Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
العربية
Čeština
Deutsch
Español
Euskara
Français
한국어
Հայերեն
Bahasa Indonesia
Italiano
Português
Русский
Slovenščina
Suomi
Svenska
Українська
中文
Edit links

Stemming

From Wikipedia, the free encyclopedia

For the skiing technique, see Stem (skiing).

This article **needs attention from an expert on the subject**. Please add a *reason* or a *talk* parameter to this template to explain the issue with the article. Consider associating this request with a WikiProject. *(October 2010)*

Stemming is the term used in linguistic morphology and information retrieval to describe the process for reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem needs not to be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation.

Stemming programs are commonly referred to as stemming algorithms or stemmers.

Contents [hide]

Examples [edit]

A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stems", "stemmer", "stemming", "stemmed" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish". On the other hand, "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu" (illustrating the case where the stem is not itself a word or root) but "argument" and "arguments" reduce to the stem "argument".

History [edit]

The first published stemmer was written by Julie Beth Lovins in 1968.[1] This paper was remarkable for its early date and had great influence on later work in this area.

A later stemmer was written by Martin Porter and was published in the July 1980 issue of the journal *Program*. This stemmer was very widely used and became the de facto standard algorithm used for English stemming. Dr. Porter received the Tony Kent Strix award in 2000 for his work on stemming and information retrieval.

Many implementations of the Porter stemming algorithm were written and freely distributed; however, many of these implementations contained subtle flaws. As a result, these stemmers did not match their potential. To eliminate this source of error, Martin Porter released an official free-software implementation of the algorithm around the year 2000. He extended this work over the next few years by building Snowball, a framework for writing stemming algorithms, and implemented an improved English stemmer together with stemmers for several other languages.

Algorithms [edit]

There are several types of stemming algorithms which differ in respect to performance and accuracy and how certain stemming obstacles are overcome.

Lookup algorithms [edit]

By Lalit rastogi M.Techc C.S.E Kec Dwarahat

A simple stemmer looks up the inflected form in a lookup table. The advantages of this approach is that it is simple, fast, and easily handles exceptions. The disadvantages are that all inflected forms must be explicitly listed in the table: new or unfamiliar words are not handled, even if they are perfectly regular (e.g. iPads ~ iPad), and the table may be large. For languages with simple morphology, like English, table sizes are modest, but highly inflected languages like Turkish may have hundreds of potential inflected forms for each root.

A lookup approach may use preliminary part-of-speech tagging to avoid overstemming.[2]

The production technique [edit]

The lookup table used by a stemmer is generally produced semi-automatically. For example, if the word is "run", then the inverted algorithm might automatically generate the forms "running", "runs", "runned", and "runly". The last two forms are valid constructions, but they are unlikely.

Suffix-stripping algorithms [edit]

Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. Instead, a typically smaller list of "rules" is stored which provides a path for the algorithm, given an input word form, to find its root form. Some examples of the rules include:

- if the word ends in 'ed', remove the 'ed'
- if the word ends in 'ing', remove the 'ing'
- if the word ends in 'ly', remove the 'ly'

Suffix stripping approaches enjoy the benefit of being much simpler to maintain than brute force algorithms, assuming the maintainer is sufficiently knowledgeable in the challenges of linguistics and morphology and encoding suffix stripping rules. Suffix stripping algorithms are sometimes regarded as crude given the poor performance when dealing with exceptional relations (like 'ran' and 'run'). The solutions produced by suffix stripping algorithms are limited to those lexical categories which have well known suffixes with few exceptions. This, however, is a problem, as not all parts of speech have such a well formulated set of rules. Lemmatisation attempts to improve upon this challenge.

Prefix stripping may also be implemented. Of course, not all languages use prefixing or suffixing.

Additional algorithm criteria [edit]

Suffix stripping algorithms may differ in results for a variety of reasons. One such reason is whether the algorithm constrains whether the output word must be a real word in the given language. Some approaches do not require the word to actually exist in the language lexicon (the set of all words in the language). Alternatively, some suffix stripping approaches maintain a database (a large list) of all known morphological word roots that exist as real words. These approaches check the list for the existence of the term prior to making a decision. Typically, if the term does not exist, alternate action is taken. This alternate action may involve several other criteria. The non-existence of an output term may serve to cause the algorithm to try alternate suffix stripping rules.

