Algorithm to find articles with similar text

Asked 16 years, 1 month ago Modified 4 years, 3 months ago

Viewed 39k times

64

I have many articles in a database (with title,text), I'm looking for an algorithm to find the X most similar articles, something like Stack Overflow's "Related Questions" when you ask a question.

I tried googling for this but only found pages about other "similar text" issues, something like comparing every article with all the others and storing a similarity somewhere. SO does this in "real time" on text that I just typed.

How?

string algorithm text language-agnostic similarity

Share

Improve this question

Follow

edited Sep 3, 2018 at 16:35

Serge Rogatch

14.9k • 9 • 95 • 178

asked Oct 29, 2008 at 14:16

Osama Al-Maadeed

5,695 • 5 • 31 • 48

15 Answers Sorted by: Highest score (default)

34

Edit distance isn't a likely candidate, as it would be spelling/word-order dependent, and much more computationally expensive than Will is leading you to believe, considering the size and number of the documents you'd actually be interested in searching.

Something like Lucene is the way to go. You index all your documents, and then when you want to find documents similar to a given document, you turn your given document into a query, and search the index. Internally Lucene will be using tf-idf and an inverted index to make the whole process take an amount of time proportional to the number of documents that could possibly match, not the total number of documents in the collection.

Share Improve this answer Follow

answered Oct 30, 2008 at 23:36

Jay Kominek
8,773 ● 1 ● 36 ● 53

14

It depends upon your definition of similiar.

The edit-distance algorithm is the standard algorithm for (latin language) dictionary suggestions, and can work on whole texts. Two texts are similiar if they have basically the same words (eh letters) in the same order. So the following two book reviews would be fairly similiar:

1) "This is a great book"

2) "These are not great books"

(The number of letters to remove, insert, delete or alter to turn (2) into (1) is termed the 'edit distance'.)

To implement this you would want to visit every review programmatically. This is perhaps not as costly as it sounds, and if it is too costly you could do the comparisions as a background task and store the n-most-similiar in a database field itself.

Another approach is to understand something of the structure of (latin) languages. If you strip short (non-capitialised or quoted) words, and assign weights to words (or prefixes) that are common or unique, you can do a Bayesianesque comparision. The two following book reviews might be simiplied and found to be similiar:

3) "The french revolution was blah blah War and Peace blah blah France." -> France/French(2) Revolution(1) War(1) Peace(1) (note that a dictionary has been used to combine France and French)

4) "This book is blah blah a revolution in french cuisine." -> France(1) Revolution(1)

To implement this you would want to identify the 'keywords' in a review when it was created/updated, and to find similiar reviews use these keywords in the where-clause of a query (ideally 'full text' searching if the

database supports it), with perhaps a post-processing of the results-set for scoring the candidates found.

Books also have categories - are thrillers set in France similiar to historical studies of France, and so on? Meta-data beyond title and text might be useful for keeping results relevant.

Share Improve this answer

Follow

edited Oct 29, 2008 at 15:16

answered Oct 29, 2008 at 14:57

Will

75.6k ● 43 ● 175 ● 255

10

The tutorial at this link sounds like it may be what you need. It is easy to follow and works very well.

His algorithm rewards both common substrings and a common ordering of those substrings and so should pick out similar titles quite nicely.

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:21

alex77

532 ● 2 ● 5 ● 12

3

I suggest to index your articles using Apache Lucene, *a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search,*

especially cross-platform. Once indexed, you could easily find related articles.

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:21

Guido

47.6k ● 28 ● 124 ● 177

3

Seconding the Lucene suggestion for full-text, but note that java is not a requirement; a .NET port is available. Also see the main Lucene page for links to other projects, including Lucy, a C port.

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:31

b w

4,663 ● 4 ● 32 ● 37

3

One common algorithm used is the Self-Organizing Map. It is a type of neural network that will automatically categorize your articles. Then you can simply find the location that a current article is in the map and all articles near it are related. The important part of the algorithm is how you would vector quantize your input. There are several ways to do with with text. You can hash your document/title, you can count words and use that as an n dimensional vector, etc. Hope that helps, although I may have opened up a Pandora's box for you of an endless journey in AI.

