MODUL TOPIK DALAM INFORMATION RETRIEVAL
(CMA 102)

MODUL PERTEMUAN 10
The Term Vocabulary and Postings Lists

DISUSUN OLEH

Dr. Fransiskus Adikara, S.Kom, MMSI

UNIVERSITAS ESA UNGGUL

2019

DOCUMENT DELINEATION AND CHARACTER SEQUENCE DECODING

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

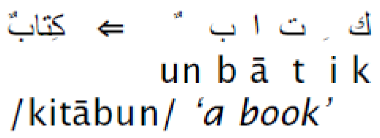
B. Uraian dan Contoh

1.1. Obtaining the character sequence in a document

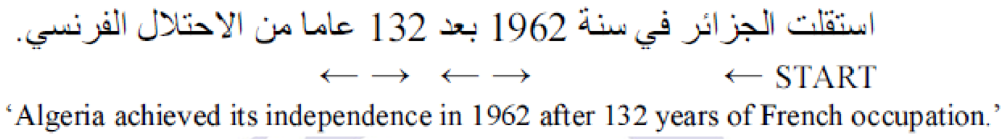
Digital documents that are the input to an indexing process are typically bytes in a file or on a web server. The first step of processing is to convert this byte sequence into a linear sequence of characters. For the case of plain English text in ASCII encoding, this is trivial. But often things get much more complex. The sequence of characters may be encoded by one of various single byte or multibyte encoding schemes, such as Unicode UTF-8, or various national or vendor-specific standards. We need to determine the correct encoding. Once the encoding is determined, we decode the byte sequence to a character sequence. We might save the choice of encoding because it gives some evidence about what language the document is written in.

The characters may have to be decoded out of some binary representation like Microsoft Word DOC files and/or a compressed format such as zip files. Again, we must determine the document format, and then an appropriate decoder has to be used. Even for plain text documents, additional decoding may need to be done. In XML documents, character entities, such as &amp;, need to be decoded to give the correct character, namely & for &amp;. Finally, the textual part of the document may need to be extracted out of other material that will not be processed. This might be the desired handling for XML files, if the markup is going to be ignored; we would almost certainly want to do this with postscript or PDF files. We will not deal further with these issues in this book, and will assume henceforth that our documents are a list of characters. Commercial products usually need to support a broad range of document types and encodings, since users want things to just work with their data as is. Often, they just think of documents as text inside applications and are not even aware of how it is encoded on disk. This problem is usually solved by licensing a software library that handles decoding document formats and character encodings.

The idea that text is a linear sequence of characters is also called into question by some writing systems, such as Arabic, where text takes on some two dimensional and mixed order characteristics, as shown in Figures 1.1 and 1.2. But, despite some complicated writing system conventions, there is an underlying sequence of sounds being represented and hence an essentially linear structure remains, and this is what is represented in the digital representation of Arabic, as shown in Figure 1.1.



► **Figure 1.1** An example of a vocalized Modern Standard Arabic word. The writing is from right to left and letters undergo complex mutations as they are combined. The representation of short vowels (here, /i/ and /u/) and the final /n/ (nunation) departs from strict linearity by being represented as diacritics above and below letters. Nevertheless, the represented text is still clearly a linear ordering of characters representing sounds. Full vocalization, as here, normally appears only in the Koran and children's books. Day-to-day text is unvocalized (short vowels are not represented but the letter for ā would still appear) or partially vocalized, with short vowels inserted in places where the writer perceives ambiguities. These choices add further complexities to indexing.


استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.
← → ← → ← START
'Algeria achieved its independence in 1962 after 132 years of French occupation.'

► **Figure 1.2** The conceptual linear order of characters is not necessarily the order that you see on the page. In languages that are written right-to-left, such as Hebrew and Arabic, it is quite common to also have left-to-right text interspersed, such as numbers and dollar amounts. With modern Unicode representation concepts, the order of characters in files matches the conceptual order, and the reversal of displayed characters is handled by the rendering system, but this may not be true for documents in older encodings.

