MODUL TOPIK DALAM INFORMATION RETRIEVAL

(CMA 102)

MODUL PERTEMUAN 11

The Term Vocabulary and Postings Lists (Part 2)

DISUSUN OLEH

Dr. Fransiskus Adikara, S.Kom, MMSI

UNIVERSITAS ESA UNGGUL

2019

DETERMINING THE VOCABULARY OF TERMS (Part 2)

A. Kemampuan Akhir Yang Diharapkan

After reading this session, you will be able to answer the following questions:

1. Understanding of the basic unit of classical information retrieval systems: words and documents: What is a document, what is a term?
2. Tokenization: how to get from raw text to words (or tokens)
3. More complex indexes: skip pointers and phrases

B. Uraian dan Contoh

2.3. Normalization (equivalence classing of terms)

Having broken up our documents (and also our query) into tokens, the easy case is if tokens in the query just match tokens in the token list of the document. However, there are many cases when two character sequences are not quite the same but you would like a match to occur. For instance, if you search for *USA*, you might hope to also match documents containing *U.S.A*.

TOKEN NORMALIZATION

EQUIVALENCE CLASSES

Token normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens. The most standard way to normalize is to implicitly create *equivalence classes*, which are normally named after one member of the set. For instance, if the tokens *anti-discriminatory* and *antidiscriminatory* are both mapped onto the term antidiscriminatory, in both the document text and queries, then searches for one term will retrieve documents that contain either.

The advantage of just using mapping rules that remove characters like hyphens is that the equivalence classing to be done is implicit, rather than being fully calculated in advance: the terms that happen to become identical as the result of these rules are the equivalence classes. It is only easy to write rules of this sort that remove characters. Since the equivalence classes are implicit, it is not obvious when you might want to add characters. For instance, it would be hard to know to turn *antidiscriminatory* into *anti-discriminatory*.

An alternative to creating equivalence classes is to maintain relations between unnormalized tokens. This method can be extended to hand-constructed lists of synonyms such as *car* and *automobile*. These term relationships can be achieved in two ways. The usual way is to index unnormalized tokens and to maintain a query expansion list of multiple vocabulary entries to consider for a certain query term. A query term is then effectively a disjunction of several postings lists. The alternative is to perform the expansion during index construction. When the document contains automobile, we index it under car as well (and, usually, also vice-versa). Use of either of these methods is considerably less efficient than equivalence classing, as there are more postings to store and merge. The first method adds a query expansion dictionary and requires more processing at query time, while the second method requires more space for storing postings. Traditionally, expanding the space

required for the postings lists was seen as more disadvantageous, but with modern storage costs, the increased flexibility that comes from distinct postings lists is appealing.

These approaches are more flexible than equivalence classes because the expansion lists can overlap while not being identical. This means there can be an asymmetry in expansion. An example of how such an asymmetry can be exploited is shown in Figure 2.4: if the user enters windows, we wish to allow matches with the capitalized *Windows* operating system, but this is not plausible if the user enters window, even though it is plausible for this query to also match lowercase *windows*.

Query term	Terms in documents that should be matched
Windows	Windows
windows	Windows, windows, window
window	window, windows

► **Figure 2.4** An example of how asymmetric expansion of query terms can usefully model users' expectations.

The best amount of equivalence classing or query expansion to do is a fairly open question. Doing some definitely seems a good idea. But doing a lot can easily have unexpected consequences of broadening queries in unintended ways. For instance, equivalence-classing *U.S.A.* and *USA* to the latter by deleting periods from tokens might at first seem very reasonable, given the prevalent pattern of optional use of periods in acronyms. However, if I put in as my query term *C.A.T.*, I might be rather upset if it matches every appearance of the word *cat* in documents.

Below we present some of the forms of normalization that are commonly employed and how they are implemented. In many cases they seem helpful, but they can also do harm. In fact, you can worry about many details of equivalence classing, but it often turns out that providing processing is done consistently to the query and to documents, the fine details may not have much aggregate effect on performance.

Accents and diacritics. Diacritics on characters in English have a fairly marginal status, and we might well want *cliché* and *cliche* to match, or *naive* and *naïve*. This can be done by normalizing tokens to remove diacritics. In many other languages, diacritics are a regular part of the writing system and distinguish different sounds. Occasionally words are distinguished only by their accents. For instance, in Spanish, *peña* is 'a cliff', while *pena* is 'sorrow'. Nevertheless, the important question is usually not prescriptive or linguistic but is a question of how users are likely to write queries for these words. In many cases, users will enter queries for words without diacritics, whether for reasons of speed, laziness, limited software, or habits born of the days when it was hard to use non-ASCII text on many computer systems. In these cases, it might be best to equate all words to a form without diacritics.