Share Improve this answer

answered Oct 29, 2008 at 15:00

Follow

mempko
316 • 2 • 8

3

you can use the following

1. Minhash/LSH https://en.wikipedia.org/wiki/MinHash

(also see: http://infolab.stanford.edu/~ullman/mmds/book.pdf Minhash chapter), also see http://ann-benchmarks.com/ for state of the art

2. collaborative filtering if you have info of users interaction with articles (clicks/likes/views): https://en.wikipedia.org/wiki/Collaborative_filtering
3. word2vec or similar embeddings to compare articles in 'semantic' vector space: https://en.wikipedia.org/wiki/Word2vec
4. Latent semantic analysis: https://en.wikipedia.org/wiki/Latent_semantic_analysis
5. Use Bag-of-words and apply some distance measure, like Jaccard coefficient to compute set similarity https://en.wikipedia.org/wiki/Jaccard_index, https://en.wikipedia.org/wiki/Bag-of-words_model

Share Improve this answer

edited Aug 10, 2020 at 18:39

Follow

answered Mar 11, 2016 at 6:29

alex

1,907 ● 4 ● 22 ● 32

2

SO does the comparison only on the title, not on the body text of the question, so only on rather short strings.

You can use their algorithm (no idea what it looks like) on the article title and the keywords. If you have more cpu time to burn, also on the abstracts of your articles.

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:22

Treb

20.3k ● 8 ● 59 ● 88

2

Maybe what your looking for is something that does paraphrasing. I only have cursory knowledge of this, but paraphrasing is a natural language processing concept to determine if two passages of text actually *mean* the same thing - although the may use entirely different words.

Unfortunately I don't know of any tools that allow you to do this (although I'd be interested in finding one)

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:33

Vinnie

12.7k ● 15 ● 61 ● 80

another question on paraphrasing...
stackoverflow.com/questions/25332/... – Vinnie Oct 30, 2008 at 13:19

1

If you are looking for words that wound alike, you could convert to soundex and the the soundex words to match ... worked for me

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:50

spacemonkeys
1,749 ● 5 ● 16 ● 29

1

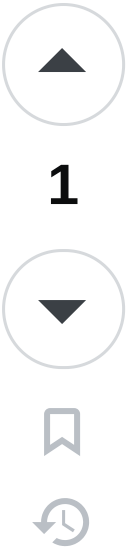
I tried some method but none works well.One may get a relatively satified result like this: First: get a Google SimHash code for every paragraph of all text and store it in databse. Second: Index for the SimHash code. Third: process your text to be compared as above,get a SimHash code and search all the text by SimHash index which apart form a Hamming distance like 5-10. Then compare simility with term vector. This may works for big data.

Share Improve this answer

Follow

answered Jul 22, 2013 at 6:47

Luna_one
31 ● 3



Given a sample text, this program lists the repository texts sorted by similarity: simple implementation of bag of words in C++. The algorithm is linear in the total length of the sample text and the repository texts. Plus the program is multi-threaded to process repository texts in parallel.

Here is the core algorithm:

```
class Statistics {
  std::unordered_map<std::string, int64_t>
_counts;
  int64_t _totWords;

  void process(std::string& token);
public:
  explicit Statistics(const std::string& text);

  double Dist(const Statistics& fellow) const;

  bool IsEmpty() const { return _totWords == 0; }
};

namespace {
  const std::string gPunctStr = ".,;:!?";
  const std::unordered_set<char>
gPunctSet(gPunctStr.begin(), gPunctStr.end());
}

Statistics::Statistics(const std::string& text) {
  std::string lastToken;
  for (size_t i = 0; i < text.size(); i++) {
    int ch = static_cast<uint8_t>(text[i]);
    if (!isspace(ch)) {
      lastToken.push_back(tolower(ch));
      continue;
    }
    process(lastToken);
  }
```

```
  process(lastToken);
}

void Statistics::process(std::string& token) {
  do {
    if (token.size() == 0) {
      break;
```

Share Improve this answer

Follow

answered Sep 3, 2018 at 15:51

Serge Rogatch

14.9k ● 9 ● 95 ● 178

1

The link in @alex77's answer points to an the Sorensen-Dice Coefficient which was independently discovered by the author of that article - the article is very well written and well worth reading.