It can be the case that two or more suffix stripping rules apply to the same input term, which creates an ambiguity as to which rule to apply. The algorithm may assign (by human hand or stochastically) a priority to one rule or another. Or the algorithm may reject one rule application because it results in a non-existent term whereas the other overlapping rule does not. For example, given the English term *friendlies*, the algorithm may identify the *ies* suffix and apply the appropriate rule and achieve the result of *friendl*. *friendl* is likely not found in the lexicon, and therefore the rule is rejected.

One improvement upon basic suffix stripping is the use of suffix substitution. Similar to a stripping rule, a substitution rule replaces a suffix with an alternate suffix. For example, there could exist a rule that replaces *ies* with *y*. How this affects the algorithm varies on the algorithm's design. To illustrate, the algorithm may identify that both the *ies* suffix stripping rule as well as the suffix substitution rule apply. Since the stripping rule results in a non-existent term in the lexicon, but the substitution rule does not, the substitution rule is applied instead. In this example, *friendlies* becomes *friendly* instead of *friendl*.

Diving further into the details, a common technique is to apply rules in a cyclical fashion (recursively, as computer scientists would say). After applying the suffix substitution rule in this example scenario, a second pass is made to identify matching rules on the term *friendly*, where the *ly* stripping rule is likely identified and accepted. In summary, *friendlies* becomes (via substitution) *friendly* which becomes (via stripping) *friend*.

This example also helps illustrate the difference between a rule-based approach and a brute force approach. In a brute force approach, the algorithm would search for *friendlies* in the set of hundreds of thousands of inflected word forms and ideally find the corresponding root form *friend*. In the rule-based approach, the three rules mentioned above would be applied in succession to converge on the same solution. Chances are that the rule-based approach would be slower, as lookup algorithms have a direct access to the solution, while rule-based should try several options, and combinations of them, and then choose which result seems to be the best.

Lemmatisation algorithms [edit]

A more complex approach to the problem of determining a stem of a word is lemmatisation. This process involves first determining the part of speech of a word, and applying different normalization rules for each part of speech. The part of speech is first detected prior to attempting to find the root since for some languages, the stemming rules change depending on a word's part of speech.

This approach is highly conditional upon obtaining the correct lexical category (part of speech). While there is overlap between the normalization rules for certain categories, identifying the wrong category or being unable to produce the right category limits the added benefit of this approach over suffix stripping algorithms. The basic idea is that, if the stemmer is able to grasp more information about the word being stemmed, then it can apply more accurate normalization rules (which unlike suffix stripping rules can also modify the stem).

Stochastic algorithms [edit]

Stochastic algorithms involve using probability to identify the root form of a word. Stochastic algorithms are trained (they "learn") on a table of root form to inflected form relations to develop a probabilistic model. This model is typically expressed in the form of complex linguistic rules, similar in nature to those in suffix stripping or lemmatisation. Stemming is performed by inputting an inflected form to the trained model and having the model produce the root form according to its internal ruleset, which again is similar to suffix stripping and lemmatisation, except that the decisions involved in applying the most appropriate rule, or whether or not to stem the word and just return the same word, or whether to apply two different rules sequentially, are applied on the grounds that the output word will have the highest probability of being correct (which is to say, the smallest probability of being incorrect, which is how it is typically measured).

Some lemmatisation algorithms are stochastic in that, given a word which may belong to multiple parts of speech, a probability is assigned to each possible part. This may take into account the surrounding words, called the context, or not. Context-free grammars do not take into account any additional information. In either case, after assigning the probabilities to each possible part of speech, the most likely part of speech is chosen, and from there the appropriate normalization rules are applied to the input word to produce the normalized (root) form.

n-gram analysis [edit]

Some stemming techniques use the n-gram context of a word to choose the correct stem for a word.[3]

Hybrid approaches [edit]

Hybrid approaches use two or more of the approaches described above in unison. A simple example is a suffix tree algorithm which first consults a lookup table using brute force. However, instead of trying to store the entire set of relations between words in a given language, the lookup table is kept small and is only used to store a minute amount of "frequent exceptions" like "ran => run". If the word is not in the exception list, apply suffix stripping or lemmatisation and output the result.

