C# Finding relevant document snippets for search result display

Asked 11 years, 4 months ago Active 1 year, 2 months ago Viewed 4k times

10

6

In developing search for a site I am building, I decided to go the cheap and quick way and use Microsoft Sql Server's Full Text Search engine instead of something more robust like Lucene.Net.

One of the features I would like to have, though, is google-esque relevant document snippets. I quickly found determining "relevant" snippets is more difficult than I realized.

I want to choose snippets based on search term density in the found text. So, essentially, I need to find the most search term dense passage in the text. Where a passage is some arbitrary number of characters (say 200 -- but it really doesn't matter).

My first thought is to use .IndexOf() in a loop and build an array of term distances (subtract the index of the found term from the previously found term), then ... what? Add up any two, any three, any four, any five, sequential array elements and use the one with the smallest sum (hence, the smallest distance between search terms).

That seems messy.

Is there an established, better, or more obvious way to do this than what I have come up with?

c# algorithm search relevance significance

edited Oct 20 '12 at 17:58

hippietrail

12.3k 12 80 117

asked Nov 11 '08 at 20:10

CleverPatrick

7,850 3 51 75

Sigh...another 150 points wasted... – Michael Haren Jul 27 '10 at 20:44

1 For anybody interested in this question, there is a much newer language-agnostic question that has a higher-rated answer than anything on this question: **Given a document, select a relevant snippet** – hippietrail Oct 20 '12 at 18:02

8 Answers

6

Although it is implemented in Java, you can see one approach for that problem here: http://rcrezende.blogspot.com/2010/08/smallest-relevant-text-snippet-for.html

answered Aug 11 '10 at 21:28

Rodes

124 1 4

Good overview of the different algorithms. – CleverPatrick Aug 17 '10 at 18:58

4

I know this thread is way old, but I gave this a try last week and it was a pain in the back side. This is far from perfect, but this is what I came up with.

The snippet generator:

```
public static string SelectKeywordSnippets(string StringToSnip, string[] Keywords, int
SnippetLength)
    {
        string snippedString = "";
        List<int> keywordLocations = new List<int>();

        //Get the locations of all keywords
        for (int i = 0; i < Keywords.Count(); i++)
            keywordLocations.AddRange(SharedTools.IndexOfAll(StringToSnip, Keywords[i],
StringComparison.CurrentCultureIgnoreCase));

        //Sort locations
        keywordLocations.Sort();

        //Remove locations which are closer to each other than the SnippetLength
        if (keywordLocations.Count > 1)
        {
            bool found = true;
            while (found)
            {
                found = false;
                for (int i = keywordLocations.Count - 1; i > 0; i--)
                    if (keywordLocations[i] - keywordLocations[i - 1] < SnippetLength /
2)
                    {
                        keywordLocations[i - 1] = (keywordLocations[i] +
keywordLocations[i - 1]) / 2;

                        keywordLocations.RemoveAt(i);

                        found = true;
                    }
            }
        }

        //Make the snippets
        if (keywordLocations.Count > 0 && keywordLocations[0] - SnippetLength / 2 > 0)
            snippedString = "... ";
        foreach (int i in keywordLocations)
        {
            int stringStart = Math.Max(0, i - SnippetLength / 2);
            int stringEnd = Math.Min(i + SnippetLength / 2, StringToSnip.Length);
            int stringLength = Math.Min(stringEnd - stringStart, StringToSnip.Length -
stringStart);
            snippedString += StringToSnip.Substring(stringStart, stringLength);
            if (stringEnd < StringToSnip.Length) snippedString += " ... ";
            if (snippedString.Length > 200) break;
        }

        return snippedString;

    }
```

The function which will find the index of all keywords in the sample text

```
 private static List<int> IndexOfAll(string haystack, string needle, StringComparison
Comparison)
    {
        int pos;
        int offset = 0;
        int length = needle.Length;
        List<int> positions = new List<int>();
        while ((pos = haystack.IndexOf(needle, offset, Comparison)) != -1)
        {
            positions.Add(pos);
            offset = pos + length;
        }
        return positions;
    }
```

It's a bit clumsy in its execution. The way it works is by finding the position of all keywords in the string. Then checking that no keywords are closer to each other than the desired snippet length, so that snippets won't overlap (that's where it's a bit iffy...). And then grabs substrings of the desired length centered around the position of the keywords and stitches the whole thing together.