I have ended up using this coefficient for my own needs. However, the original coefficient can yield erroneous results when dealing with

- three letter word pairs which contain one misspelling, e.g. `[and,amd]` and
- three letter word pairs which are anagrams e.g. `[and,dan]`

In the first case Dice erroneously reports a coefficient of zero whilst in the second case the coefficient turns up as 0.5 which is misleadingly high.

An improvement has been suggested which in its essence consists of taking the first and the last character of the word and creating an additional bigram.

In my view the improvement is only really required for 3 letter words - in longer words the other bigrams have a buffering effect that covers up the problem. My code that implements this improvement is given below.

```
function wordPairCount(word)
{
 var i,rslt = [],len = word.length - 1;
 for(i=0;i < len;i++) rslt.push(word.substr(i,2));
 if (2 == len) rslt.push(word[0] + word[len]);
 return rslt;
}

function pairCount(arr)
{
 var i,rslt = [];
 arr = arr.toLowerCase().split(' ');
 for(i=0;i < arr.length;i++) rslt = rslt.concat(word
 return rslt;
}

function commonCount(a,b)
{
 var t;
 if (b.length > a.length) t = b, b = a, a = t;
 t = a.filter(function (e){return b.indexOf(e) > -1;
 return t.length;
}

function myDice(a,b)
{
 var bigrams = [],
 aPairs = pairCount(a),
 bPairs = pairCount(b);
 debugger;
 var isct = commonCount(aPairs,bPairs);
 return 2*commonCount(aPairs,bPairs)/(aPairs.length

$('#rslt1').text(myDice('WEB Applications','PHP Web
$('#rslt2').text(myDice('And','Dan'));
```

```
$('#rslt3').text(myDice('and','aMd'));
$('#rslt4').text(myDice('abracadabra','abracabadra')
```

```
*{font-family:arial;}
table
{
 width:80%;
 margin:auto;
 border:1px solid silver;
}

thead > tr > td
{
 font-weight:bold;
 text-align:center;
 background-color:aqua;
}
```

```
<script src="https://ajax.googleapis.com/ajax/libs/j
</script>
<table>
<thead>
<tr>
<td>Phrase 1</td>
<td>Phrase 2</td>
<td>Dice</td>
</tr>
<thead>
<tbody>
<tr>
<td>WEB Applications</td>
<td>PHP Web Application</td>
<td id='rslt1'></td>
</tr>
<tr>
<td>And</td>
<td>Dan</td>
<td id='rslt2'></td>
</tr>
<tr>
<td>and</td>
```

```
<td>aMd</td>
<td id='rslt3'></td>
</tr>
<tr>
<td>abracadabra</td>
<td>abracabadra</td>
<td id='rslt4'></td>
</tr>
</tbody>
</table>
```

Run code snippet Expand snippet

Note the deliberate misspelling in the last example: abraca**dabra** vs abraca**badra**. Even though no extra bigram correction is applied the coefficient reported is 0.9. With the correction it would have been 0.91.

Share Improve this answer

Follow

edited Aug 25, 2020 at 18:29

halfer

20.4k ● 19 ● 108 ● 200

answered Feb 27, 2017 at 9:44

DroidOS

8,850 ● 17 ● 113 ● 187

0

You can use SQL Server Full-text index to get the smart comparison, I believe that SO is using an ajax call, that does a query to return the similar questions.

What technologies are you using?

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:19

Mitchel Sellers

63.1k ● 15 ● 114 ● 174

0

The simplest and fastest way to compare similarity among abstracts is probably by utilizing the set concept. First convert abstract texts into set of words. Then check how much each set overlaps. Python's set feature comes very hand performing this task. You would be surprised to see how well this method compares to those "similar/related papers" options out there provided by GScholar, ADS, WOS or Scopus.

Share Improve this answer

Follow

answered Apr 18, 2015 at 20:04

Gökhan Sever

8,452 ● 14 ● 38 ● 38