Affix stemmers [edit]

In linguistics, the term affix refers to either a prefix or a suffix. In addition to dealing with suffixes, several approaches also attempt to remove common prefixes. For example, given the word *indefinitely*, identify that the leading "in" is a prefix that can be removed. Many of the same approaches mentioned earlier apply, but go by the name **affix stripping**. A study of affix stemming for several European languages can be found here.[4]

Matching algorithms [edit]

Such algorithms use a stem database (for example a set of documents that contain stem words). These stems, as mentioned above, are not necessarily valid words themselves (but rather common sub-strings, as the "brows" in "browse" and in "browsing"). In order to stem a word the algorithm tries to match it with stems from the database, applying various constraints, such as on the relative length of the candidate stem within the word (so that, for example, the short prefix "be", which is the stem of such words as "be", "been" and "being", would not be considered as the stem of the word "beside").

Language challenges [edit]

While much of the early academic work in this area was focused on the English language (with significant use of the Porter Stemmer algorithm), many other languages have been investigated.[5][6][7][8][9]

Hebrew and Arabic are still considered difficult research languages for stemming. English stemmers are fairly trivial (with only occasional problems, such as "dries" being the third-person singular present form of the verb "dry", "axes" being the plural of "axe" as well as "axis"); but stemmers become harder to design as the morphology, orthography, and character encoding of the target language becomes more complex. For example, an Italian stemmer is more complex than an English one (because of a greater number of verb inflections), a Russian one is more complex (more noun declensions), a Hebrew one is even more complex (due to nonconcatenative morphology, a writing system without vowels, and the requirement of prefix stripping: Hebrew stems can be two, three or four characters, but not more), and so on.

Multilingual stemming [edit]

Multilingual stemming applies morphological rules of two or more languages simultaneously instead of rules for only a single language when interpreting a search query. Commercial systems using multilingual stemming exist[*citation needed*].

Error metrics [edit]

There are two error measurements in stemming algorithms, overstemming and understemming. Overstemming is an error where two separate inflected words are stemmed to the same root, but should not have been—a false positive. Understemming is an error where two separate inflected words should be stemmed to the same root, but are not—a false negative. Stemming algorithms attempt to minimize each type of error, although reducing one type can lead to increasing the other.

For example, the widely used Porter stemmer stems "universal", "university", and "universe" to "univers". This is a case of overstemming: though these three words are etymologically related, their modern meanings are in widely different domains, so treating them as synonyms in a search engine will likely reduce the relevance of the search results.

An example of understemming in the Porter stemmer is "alumnus" → "alumnu", "alumni" → "alumni", "alumna"/"alumnae" → "alumna". This English word keeps Latin morphology, and so these near-synonyms are not conflated.

Applications [edit]

Stemming is used as an approximate method for grouping words with a similar basic meaning together. For example, a text mentioning "daffodils" is probably closely related to a text mentioning "daffodil" (without the s). But in some cases, words with the same morphological stem have idiomatic meanings which are not closely related: a user searching for "marketing" will not be satisfied by most documents mentioning "markets" but not "marketing".

Information retrieval [edit]

Stemmers are common elements in query systems such as Web search engines. The effectiveness of stemming for English query systems were soon found to be rather limited, however, and this has led early information retrieval researchers to deem stemming irrelevant in general.[10] An alternative approach, based on searching for n-grams rather than stems, may be used instead. Also, recent research has shown greater benefits for retrieval in other languages.[11][12]

Domain Analysis [edit]

Stemming is used to determine domain vocabularies in domain analysis. [13]

Use in commercial products [edit]

Many commercial companies have been using stemming since at least the 1980s and have produced algorithmic and lexical stemmers in many languages.[14][15]

The Snowball stemmers have been compared with commercial lexical stemmers with varying results.[16][17]

Google search adopted word stemming in 2003.[18] Previously a search for "fish" would not have returned "fishing". Other software search algorithms vary in their use of word stemming. Programs that simply search for substrings obviously will find "fish" in "fishing" but when searching for "fishes" will not find occurrences of the word "fish".