1.2. Choosing a document unit

DOCUMENT UNIT

The next phase is to determine what the *document unit* for indexing is. Thus far we have assumed that documents are fixed units for the purposes of indexing. For example, we take each file in a folder as a document. But there are many cases in which you might want to do something different. A traditional Unix (mbox-format) email file stores a sequence of email messages (an email folder) in one file, but you might wish to regard each email message as a separate document. Many email messages now contain attached documents, and you might then want to regard the email message and each contained attachment as separate documents. If an email message has an attached zip file, you might want to decode the zip file and regard each file it contains as a separate document. Going in the opposite direction, various pieces of web software (such as latex2html) take things that you might regard as a single document (e.g., a Powerpoint file or a LATEX document) and split them into separate HTML pages for each slide or subsection, stored as separate files. In these cases, you might want to combine multiple files into a single document.

INDEXING GRANULARITY

More generally, for very long documents, the issue of indexing *granularity* arises. For a collection of books, it would usually be a bad idea to index an entire book as a document. A search for Chinese toys might bring up a book that mentions China in the first chapter and toys in the last chapter, but this does not make it relevant to the query. Instead, we may well wish to index each chapter or paragraph as a mini-document. Matches are then more likely to be relevant, and since the documents are smaller it will be much easier for the user to find the relevant passages in the document. But why stop there? We could treat individual sentences as mini-documents. It becomes clear that there is a precision/recall tradeoff here. If the units get too small, we are likely to miss important passages because terms were distributed over several mini-documents, while if units are too large we tend to get spurious matches and the relevant information is hard for the user to find.

The problems with large document units can be alleviated by use of explicit or implicit proximity search. The issue of index granularity, and in particular a need to simultaneously index documents at multiple levels of granularity, appears prominently in XML retrieval. An IR system should be designed to offer choices of granularity. For this choice to be made well, the person who is deploying the system must have a good understanding of the document collection, the users, and their likely information needs and usage patterns. For now, we will henceforth assume that a suitable size document unit has been chosen, together with an appropriate way of dividing or aggregating files, if needed.

C. Latihan dan Jawaban

1. What do you mean by digital document?

   Digital documents are those that exist in electronic form. They may also exist in physical paper form, or they may exist exclusively in digital form. Digital documents may be stored on electronic media, such as hard drives, iCloud servers, exabyte tape or any manner of electronic media.

2. What are the types of digital documents?

   The digital documents are :
   - Word Documents (.doc or .docx)
   - Portable File Documents (PDF)
   - Spreadsheet (.xls or .xlsx)
   - PowerPoint (.ppt or .pptx)

D. Daftar Pustaka

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

DETERMINING THE VOCABULARY OF TERMS

A. Kemampuan Akhir Yang Diharapkan

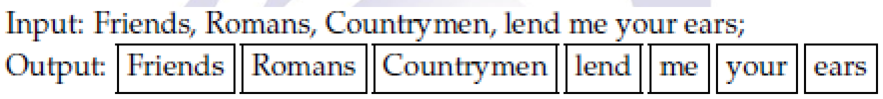
After reading this session, you will be able to answer the following questions:

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

2.1. Tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens*, perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:


Input: Friends, Romans, Countrymen, lend me your ears;
Output: Friends | Romans | Countrymen | lend | me | your | ears

TOKEN
TYPE
TERM

These tokens are often loosely referred to as terms or words, but it is sometimes important to make a type/token distinction. A *token* is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing. A *type* is the class of all tokens containing the same character sequence. A *term* is a (perhaps normalized) type that is included in the IR system's dictionary. The set of index terms could be entirely distinct from the tokens, for instance, they could be semantic identifiers in a taxonomy, but in practice in modern IR systems they are strongly related to the tokens in the document. However, rather than being exactly the tokens that appear in the document, they are usually derived from them by various normalization processes. For example, if the document to be indexed is *to sleep perchance to dream*, then there are 5 tokens, but only 4 types (since there are 2 instances of *to*). However, if *to* is omitted from the index (as a stop word), then there will be only 3 terms: *sleep*, *perchance*, and *dream*.