CASE-FOLDING **Capitalization/case-folding.** A common strategy is to do *case-folding* by reducing all letters to lower case. Often this is a good idea: it will allow instances of *Automobile* at the beginning of a sentence to match with a query of *automobile*. It will also help on a web search engine when most of your users type in *ferrari* when they are interested in a *Ferrari* car. On the other hand, such case folding can equate words that might better be kept apart. Many proper nouns are derived from common nouns and so are distinguished only by case, including companies (*General Motors*, *The Associated Press*), government organizations (*the Fed* vs. *fed*) and person names (*Bush*, *Black*). We already mentioned an example of unintended query expansion with acronyms, which involved not only acronym normalization (*C.A.T.* → *CAT*) but also case-folding (*CAT* →*cat*).

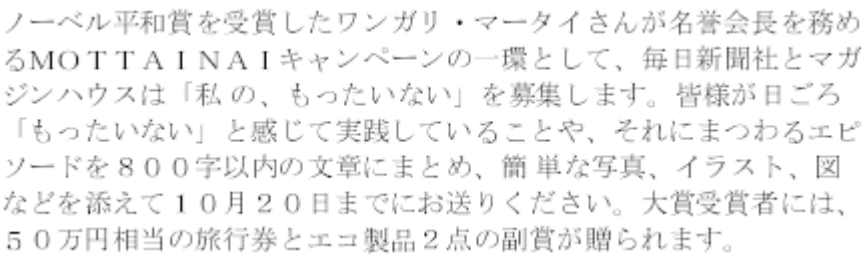
For English, an alternative to making every token lowercase is to just make some tokens lowercase. The simplest heuristic is to convert to lowercase words at the beginning of a sentence and all words occurring in a title that is all uppercase or in which most or all words are capitalized. These words are usually ordinary words that have been capitalized. Mid-sentence capitalized words are left as capitalized (which is usually correct). This will mostly avoid case-folding in cases where distinctions should be kept apart. The same task can be done more accurately by a machine learning sequence model which uses more features to make the decision of when to case-fold. This is known as
TRUECASING *truecasing*. However, trying to get capitalization right in this way probably doesn't help if your users usually use lowercase regardless of the correct case of words. Thus, lowercasing everything often remains the most practical solution.

Other issues in English. Other possible normalizations are quite idiosyncratic and particular to English. For instance, you might wish to equate *ne'er* and *never* or the British spelling *colour* and the American spelling *color*. Dates, times and similar items come in multiple formats, presenting additional challenges. You might wish to collapse together *3/12/91* and *Mar. 12, 1991*. However, correct processing here is complicated by the fact that in the U.S., *3/12/91* is *Mar. 12, 1991*, whereas in Europe it is *3 Dec 1991*.

Other languages. English has maintained a dominant position on the WWW; approximately 60% of web pages are in English (Gerrand 2007). But that still leaves 40% of the web, and the non-English portion might be expected to grow over time, since less than one third of Internet users and less than 10% of the world's population primarily speak English. And there are signs of change: Sifry (2007) reports that only about one third of blog posts are in English.

Other languages again present distinctive issues in equivalence classing. The French word for *the* has distinctive forms based not only on the gender (masculine or feminine) and number of the following noun, but also depending on whether the following word begins with a vowel: *le*, *la*, *l'*, *les*. We may well wish to equivalence class these various forms of *the*. German has a convention whereby vowels with an umlaut can be rendered instead as a two vowel digraph. We would want to treat *Schütze* and *Schuetze* as equivalent.

Japanese is a well-known difficult writing system, as illustrated in Figure 2.5. Modern Japanese is standardly an intermingling of multiple alphabets, principally Chinese characters, two syllabaries (hiragana and katakana) and western characters (Latin letters, Arabic numerals, and various symbols). While there are strong conventions and standardization through the education system over the choice of writing system, in many cases the same word can be written with multiple writing systems. For example, a word may be written in katakana for emphasis (somewhat like italics). Or a word may sometimes be written in hiragana and sometimes in Chinese characters. Successful retrieval thus requires complex equivalence classing across the writing systems. In particular, an end user might commonly present a query entirely in hiragana, because it is easier to type, just as Western end users commonly use all lowercase.


ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

► **Figure 2.5** Japanese makes use of multiple intermingled writing systems and, like Chinese, does not segment words. The text is mainly Chinese characters with the hiragana syllabary for inflectional endings and function words. The part in latin letters is actually a Japanese expression, but has been taken up as the name of an environmental campaign by 2004 Nobel Peace Prize winner Wangari Maathai. His name is written using the katakana syllabary in the middle of the first line. The first four characters of the final line express a monetary amount that we would want to match with ¥500,000 (500,000 Japanese yen).

Document collections being indexed can include documents from many different languages. Or a single document can easily contain text from multiple languages. For instance, a French email might quote clauses from a contract document written in English. Most commonly, the language is detected and language-particular tokenization and normalization rules are applied at a predetermined granularity, such as whole documents or individual paragraphs, but this still will not correctly deal with cases where language changes occur for brief quotations. When document collections contain multiple languages, a single index may have to contain terms of several languages. One option is to run a language identification classifier on documents and then to tag terms in the vocabulary for their language. Or this tagging can simply be omitted, since it is relatively rare for the exact same character sequence to be a word in different languages.

When dealing with foreign or complex words, particularly foreign names, the spelling may be unclear or there may be variant transliteration standards giving different spellings (for example, *Chebyshev* and *Tchebycheff* or *Beijing* and *Peking*). One way of dealing with this is to use heuristics to equivalence class or expand terms with phonetic equivalents. The traditional and best known such algorithm is the Soundex algorithm.

2.4. Stemming and lemmatization

For grammatical reasons, documents are going to use different forms of a word, such as *organize*, *organizes*, and *organizing*. Additionally, there are families of derivationally related word swith similar meanings, such as *democracy*, *democratic*, and *democratization*. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is ⇒ be
car, cars, car's, cars' ⇒ car

The result of this mapping of text will be something like:

the boy's cars are different colors ⇒
the boy car be differ color

STEMMING

LEMMATIZATION

LEMMA

However, the two words differ in their flavor. *Stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*. If confronted with the token *saw*, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open-source.

PORTER STEMMER

The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is *Porter's algorithm* (Porter 1980). The entire algorithm is too long and intricate to present here, but we will indicate its general nature. Porter's algorithm consists of 5 phases of word reductions, applied sequentially. Within each phase there are various conventions to select rules, such as selecting the rule from each rule group that applies to the longest suffix. In the first phase, this convention is used with the following rule group:

(2.1)

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Many of the later rules use a concept of the *measure* of a word, which loosely checks the number of syllables to see whether a word is long enough that it is reasonable to regard the matching portion of a rule as a suffix rather than as part of the stem of a word. For example, the rule:

(*m* > 1) EMENT →

would map *replacement* to *replac*, but not *cement* to *c*. The official site for the Porter Stemmer is:

http://www.tartarus.org/˜martin/PorterStemmer/

Other stemmers exist, including the older, one-pass Lovins stemmer (Lovins 1968), and newer entrants like the Paice/Husk stemmer (Paice 1990); see:

http://www.cs.waikato.ac.nz/˜eibe/stemmers/
http://www.comp.lancs.ac.uk/computing/research/stemming/

Figure 2.6 presents an informal comparison of the different behaviors of these stemmers. Stemmers use language-specific rules, but they require less knowledge than a lemmatizer, which needs a complete vocabulary and morphological analysis to correctly lemmatize words. Particular domains may also require special stemming rules. However, the exact stemmed form does not matter, only the equivalence classes it forms.

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation
Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres
Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret
Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

► **Figure 2.6** A comparison of three stemming algorithms on a sample text.

LEMMATIZER

Rather than using a stemmer, you can use a *lemmatizer*, a tool from Natural Language Processing which does full morphological analysis to accurately identify the lemma for each word. Doing full morphological analysis produces at most very modest benefits for retrieval. It is hard to say more, because either form of normalization tends not to improve English information retrieval performance in aggregate – at least not by very much. While it helps a lot for some queries, it equally hurts performance a lot for others. Stemming increases recall while harming precision. As an example of what can go wrong, note that the Porter stemmer stems all of the following words:

operate operating operates operation operative operatives operational

to oper. However, since *operate* in its various forms is a common verb, we would expect to lose considerable precision on queries such as the following with Porter stemming:

operational AND research
operating AND system
operative AND dentistry

For a case like this, moving to using a lemmatizer would not completely fix the problem because particular inflectional forms are used in particular collocations: a sentence with the words *operate* and *system* is not a good match for the query operating AND system. Getting better value from term normalization depends more on pragmatic issues of word use than on formal issues of linguistic morphology.