See also [edit]

- Root (linguistics) - linguistic definition of the term "root"
- Stem (linguistics) - linguistic definition of the term "stem"
- Morphology (linguistics)
- Lemma (morphology) - linguistic definition
- Lemmatization
- Lexeme
- Inflection
- Derivation - stemming is a form of reverse derivation
- Natural language processing - stemming is generally regarded as a form of NLP
- Text mining - stemming algorithms play a major role in commercial NLP software
- Computational linguistics
- Snowball (programming language) - designed for creating stemming algorithms

v· t· e **Natural Language Processing** [show]

References [edit]

1. ^ Lovins, Julie Beth (1968). "Development of a Stemming Algorithm". *Mechanical Translation and Computational Linguistics* **11**: 22–31.
2. ^ Yatsko, V. A.; *Y-stemmer*
3. ^ McNamee, Paul (September 21–22, 2005). "Exploring New Languages with HAIRCUT at CLEF 2005" (PDF). *CEUR Workshop Proceedings* **1171**. Retrieved 3/6/15. Check date values in: |accessdate= (help)
4. ^ Jongejan, B.; and Dalianis, H.; *Automatic Training of Lemmatization Rules that Handle Morphological Changes in pre-, in- and Suffixes Alike*, in the *Proceeding of the ACL-2009, Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Singapore, August 2–7, 2009*, pp. 145-153 [1]
5. ^ Dolamic, Ljiljana; and Savoy, Jacques; *Stemming Approaches for East European Languages (CLEF 2007)*
6. ^ Savoy, Jacques; *Light Stemming Approaches for the French, Portuguese, German and Hungarian Languages*, ACM Symposium on Applied Computing, SAC 2006, ISBN 1-59593-108-2
7. ^ Popovič, Mirko; and Willett, Peter (1992); *The Effectiveness of Stemming for Natural-Language Access to Slovene Textual Data*, Journal of the American Society for Information Science, Volume 43, Issue 5 (June), pp. 384–390
8. ^ *Stemming in Hungarian at CLEF 2005*
9. ^ Viera, A. F. G. & Virgil, J. (2007); *Uma revisão dos algoritmos de radicalização em língua portuguesa*, Information Research, 12(3), paper 315
10. ^ Baeza-Yates, Ricardo; and Ribeiro-Neto, Berthier (1999); *Modern Information Retrieval*, ACM Press/Addison Wesley

11. ^ Kamps, Jaap; Monz, Christof; de Rijke, Maarten; and Sigurbjörnsson, Börkur (2004); *Language-Dependent and Language-Independent Approaches to Cross-Lingual Text Retrieval*, in Peters, C.; Gonzalo, J.; Braschler, M.; and Kluck, M. (eds.); *Comparative Evaluation of Multilingual Information Access Systems*, Springer Verlag, pp. 152–165
12. ^ Airio, Eija (2006); *Word Normalization and Decompounding in Mono- and Bilingual IR*, Information Retrieval **9**:249–271
13. ^ Frakes, W.; Prieto-Diaz, R.; & Fox, C. (1998); *DARE: Domain Analysis and Reuse Environment*, Annals of Software Engineering (5), pp. 125-141
14. ^ *Language Extension Packs*, dtSearch
15. ^ *Building Multilingual Solutions by using Sharepoint Products and Technologies*, Microsoft Technet
16. ^ CLEF 2003: Stephen Tomlinson compared the Snowball stemmers with the Hummingbird lexical stemming (lemmatization) system
17. ^ CLEF 2004: Stephen Tomlinson "Finnish, Portuguese and Russian Retrieval with Hummingbird SearchServer"
18. ^ *The Essentials of Google Search*, Web Search Help Center, Google Inc.

Further reading [edit]