The major question of the tokenization phase is what are the correct tokens to use? In this example, it looks fairly trivial: you chop on whitespace and throw away punctuation characters. This is a starting point, but even for English there are a number of tricky cases. For example, what do you do about the various uses of the apostrophe for possession and contractions?

Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.

For *O’Neill*, which of the following is the desired tokenization?

$\boxed{\text{neill}}$
$\boxed{\text{oneill}}$
$\boxed{\text{o'neill}}$
$\boxed{\text{o'}}\ \boxed{\text{neill}}$
$\boxed{\text{o}}\ \boxed{\text{neill}}$?

And for *aren’t*, is it:

$\boxed{\text{aren't}}$
$\boxed{\text{arent}}$
$\boxed{\text{are}}\ \boxed{\text{n't}}$
$\boxed{\text{aren}}\ \boxed{\text{t}}$?

A simple strategy is to just split on all non-alphanumeric characters, but while $\boxed{\text{o}}\ \boxed{\text{neill}}$ looks okay, $\boxed{\text{aren}}\ \boxed{\text{t}}$ looks intuitively bad. For all of them, the choices determine which Boolean queries will match. A query of neill AND capital will match in three cases but not the other two. In how many cases would a query of o’neill AND capital match? If no preprocessing of a query is done, then it would match in only one of the five cases. For either Boolean or free text queries, you always want to do the exact same tokenization of document and query words, generally by processing queries with the same tokenizer. This guarantees that a sequence of characters in a text will always match the same sequence typed in a query.

LANGUAGE IDENTIFICATION

These issues of tokenization are language-specific. It thus requires the language of the document to be known. *Language identification* based on classifiers that use short character subsequences as features is highly effective; most languages have distinctive signature patterns.

For most languages and particular domains within them there are unusual specific tokens that we wish to recognize as terms, such as the programming languages C++ and C#, aircraft names like B-52, or a T.V. show name such as M*A*S*H – which is sufficiently integrated into popular culture that you find usages such as *M*A*S*H-style hospitals*. Computer technology has introduced new types of character sequences that a tokenizer should probably tokenize as a single token, including email addresses (jblack@mail.yahoo.com), web URLs (http://stuff.big.com/new/specials.html), numeric IP addresses (142.32.48.231), package tracking numbers (1Z9999W99845399981), and more. One possible solution is to omit from indexing tokens such as monetary amounts, numbers, and URLs, since their presence greatly expands the size of the vocabulary. However, this comes at a large cost in restricting what people can search for. For instance, people might want to search in a bug database for the line number where an error occurs. Items such as the date of an email, which have a clear semantic type, are often indexed separately as document metadata.

HYPHENS

In English, *hyphenation* is used for various purposes ranging from splitting up vowels in words (*co-education*) to joining nouns as names (*Hewlett-Packard*) to a copyediting device to show word grouping (*the hold-him-back and-drag-him-*

away maneuver). It is easy to feel that the first example should be regarded as one token (and is indeed more commonly written as just *coeducation*), the last should be separated into words, and that the middle case is unclear. Handling hyphens automatically can thus be complex: it can either be done as a classification problem, or more commonly by some heuristic rules, such as allowing short hyphenated prefixes on words, but not longer hyphenated forms.

Conceptually, splitting on white space can also split what should be regarded as a single token. This occurs most commonly with names (*San Francisco, Los Angeles*) but also with borrowed foreign phrases (*au fait*) and compounds that are sometimes written as a single word and sometimes space separated (such as *white space* vs. *whitespace*). Other cases with internal spaces that we might wish to regard as a single token include phone numbers ((800) 234-2333) and dates (Mar 11, 1983). Splitting tokens on spaces can cause bad retrieval results, for example, if a search for York University mainly returns documents containing *New York University*. The problems of hyphens and non-separating whitespace can even interact. Advertisements for air fares frequently contain items like *San Francisco-Los Angeles*, where simply doing whitespace splitting would give unfortunate results. In such cases, issues of tokenization interact with handling phrase queries, particularly if we would like queries for all of *lowercase*, *lower-case* and *lower case* to return the same results. The last two can be handled by splitting on hyphens and using a phrase index. Getting the first case right would depend on knowing that it is sometimes written as two words and also indexing it in this way. One effective strategy in practice, which is used by some Boolean retrieval systems such as Westlaw and Lexis-Nexis, is to encourage users to enter hyphens wherever they may be possible, and whenever there is a hyphenated form, the system will generalize the query to cover all three of the one word, hyphenated, and two word forms, so that a query for over-eager will search for over-eager OR "over eager" OR overeager. However, this strategy depends on user training, since if you query using either of the other two forms, you get no generalization.