I know this is years late, but posting just in case it might help somebody coming across this question.

answered Jun 24 '11 at 1:46

yu_ominae

2,809 5 33 68

I found this approach very helpful with my Sitecore 7 search implementation. Thank you. – Coding101 Aug 8 '13 at 15:53

2

```
public class Highlighter
{
    private class Packet
    {
        public string Sentence;
        public double Density;
        public int Offset;
    }

    public static string FindSnippet(string text, string query, int maxLength)
    {
        if (maxLength < 0)
        {
            throw new ArgumentException("maxLength");
        }
        var words = query.Split(' ').Where(w =>
!string.IsNullOrWhiteSpace(w)).Select(word => word.ToLower()).ToLookup(s => s);
        var sentences = text.Split('.');
        var i = 0;
        var packets = sentences.Select(sentence => new Packet
        {
            Sentence = sentence,
            Density = ComputeDensity(words, sentence),
            Offset = i++
        }).OrderByDescending(packet => packet.Density);
        var list = new SortedList<int, string>();
```

```
                {
                    break;
                }
                string sentence = packet.Sentence;
                list.Add(packet.Offset, sentence.Substring(0, Math.Min(sentence.Length,
maxLength - length)));
                length += packet.Sentence.Length;
            }
            var sb = new List<string>();
            int previous = -1;
            foreach (var item in list)
            {
                var offset = item.Key;
                var sentence = item.Value;
                if (previous != -1 && offset - previous != 1)
                {
                    sb.Add(".");
                }
                previous = offset;
                sb.Add(Highlight(sentence, words));
            }
            return String.Join(".", sb);
        }

        private static string Highlight(string sentence, ILookup<string, string> words)
        {
            var sb = new List<string>();
            var ff = true;
            foreach (var word in sentence.Split(' '))
            {
                var token = word.ToLower();
                if (ff && words.Contains(token))
                {
                    sb.Add("[[HIGHLIGHT]]");
                    ff = !ff;
                }
                if (!ff && !string.IsNullOrWhiteSpace(token) && !words.Contains(token))
                {
                    sb.Add("[[ENDHIGHLIGHT]]");
                    ff = !ff;
                }
                sb.Add(word);
            }
            if (!ff)
            {
                sb.Add("[[ENDHIGHLIGHT]]");
            }
            return String.Join(" ", sb);
        }

        private static double ComputeDensity(ILookup<string, string> words, string sentence)
        {
            if (string.IsNullOrEmpty(sentence) || words.Count == 0)
            {
                return 0;
            }
            int numerator = 0;
            int denominator = 0;
            foreach(var word in sentence.Split(' ').Select(w => w.ToLower()))
            {
                if (words.Contains(word))
                {
                    numerator++;
                }
                denominator++;
            }
```

```
            }
            else
            {
                return 0;
            }
        }
    }
```

Example:

> highlight "Optic flow is defined as the change of structured light in the image, e.g. on the retina or the camera's sensor, due to a relative motion between the eyeball or camera and the scene. Further definitions from the literature highlight different properties of optic flow" "optic flow"

Output:

[[HIGHLIGHT]] Optic flow [[ENDHIGHLIGHT]] is defined as the change of structured light in the image, e... Further definitions from the literature highlight diff erent properties of [[HIGHLIGHT]] optic flow [[ENDHIGHLIGHT]]

answered Sep 14 '13 at 0:33

morpheus
13.4k 15 64 124

1

Well, here's the hacked together version I made using the algorithm I described above. I don't think it is all that great. It uses three (count em, three!) loops an array and two lists. But, well, it is better than nothing. I also hardcoded the maximum length instead of turning it into a parameter.

```
private static string FindRelevantSnippets(string infoText, string[] searchTerms)
    {
        List<int> termLocations = new List<int>();
        foreach (string term in searchTerms)
        {
            int termStart = infoText.IndexOf(term);
            while (termStart > 0)
            {
                termLocations.Add(termStart);
                termStart = infoText.IndexOf(term, termStart + 1);
            }
        }

        if (termLocations.Count == 0)
        {
            if (infoText.Length > 250)
                return infoText.Substring(0, 250);
            else
                return infoText;
        }

        termLocations.Sort();

        List<int> termDistances = new List<int>();
        for (int i = 0; i < termLocations.Count; i++)
        {
```

```
                continue;
            }
            termDistances.Add(termLocations[i] - termLocations[i - 1]);
        }

        int smallestSum = int.MaxValue;
        int smallestSumIndex = 0;
        for (int i = 0; i < termDistances.Count; i++)
        {
            int sum = termDistances.Skip(i).Take(5).Sum();
            if (sum < smallestSum)
            {
                smallestSum = sum;
                smallestSumIndex = i;
            }
        }
        int start = Math.Max(termLocations[smallestSumIndex] - 128, 0);
        int len = Math.Min(smallestSum, infoText.Length - start);
        len = Math.Min(len, 250);
        return infoText.Substring(start, len);
    }
```

Some improvements I could think of would be to return multiple "snippets" with a shorter length that add up to the longer length -- this way multiple parts of the document can be sampled.

answered Nov 12 '08 at 14:25

CleverPatrick
7,850 3 51 75

Note that in practice you have to tokenize the input string and compare the tokens - otherwise your users may start finding SEX in Sussex and Essex:-) (A problem with some web filtering programs!) – MZB Jul 27 '10 at 19:39

1

This is a nice problem :)

I think I'd create an index vector: For each word, create an entry 1 if search term or otherwise 0. Then find the i such that sum(indexvector[i:i+maxlength]) is maximized.

This can actually be done rather efficiently. Start with the number of searchterms in the first maxlength words. then, as you move on, decrease your counter if indexvector[i]=1 (i.e. your about to lose that search term as you increase i) and increase it if indexvector[i+maxlength+1]=1. As you go, keep track of the i with the highest counter value.