- Dawson, J. L. (1974); *Suffix Removal for Word Conflation*, Bulletin of the Association for Literary and Linguistic Computing, 2(3): 33–46
- Frakes, W. B. (1984); *Term Conflation for Information Retrieval*, Cambridge University Press
- Frakes, W. B. & Fox, C. J. (2003); *Strength and Similarity of Affix Removal Stemming Algorithms*, SIGIR Forum, 37: 26–30
- Frakes, W. B. (1992); *Stemming algorithms, Information retrieval: data structures and algorithms*, Upper Saddle River, NJ: Prentice-Hall, Inc.
- Hafer, M. A. & Weiss, S. F. (1974); *Word segmentation by letter successor varieties*, Information Processing & Management 10 (11/12), 371–386
- Harman, D. (1991); *How Effective is Suffixing?*, Journal of the American Society for Information Science 42 (1), 7–15
- Hull, D. A. (1996); *Stemming Algorithms – A Case Study for Detailed Evaluation*, JASIS, 47(1): 70–84
- Hull, D. A. & Grefenstette, G. (1996); *A Detailed Analysis of English Stemming Algorithms*, Xerox Technical Report
- Kraaij, W. & Pohlmann, R. (1996); *Viewing Stemming as Recall Enhancement*, in Frei, H.-P.; Harman, D.; Schauble, P.; and Wilkinson, R. (eds.); *Proceedings of the 17th ACM SIGIR conference held at Zurich, August 18–22*, pp. 40–48
- Krovetz, R. (1993); *Viewing Morphology as an Inference Process*, in *Proceedings of ACM-SIGIR93*, pp. 191–203
- Lennon, M.; Pierce, D. S.; Tarry, B. D.; & Willett, P. (1981); *An Evaluation of some Conflation Algorithms for Information Retrieval*, Journal of Information Science, 3: 177–183
- Lovins, J. (1971); *Error Evaluation for Stemming Algorithms as Clustering Algorithms*, JASIS, 22: 28–40
- Lovins, J. B. (1968); *Development of a Stemming Algorithm*, Mechanical Translation and Computational Linguistics, 11, 22—31
- Jenkins, Marie-Claire; and Smith, Dan (2005); *Conservative Stemming for Search and Indexing*
- Paice, C. D. (1990); *Another Stemmer*, SIGIR Forum, 24: 56–61
- Paice, C. D. (1996) *Method for Evaluation of Stemming Algorithms based on Error Counting*, JASIS, 47(8): 632–649
- Popovič, Mirko; and Willett, Peter (1992); *The Effectiveness of Stemming for Natural-Language Access to Slovene Textual Data*, Journal of the American Society for Information Science, Volume 43, Issue 5 (June), pp. 384–390
- Porter, Martin F. (1980); *An Algorithm for Suffix Stripping*, Program, 14(3): 130–137
- Savoy, J. (1993); *Stemming of French Words Based on Grammatical Categories* Journal of the American Society for Information Science, 44(1), 1–9
- Ulmschneider, John E.; & Doszkocs, Tamas (1983); *A Practical Stemming Algorithm for Online Search Assistance*, Online Review, 7(4), 301–318
- Xu, J.; & Croft, W. B. (1998); *Corpus-Based Stemming Using Coocurrence of Word Variants*, ACM Transactions on Information Systems, 16(1), 61–81

External links [edit]

- Apache OpenNLP includes Porter and Snowball stemmers
- SMILE Stemmer - free online service, includes Porter and Paice/Husk' Lancaster stemmers (Java API)
- Themis - open source IR framework, includes Porter stemmer implementation (PostgreSQL, Java API)
- Snowball - free stemming algorithms for many languages, includes source code, including stemmers for five romance languages
- Snowball on C# - port of Snowball stemmers for C# (14 languages)
- Python bindings to Snowball API
- Ruby-Stemmer - Ruby extension to Snowball API
- PECL - PHP extension to the Snowball API
- Oleander Porter's algorithm - stemming library in C++ released under BSD
- Unofficial home page of the Lovins stemming algorithm - with source code in a couple of languages

- Official home page of the Porter stemming algorithm - including source code in several languages
- Official home page of the Lancaster stemming algorithm - Lancaster University, UK
- Official home page of the UEA-Lite Stemmer - University of East Anglia, UK
- Overview of stemming algorithms
- PTStemmer - A Java/Python/.Net stemming toolkit for the Portuguese language
- jsSnowball - open source JavaScript implementation of Snowball stemming algorithms for many languages
- Snowball Stemmer - implementation for Java
- hindi_stemmer - open source stemmer for Hindi
- czech_stemmer - open source stemmer for Czech
- Comparative Evaluation of Arabic Language Morphological Analysers and Stemmers
- Tamil Stemmer

This article is based on material taken from the Free On-line Dictionary of Computing prior to 1 November 2008 and incorporated under the "relicensing" terms of the GFDL, version 1.3 or later.

Categories: Linguistic morphology | Natural language processing | Tasks of natural language processing | Computational linguistics | Information retrieval techniques