Each new language presents some new issues. For instance, French has a variant use of the apostrophe for a reduced definite article 'the' before a word beginning with a vowel (e.g., *l'ensemble*) and has some uses of the hyphen with postposed clitic pronouns in imperatives and questions (e.g., *donnemoi* 'give me'). Getting the first case correct will affect the correct indexing of a fair percentage of nouns and adjectives: you would want documents mentioning both *l'ensemble* and *un ensemble* to be indexed under *ensemble*. Other languages make the problem harder in new ways. German writes *compound nouns* without spaces (e.g., *Computerlinguistik* 'computational linguistics'; *Lebensversicherungsgesellschaftsangestellter* 'life insurance company employee'). Retrieval systems for German greatly benefit from the use of a *compound-splitter* module, which is usually implemented by seeing if a word can be subdivided into multiple words that appear in a vocabulary. This phenomenon reaches its limit case with major East Asian Languages (e.g., Chinese, Japanese, Korean, and Thai), where text is written without any spaces between words. An example is shown in Figure 2.1. One approach here is to perform *word segmentation* as prior linguistic processing. Methods of word segmentation vary from having a large vocabulary and taking the longest vocabulary match with some heuristics for unknown words to the use of

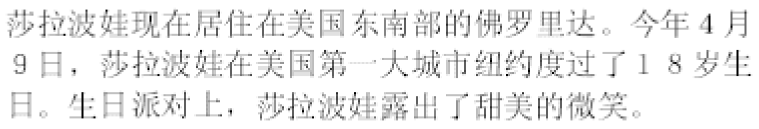
COMPOUNDS

COMPOUND-SPLITTER

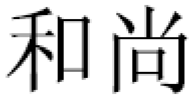
WORD SEGMENTATION

machine learning sequence models, such as hidden Markov models or conditional random fields, trained over hand-segmented words.

Since there are multiple possible segmentations of character sequences (see Figure 2.2), all such methods make mistakes sometimes, and so you are never guaranteed a consistent unique tokenization. The other approach is to abandon word-based indexing and to do all indexing via just short subsequences of characters (character *k*-grams), regardless of whether particular sequences cross word boundaries or not. Three reasons why this approach is appealing are that an individual Chinese character is more like a syllable than a letter and usually has some semantic content, that most words are short (the commonest length is 2 characters), and that, given the lack of standardization of word breaking in the writing system, it is not always clear where word boundaries should be placed anyway. Even in English, some cases of where to put word boundaries are just orthographic conventions – think of *notwithstanding* vs. *not to mention* or *into* vs. *on to* – but people are educated to write the words with consistent use of spaces.


莎拉波娃现在居住在美国东南部的佛罗里达。今年 4 月 9 日，莎拉波娃在美国第一大城市纽约度过了 1 8 岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

► **Figure 2.1** The standard unsegmented form of Chinese text using the simplified characters of mainland China. There is no whitespace between words, not even between sentences – the apparent space after the Chinese period (◦) is just a typographical illusion caused by placing the character on the left side of its square box. The first sentence is just words in Chinese characters with no spaces between them. The second and third sentences include Arabic numerals and punctuation breaking up the Chinese characters.


和尚

► **Figure 2.2** Ambiguities in Chinese word segmentation. The two characters can be treated as one word meaning ‘monk’ or as a sequence of two words meaning ‘and’ and ‘still’.