Once you got your favourite i, you can still do finetuning like see if you can reduce the actual size without compromising your counter, e.g. in order to find sentence boundaries or whatever. Or like picking the right i of a number of is with equivalent counter values.

Not sure if this is a better approach than yours - it's a different one.

You might also want to check out this paper on the topic, which comes with yet-another baseline: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.4357&rep=rep1&type=pdf

answered Jul 27 '10 at 17:41

1

I took another approach, perhaps it will help someone...

First it searches if it word appears in my case with IgnoreCase (you change this of course yourself). Then I create a list of Regex matches on each separators and search for the first occurrence of the word (allowing partial case insensitive matches). From that index, I get the 10 matches in front and behind the word, which makes the snippet.

```
public static string GetSnippet(string text, string word)
{
    if (text.IndexOf(word, StringComparison.InvariantCultureIgnoreCase) == -1)
    {
        return "";
    }

    var matches = new Regex(@"\b(\S+)\s?", RegexOptions.Singleline |
RegexOptions.Compiled).Matches(text);

    var p = -1;
    for (var i = 0; i < matches.Count; i++)
    {
        if (matches[i].Value.IndexOf(word, StringComparison.InvariantCultureIgnoreCase)
!= -1)
        {
            p = i;
            break;
        }
    }

    if (p == -1) return "";
    var snippet = "";
    for (var x = Math.Max(p - 10, 0); x < p + 10; x++)
    {
        snippet += matches[x].Value + " ";
    }
    return snippet;
}
```

edited Dec 14 '18 at 1:12

Banghua Zhao
1,378 1 9 21

answered Dec 13 '18 at 23:46

Peter Rakké
11 2

0

If you use CONTAINSTABLE you will get a RANK back , this is in essence a density value - higher the RANK value, the higher the density. This way, you just run a query to get the results you want and dont have to result to massaging the data when its returned.

answered Dec 10 '08 at 9:43

Coolcoder
3,875 5 24 32

This is for displaying the results of the search. They are displayed in Rank order. But in order to display them, I need a google-esque snippet of relevant text from the document -- not the whole document. – CleverPatrick Dec 12 '08 at 20:30

Wrote a function to do this just now. You want to pass in:

Document text
This is the full text of the document you're taking a snippet from. Most likely you will want to strip out any BBCode/HTML from this document.

Original query
The string the user entered as their search

Snippet length
Length of the snippet you wish to display.

Return Value:

Start index of the document text to take the snippet from. To get the snippet simply do `documentText.Substring(returnValue, snippetLength)`. This has the advantage that you know if the snippet is take from the start/end/middle so you can add some decoration like `...` if you wish at the snippet start/end.

Performance

A `resolution` set to `1` *will find the best snippet* but moves the window along 1 char at a time. Set this value higher to speed up execution.

Tweaks

You can work out `score` however you want. In this example I've done `Math.pow(wordLength, 2)` to favour longer words.

```
private static int GetSnippetStartPoint(string documentText, string originalQuery, int
snippetLength)
{
    // Normalise document text
    documentText = documentText.Trim();
    if (string.IsNullOrWhiteSpace(documentText)) return 0;

    // Return 0 if entire doc fits in snippet
    if (documentText.Length <= snippetLength) return 0;

    // Break query down into words
    var wordsInQuery = new HashSet<string>();
    {
        var queryWords = originalQuery.Split(' ');
        foreach (var word in queryWords)
        {
            var normalisedWord = word.Trim().ToLower();
            if (string.IsNullOrWhiteSpace(normalisedWord)) continue;
            if (wordsInQuery.Contains(normalisedWord)) continue;
            wordsInQuery.Add(normalisedWord);
        }
    }

    // Create moving window to get maximum trues
    var windowStart = 0;
    double maxScore = 0;
    var maxWindowStart = 0;

    // Higher number less accurate but faster
    const int resolution = 5;
```

```
        // Get score of this chunk
        // This isn't perfect, as window moves in steps of resolution first and last
words will be partial.
        // Could probably be improved to iterate words and not characters.
        var words = text.Split(' ').Select(c => c.Trim().ToLower());
        double score = 0;
        foreach (var word in words)
        {
            if (wordsInQuery.Contains(word))
            {
                // The longer the word, the more important.
                // Can simply replace with score += 1 for simpler model.
                score += Math.Pow(word.Length, 2);
            }
        }
        if (score > maxScore)
        {
            maxScore = score;
            maxWindowStart = windowStart;
        }

        // Setup next iteration
        windowStart += resolution;

        // Window end passed document end
        if (windowStart + snippetLength >= documentText.Length)
        {
            break;
        }
    }

    return maxWindowStart;
}
```

Lots more you can add to this, for example instead of comparing exact words perhaps you might want to try comparing the `SOUNDEX` where you weight soundex matches less than exact matches.

edited Aug 14 '17 at 16:52

answered Aug 14 '17 at 16:32

Tom Gullen

51k 74 255 419