

1. Problem

Although changes in tense and plurality can result in words spelled differently, the meaning of the words is still the same. When dealing with a large number of words it is sometimes useful to group similar words, such as fish, fishes, and fishing, together. This can be achieved through the use of stemming algorithms. The first widespread stemming algorithm was published in 1979 by Martin Porter, it is known as the Porter Stemmer.

The Porter Stemmer algorithm uses the morphological rules of the English language to remove common suffixes from words. This means that the stemmer is very language specific to English. The creator of the Porter Stemmer algorithm has also created a language called Snowball designed to make encoding stemmers in all languages easier and more uniform.¹ The Porter Stemmer algorithm was designed to aid in Information Retrieval problems so words with the same root or meaning would be treated as one word or group. This is useful when searching because the answers retrieved are slightly more flexible and potentially useful. It allows for connections to be made between words of similar meaning. The algorithm can also be used to make the set of things being searched smaller by grouping words together which could save time when searching a large database.

2. Use

The algorithm is useful for research done in the field of information retrieval² It is also used for more general language processing. The first time I encountered the algorithm was in the Natural Language Toolkit for python. The version of the algorithm used in the natural language toolkit was encoded in Python by Vivake Gupta. Personally, I used the stemmer to make lists of unique words in long word documents. It could also have been used to search those same documents for words if you wanted instances of the words meaning and not a specific tense.

3. history/development

The original paper was written in 1979 as part of a greater information retrieval project *New models in probabilistic information retrieval*.¹ When the algorithm was initially written it was coded in BCPL but since that language has fallen out of common usage the original author has implemented his algorithm with slight modifications from his original paper in C, java, and Perl. Many others have written the algorithm in several other languages including Csharp, Lisp, Ruby, Visual Basic, Delphi, Javascript, Prolog, Haskell, and matlab.³ The author says this about the current status of the algorithm "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."⁴ This means that no more new

¹ Martin Porter, Snowball: Quick Introduction, <<http://snowball.tartarus.org/texts/quickintro.html>>, April 17th, 2008

² Martin Porter, Porter Stemming Algorithm <<http://tartarus.org/~martin/PorterStemmer/>>, April 20, 2008

³ Martin Porter, Porter Stemming Algorithm <<http://tartarus.org/~martin/PorterStemmer/>>, April 20, 2008

⁴ Martin Porter, Porter Stemming Algorithm <<http://tartarus.org/~martin/PorterStemmer/>>, April 20, 2008

work is being done on the algorithm although it can still be used in its current form.

4.The algorithm

The algorithm takes common words endings in English morphology such as -s, and -ing, and removes them. This algorithm only deals with suffixes, there is no consideration or alteration of prefixes and infixes or other morphological changes present in the English language. The algorithm is mainly a list of rules arranged by priority. The rules use a series of definitions. These definitions, as based on the definitions in the original paper are that a consonant is any letter that is not A, E, I, O, or U, or if preceded by another consonant, Y. This accounts for the fact that Y is used as a consonant when preceded by a vowel as in the word JOY. All letters that are not consonants are vowels, and so will be represented by the letter v as all consonants are represented by the letter c. A list of multiple consonants or vowels is represented by one single uppercase letter. If the same pattern of Vs and Cs is repeated it can be denoted as (pattern of Vs and Cs){number of times it is repeated}. The word **COLLATERAL** would then map to **C V C V C V C V C** with **m= 4**, and **TREE** would map to **C V** with **m=0**.

These definitions are needed because some of the morphological rules of the English language only apply when they follow certain letter patterns. The rules present a condition S1 -> followed by the consequence S2, that is to say if the condition S1 is met then S1 will be replaced by S2, which can be null. Conditions may specify that the stem ends with a specific letter *letter, that the stem contains a vowel * v*, that the stem ends with a doubled consonant *d, and that the stem ends with the pattern cvc provided that the second c is not W,X, or Y *o. The conditions may have expressions with \and\, \or\, and \not as well so one condition may have many variables. One of the more common errors in the algorithm comes from the fact that because in a set of rules grouped together only the one with the longest matching condition for the word is obeyed. This leads to errors such as the fact that argument, which should stem to argument, stems to argum-.

There are five stemming steps in this algorithm. Step 1 has all of the rules relevant to plurals and past participles.

Step 1a

SSSES -> SS	finesses -> finess
IES -> I	cities -> citi
SS -> SS	caress -> caress
S ->	dogs -> dog

Step 1b

(m>0) EED -> EE	feed -> feed
	agreed -> agree
	conflated -> conflat
'feed' does not go to 'fee' because in the word feed m=0.	
(*v*) ED ->	plastered -> plaster
	bled -> bled
(*v*) ING ->	motoring -> motor
	sing -> sing

likewise 'bled' and 'sing' do not change because they do not have the pattern (letter)vowel(letter) before the end of the word.

If the second or third of the rules in Step 1b is successful, the following is done:

AT -> ATE	conflat(ed) -> conflate
BL -> BLE	troubl(ed) -> trouble
IZ -> IZE	siz(ed) -> size
(*d and not (*L or *S or *Z))	
-> single letter	
	hopp(ing) -> hop
	tann(ed) -> tan
	fall(ing) -> fall
	hiss(ing) -> hiss
	fizz(ed) -> fizz
(m=1 and *o) -> E	fail(ing) -> fail
	fil(ing) -> file

This is done to replace e's that are at the end of words with those common endings because the previous rule would leave the words incorrect otherwise. The second part of this rule is to correct words that have consonants doubled in their longer forms.