2.2. Dropping common terms: stop words

STOP WORDS

COLLECTION FREQUENCY

STOP LIST

Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called *stop words*. The general strategy for determining a stop list is to sort the terms by *collection frequency* (the total number of times each term appears in the document collection), and then to take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as a *stop list*, the members of which are then discarded during indexing. An example of a stop list is shown in Figure 2.3. Using a stop list significantly reduces the number of postings that a system has to store. And a lot of the time not indexing stop words does little harm: keyword searches with terms like the and by don’t seem very useful. However, this is not true for phrase searches. The phrase query “President of the United States”, which contains two stop words, is more precise than President AND “United States”. The meaning of flights to London is likely to be lost if the word to is stopped out. A search for Vannevar Bush's article *As we may think* will be difficult if the first three words are stopped out, and the system searches simply for documents containing the word think. Some special query types are disproportionately affected. Some song titles and well known pieces of verse consist entirely of words that are commonly on stop lists (*To be or not to be*, *Let It Be*, *I don't want to be*, . . .).

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

► **Figure 2.3** A stop list of 25 semantically non-selective words which are common in Reuters-RCV1.

The general trend in IR systems over time has been from standard use of quite large stop lists (200–300 terms) to very small stop lists (7–12 terms) to no stop list whatsoever. Web search engines generally do not use stop lists. Some of the design of modern IR systems has focused precisely on how we can exploit the statistics of language so as to be able to cope with common words in better ways. So for most modern IR systems, the additional cost of including stop words is not that big – neither in terms of index size nor in terms of query processing time.

C. Latihan dan Jawaban

1. Penerapan Tokenisasi dan *Stopword Removal*

   Input : hotel yang bisa dibilang modern dengan nuansa klasik khas jogja. pelayanan yang baik, fasilitas lengkap ada kolam renang, kamar yang cukup luas juga kebersihannya cukup terjaga, letak yang cukup strategis tidak jauh dari tugu jogja serta menu sarapan yang enak dan beragam membuat kita betah.

   Output : ...

- Proses pertama yang dilakukan adalah proses tokenisasi, proses ini dilakukan untuk mendapatkan token atau potongan kata yang akan menjadi entitas yang memiliki nilai dalam penyusunan matriks dokumen pada proses selanjutnya.