The situation is different for languages with much more morphology (such as Spanish, German, and Finnish). Results in the European CLEF evaluations have repeatedly shown quite large gains from the use of stemmers (and compound splitting for languages like German).

Exercise 2.1 [⋆]

Are the following statements true or false?

a. In a Boolean retrieval system, stemming never lowers precision.
b. In a Boolean retrieval system, stemming never lowers recall.
c. Stemming increases the size of the vocabulary.
d. Stemming should be invoked at indexing time but not while processing a query.

Exercise 2.2 [⋆]

Suggest what normalized form should be used for these words (including the word itself as a possibility):

a. 'Cos
b. Shi'ite
c. cont'd
d. Hawai'i
e. O'Rourke

Exercise 2.3 [⋆]

The following pairs of words are stemmed to the same form by the Porter stemmer. Which pairs would you argue shouldn't be conflated. Give your reasoning.

a. abandon/abandonment
b. absorbency/absorbent
c. marketing/markets
d. university/universe
e. volume/volumes

Exercise 2.4 [⋆]

For the Porter stemmer rule group shown in (2.1):

a. What is the purpose of including an identity rule such as SS →SS?
b. Applying just this rule group, what will the following words be stemmed to? *circus canaries boss*
c. What rule should be added to correctly stem *pony*?
d. The stemming for *ponies* and *pony* might seem strange. Does it have a deleterious effect on retrieval? Why or why not?

C. Latihan dan Jawaban

1. Penerapan *Case-Folding*, Tokenisasi, *Filtering,* dan *Stemming.*

   Input : Dalam setahun belakangan ini, pengaksesan KRS diganti ke SIAM (sebelumnya menggunakan SINERGI). Saat menggunakan SINERGI, fitur serta kecepatan akses sangat handal dan nyaman. Tapi setelah diganti menggunakan SIAM, keadaan berbalik menjadi buruk (lambat dan bahkan sampai keluar dengan sendirinya). *KRS tidak hanya berpengaruh bagi mahasiswa semester muda, tapi juga keseluruhan mahasiswa.

   Output : ...

Dokumen
Dalam setahun belakangan ini, pengaksesan KRS diganti ke SIAM (sebelumnya menggunakan SINERGI). Saat menggunakan SINERGI, fitur serta kecepatan akses sangat handal dan nyaman. Tapi setelah diganti menggunakan SIAM, keadaan berbalik menjadi buruk (lambat dan bahkan sampai keluar dengan sendirinya). *KRS tidak hanya berpengaruh bagi mahasiswa semester muda, tapi juga keseluruhan mahasiswa.

⇩

Case-Folding
dalam setahun belakangan ini, pengaksesan krs diganti ke siam (sebelumnya menggunakan sinergi). saat menggunakan sinergi, fitur serta kecepatan akses sangat handal dan nyaman. tapi setelah diganti menggunakan siam, keadaan berbalik menjadi buruk (lambat dan bahkan sampai keluar dengan sendirinya). *krs tidak hanya berpengaruh bagi mahasiswa semester muda, tapi juga keseluruhan mahasiswa.

⇩

Tokenisasi
dalam setahun belakangan ini pengaksesan krs diganti ke siam sebelumnya menggunakan sinergi saat menggunakan sinergi fitur serta kecepatan akses sangat handal dan nyaman tapi setelah diganti menggunakan siam keadaan berbalik menjadi buruk lambat dan bahkan sampai keluar dengan sendirinya krs tidak hanya berpengaruh bagi mahasiswa semester muda tapi juga keseluruhan mahasiswa

⇩

Filtering
setahun belakangan pengaksesan krs diganti siam sinergi sinergi fitur kecepatan akses handal nyaman diganti siam keadaan berbalik buruk lambat sendirinya krs berpengaruh mahasiswa semester muda keseluruhan mahasiswa

Stemming
tahun belakang akses krs ganti siam sinergi sinergi fitur cepat akses handal nyaman ganti siam ada balik buruk lambat sendiri krs pengaruh mahasiswa semester muda luruh mahasiswa

D. Daftar Pustaka

1. Manning, C. D., Raghavan, P., & Schutze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press.