good collection?

Problems

- Ill defined goal:
  - No standard benchmarks or test data
  - No right or wrong answers
  - What does a score of 0.0032134 mean?
  - Even for humans the problem is hard
- Similarity of two texts depends on collection:
  - Do we even want that?
  - Are our tests therefore useless?
  - What is a good collection?
- Unclear assumptions. When do which heuristics work?
  - Some heuristics from paper are nonsense
  - Some are nonsense only sometimes
  - Some seem to work better

Conclusions

- We do not contradict the paper:
  - The scores tend to be high for similar texts
  - What they said was broken actually was
  - Fingerprinting makes sense
  - Suffix arrays would have made at least as much sense
- To evaluate heuristics you need a clear picture of what they *should* do.

End

Thanks for your attention.
Do you have any questions?

Bibliography

Timothy C. Hoad and Justin Zobel. “Methods for identifying versioned and plagiarized documents”. In: *J. Am. Soc. Inf. Sci. Technol.* 54.3 (Feb. 2003), pp. 203–215. ISSN: 1532-2882. DOI: 10.1002/asi.10170.

M. V. Ramakrishna and Justin Zobel. “Performance in Practice of String Hashing Functions”. In: *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*. World Scientific Press, 1997, pp. 215–224. ISBN: 981-02-3107-5.