The Porter Stemming Algorithm

This page was completely revised Jan 2006. The earlier edition is here.

(It has also been translated into Serbo-Croat by Jovana Milutinovich from Web Geeks Resources. Thank you, Jovana!)

This is the 'official' home page for distribution of the Porter Stemming Algorithm, written and maintained by its author, Martin Porter.

The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems.

History

The original stemming algorithm paper was written in 1979 in the Computer Laboratory, Cambridge (England), as part of a larger IR project, and appeared as Chapter 6 of the final project report,

> C.J. van Rijsbergen, S.E. Robertson and M.F. Porter, 1980. *New models in probabilistic information retrieval*. London: British Library. (British Library Research and Development Report, no. 5587).

With van Rijsbergen's encouragement, it was also published in,

> M.F. Porter, 1980, An algorithm for suffix stripping, *Program*, **14**(3) pp 130–137.

And since then it has been reprinted in

> Karen Sparck Jones and Peter Willet, 1997, *Readings in Information Retrieval*, San Francisco: Morgan Kaufmann, ISBN 1-55860-454-4.

The original stemmer was written in BCPL, a language once popular, but now defunct. For the first few years after 1980 it was distributed in its BCPL form, via the medium of punched paper tape. Versions in other languages soon began to appear, and by 1999 it was being widely used, quoted and adapted. Unfortunately there were numerous variations in functionality among these versions, and this web page was set up primarily to 'put the record straight' and establish a definitive version for distribution.

Encodings

The ANSI C version that heads the table below is exactly equivalent to the original BCPL version. The BCPL version did, however, differ in three minor points from the published algorithm and these are clearly marked in the downloadable ANSI C version. They are discussed further below.

This ANSI C version may be regarded as definitive, in that it now acts as a better definition of the algorithm than the original published paper.

Over the years, I have received many encoding from other workers, and they are also presented below. I have a reasonable confidence that all these versions are correctly encoded.

language	*author*	*affiliation*	*received*	*notes*
ANSI C	me			
ANSI C thread safe	me			
java	me			
Perl	me			
Perl	Daniel van Balen		Oct 1999	slightly faster?
python	Vivake Gupta		Jan 2001	
Csharp	André Hazelwood	The Official Web Guide	Sep 2001	
Csharp .NET compliant	Leif Azzopardi	Univerity of Paisley, Scotland	Nov 2002	
Common Lisp	Steven M. Haflich	Franz Inc	Mar 2002	
Ruby	Ray Pereda	www.raypereda.com	Jan 2003	github link
Visual Basic VB6	Navonil Mustafee	Brunel University	Apr 2003	
Delphi	Jo Rabin		Apr 2004	
Javascript	'Andargor'	www.andargor.com	Jul 2004	substantial revisions by Christopher McKenzie
Visual Basic	Christos Attikos	University of Piraeus, Greece	Jan 2005	

VB7; .NET compliant				
php	Richard Heyes	www.phpguru.org	Feb 2005	
Prolog	Philip Brooks	University of Georgia	Oct 2005	
Haskell	Dmitry Antonyuk		Nov 2005	
T-SQL	Keith Lubell	www.atelierdevitraux.com	May 2006	
matlab	Juan Carlos Lopez	California Pacific Medical Center Research Institute	Sep 2006	
Tcl	Aris Theodorakos	NCSR Demokritos	Nov 2006	
D	Daniel Truemper	Humboldt-Universitaet zu Berlin	May 2007	
erlang (1) erlang (2)	Alden Dima	National Institute of Standards and Technology, Gaithersburg, MD USA	Sep 2007	
REBOL	Dale K Brearcliffe		Apr 2009	
Scala	Ken Faulkner		May 2009	
sas	Antoine St-Pierre	Business Researchers, Inc	Apr 2010	
plugin vim script	Mitchell Bowden		May 2010	github link
node.js	Jed Parsons	jedparsons.com	May 2011	github link
Google Go	Alex Gonopolskiy		Oct 2011	github link
awk	Gregory Grefenstette	3ds.com/exalead	Jul 2012	
clojure	Yushi Wang		Mar 2013	bitbucket link
Rust	Do Nhat Minh	Nanyang Technological University	Aug 2013	github link
vala	Serge Hulne		Sep 2013	
MySQL	John Carty	Enlighten Jobs	Jan 2015	github link

All these encodings of the algorithm can be used free of charge for any purpose. Questions about the algorithms should be directed to their authors, and not to

Martin Porter (except when he is the author).