Step 1c

(*v*) Y -> I	happy -> happi
	city -> citi
	sky -> sky

Step 1c changes the Y ending of most words to an I so that the words will match up with roots created in step 1a.

Step 2 removes many of the longer endings in the English language and replaces them with their proper root ending.

Step 2

(m>0) ATIONAL -> ATE	relational -> relate
(m>0) TIONAL -> TION	conditional -> condition
	rational -> rational
(m>0) ENCI -> ENCE	valenci -> valence
(m>0) ANCI -> ANCE	hesitanci -> hesitance
(m>0) IZER -> IZE	digitizer -> digitize
(m>0) ABLI -> ABLE	conformabli -> conformable
(m>0) ALLI -> AL	radicalli -> radical
(m>0) ENTLI -> ENT	differentli -> different
(m>0) ELI -> E	vileli -> vile
(m>0) OUSLI -> OUS	analogousli -> analogous
(m>0) IZATION -> IZE	vietnamization -> vietnamize
(m>0) ATION -> ATE	predication -> predicate
(m>0) ATOR -> ATE	operator -> operate
(m>0) ALISM -> AL	feudalism -> feudal
(m>0) IVENESS -> IVE	decisiveness -> decisive
(m>0) FULNESS -> FUL	hopefulness -> hopeful
(m>0) OUSNESS -> OUS	callousness -> callous
(m>0) ALITI -> AL	formaliti -> formal
(m>0) IVITI -> IVE	sensitiviti -> sensitive
(m>0) BILITI -> BLE	sensibiliti -> sensible

This step of the algorithm is made quick and efficient by presenting the S1-strings in alphabetical order of their penultimate letter.

Step 3 does much the same thing as step 2 but it must come after step 2 for many of its s1-strings were contained in the larger s1-strings of step 2. What I say here also holds true for step 4.

Step 3

(m>0) ICATE ->	IC	triplicate	->	triplic
(m>0) ATIVE ->		formative	->	form
(m>0) ALIZE ->	AL	formalize	->	formal
(m>0) ICITI ->	IC	electriciti	->	electric
(m>0) ICAL ->	IC	electrical	->	electric
(m>0) FUL ->		hopeful	->	hope
(m>0) NESS ->		goodness	->	good

Step 4

(m>1) AL ->		revival	->	reviv
(m>1) ANCE ->		allowance	->	allow
(m>1) ENCE ->		inference	->	infer
(m>1) ER ->		airliner	->	airlin
(m>1) IC ->		gyroscopic	->	gyroscop
(m>1) ABLE ->		adjustable	->	adjust
(m>1) IBLE ->		defensible	->	defens
(m>1) ANT ->		irritant	->	irrit
(m>1) EMENT ->		replacement	->	replac
(m>1) MENT ->		adjustment	->	adjust
(m>1) ENT ->		dependent	->	depend
(m>1 and (*S or *T)) ION ->		adoption	->	adopt
(m>1) OU ->		homologou	->	homolog
(m>1) ISM ->		communism	->	commun
(m>1) ATE ->		activate	->	activ
(m>1) ITI ->		angulariti	->	angular
(m>1) OUS ->		homologous	->	homolog
(m>1) IVE ->		effective	->	effect
(m>1) IZE ->		bowdlerize	->	bowdler

After the first four steps all of the suffixes are removed, as in step 1b however, just because the suffixes are removed does not mean that the root words are correct yet. Step 5 is needed to clean up the root words so that no matter which step they took to get to the root, as many words as possible will be matched up correctly.

Step 5a

(m>1) E ->		probate	->	probat
		rate	->	rate
(m=1 and not *o) E ->		cease	->	ceas

Step 5b

(m > 1 and *d and *L) ->	single letter		
	controll	->	control
	roll	->	roll

The algorithm presented above was taken from the original paper.⁵ The programs written by Martin Porter, and many others, contain the following departures from the original algorithm as presented above. Two changes to Step 2, the addition of (m>0)logi -> log and the

⁵Martin Porter, *An Algorithm for Suffix Stripping*, <<http://tartarus.org/~martin/PorterStemmer/def.txt>>, (July 1980), April 20, 2008

alteration of (m>0) abli -> able to (m>0) bli->ble the algorithm also leaves alone strings of one and two so **AS** is no longer reduced to **A**.⁶

An example of the output of the algorithm, using Martin Porters C program which can be found at http://tartarus.org/~martin/PorterStemmer/c_thread_safe.txt demonstrates some strengths and weaknesses of the algorithm. The input words provided to the stemmer were **COMPUTE**, **COMPUTER**, **COMPUTERIZATION**, **COMPUTATION**, and **COMPUTATIONALLY**. The resulting stems were **COMPUT**, **COMPUT**, **COMPUTER**, **COMPUT**, and **COMPUTATION**. This means that **COMPUTE**, **COMPUTER**, and **COMPUTATION** were grouped together but the other words were not stemmed that far. This shows that while mostly words are stemmed correctly, the algorithm does not account for all the words that may have more than one morphological ending on them.

Although the many steps to this algorithm seem complex they are far from perfect. This algorithm will stem many words incorrectly, for example **WANDER** will be stemmed to **WAND**, however creating an English stemming algorithm that catches all the idiosyncrasies of the English language is impractical if not downright impossible. IF additional rules were created, they would not only make the run time of the algorithm longer, they would also create additional mistakes in implementation.

⁶Martin Porter, Porter Stemming Algorithm <<http://tartarus.org/~martin/PorterStemmer/>>, April 20, 2008