To test the programs out, here is a **sample vocabulary** (0.19 megabytes), and the corresponding **output**.

Email **any comments, suggestions, queries**

Points of difference from the published algorithm

There is an extra rule in Step 2,

(m>0) logi → log

So *archaeology* is equated with *archaeological* etc.

The Step 2 rule

(m>0) abli → able

is replaced by

(m>0) bli → ble

So *possibly* is equated with *possible* etc.

The algorithm leaves alone strings of length 1 or 2. In any case a string of length 1 will be unchanged if passed through the algorithm, but strings of length 2 might lose a final *s*, so *as* goes to *a* and *is* to *i*.

These differences may have been present in the program from which the published algorithm derived. But at such a great distance from the original publication it is now difficult to say.

It must be emphasised that these differences are very small indeed compared to the variations that have been observed in other encodings of the algorithm.

Status

The Porter stemmer should be regarded as 'frozen', that is, strictly defined, and not amenable to further modification. As a stemmer, it is slightly inferior to the Snowball *English* or *Porter2* stemmer, which derives from it, and which is subjected to occasional improvements. For practical work, therefore, the new Snowball stemmer is recommended. The Porter stemmer is appropriate to IR research work involving stemming where the experiments need to be exactly repeatable.

Common errors

Historically, the following shortcomings have been found in other encodings of the stemming algorithm.

The algorithm clearly explains that when a set of rules of the type

(condition)S1 → S2

are presented together, only one rule is applied, the one with the longest matching suffix *S1* for the given word. This is true whether the rule succeeds or fails (i.e. whether or not *S2* replaces *S1*). Despite this, the rules are sometimes simply applied in turn until either one of them succeeds or the list runs out.

This leads to small errors in various places, for example in the Step 4 rules

(m>1)ement →
(m>1)ment →
(m>1)ent →

to remove final *ement*, *ment* and *ent*.

Properly, *argument* stems to *argument*. The longest matching suffix is *-ment*. Then stem *argu-* has measure *m* equal to 1 and so *-ment* will not be removed. End of Step 4. But if the three rules are applied in turn, then for suffix *-ent* the stem *argum-* has measure *m* equal to 2, and *-ent* gets removed.

The more delicate rules are liable to misinterpretation. (Perhaps greater care was required in explaining them.) So

*((m>1) and (*s or *t))ion*

is taken to mean

(m>1)(s or t)ion

The former means that taking off *-ion* leaves a stem with measure greater than 1 ending *-s* or *-t*; the latter means that taking off *-sion* or *-tion* leaves a stem of measure greater than 1. A similar confusion tends to arise in interpreting rule 5*b*, to reduce final double *L* to single *L*.

Occasionally cruder errors have been seen. For example the test for *Y* being consonant or vowel set up the wrong way round.

It is interesting that although the published paper explains how to do the tests on the strings *S1* by a program switch on the last or last but one letter, many encodings fail to use this technique, making them much slower than they need be.

FAQs (frequently asked questions)

#1. What is the licensing arrangement for this software?

This question has become very popular recently (the period 2008–2009), despite the clear statment above that "all these encodings of the algorithm can be used free of charge for any purpose." The problem I think is that intellectual property has become such a major issue that some more formal statement is expected. So to restate it:

The software is completely free for any purpose, unless notes at the head of the program text indicates otherwise (which is rare). In any case, the notes about licensing are never more restrictive than the BSD License.

In every case where the software is not written by me (Martin Porter), this licensing arrangement has been endorsed by the contributor, and it is therefore unnecessary to ask the contributor again to confirm it.

I have not asked any contributors (or their employers, if they have them) for proofs that they have the right to distribute their software in this way.

(For anyone taking software from the Snowball website, the position is similar but simpler. There, all the software is issued under the BSD License, and for contributions not written by Martin Porter and Richard Boulton, we have again not asked the authors, or the authors' employers, for proofs that they have such distribution rights.)

#2. Why is the stemmer not producing proper words?

It is often taken to be a crude error that a stemming algorithm does not leave a real word after removing the stem. But the purpose of stemming is to bring variant forms of a word together, not to map a word onto its 'paradigm' form.

And connected with this,

#3. Why are there errors?

The question normally comes in the form, why should word *X* be stemmed to *x1*, when one would have expected it to be stemmed to *x2*? It is important to remember that the stemming algorithm cannot achieve perfection. On balance it will (or may) improve IR performance, but in individual cases it may sometimes make what are, or what seem to be, errors. Of course, this is a different matter from suggesting an additional rule that might be included in the stemmer to improve its performance.