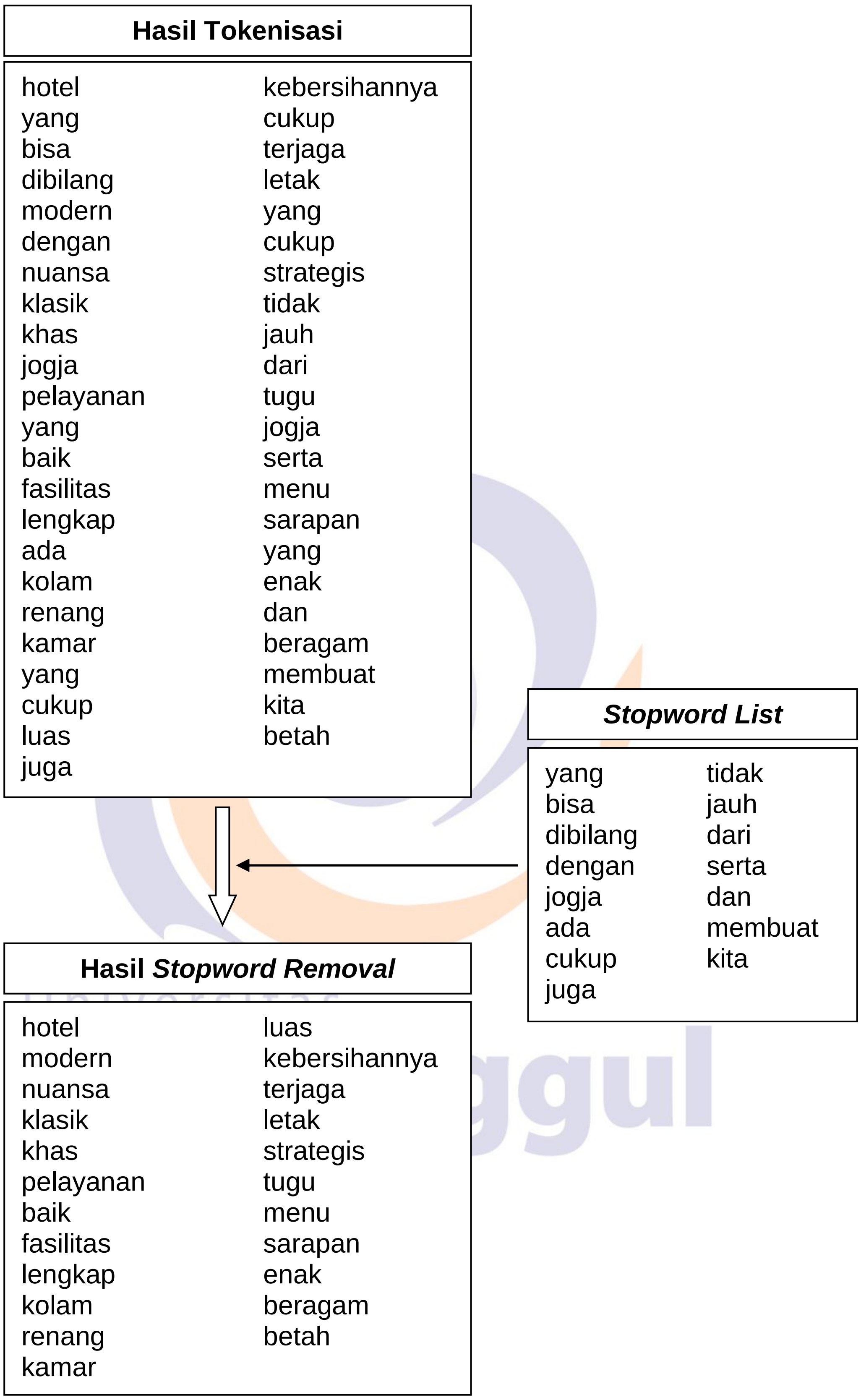
Dokumen
hotel yang bisa dibilang modern dengan nuansa klasik khas jogja. pelayanan yang baik, fasilitas lengkap ada kolam renang, kamar yang cukup luas juga kebersihannya cukup terjaga, letak yang cukup strategis tidak jauh dari tugu jogja serta menu sarapan yang enak dan beragam membuat kita betah.

⇩

Hasil Tokenisasi

hotel	ada	tidak
yang	kolam	jauh
bisa	renang	dari
dibilang	kamar	tugu
modern	yang	jogja
dengan	cukup	serta
nuansa	luas	menu
klasik	juga	sarapan
khas	kebersihannya	yang
jogja	cukup	enak
pelayanan	terjaga	dan
yang	letak	beragam
baik	yang	membuat
fasilitas	cukup	kita
lengkap	strategis	betah

- Proses selanjutnya setelah dilakukan pemisahan kata pada dokumen adalah proses *stopword removal*. Tahap ini akan memproses kata hasil tokenisasi menjadi lebih sedikit dengan cara mengurangi kata tersebut dengan kata yang termasuk dalam *stopwords*.



Hasil Tokenisasi

hotel	kebersihannya
yang	cukup
bisa	terjaga
dibilang	letak
modern	yang
dengan	cukup
nuansa	strategis
klasik	tidak
khas	jauh
jogja	dari
pelayanan	tugu
yang	jogja
baik	serta
fasilitas	menu
lengkap	sarapan
ada	yang
kolam	enak
renang	dan
kamar	beragam
yang	membuat
cukup	kita
luas	betah
juga	

Stopword List

yang	tidak
bisa	jauh
dibilang	dari
dengan	serta
jogja	dan
ada	membuat
cukup	kita
juga	

Hasil *Stopword Removal*

hotel	luas
modern	kebersihannya
nuansa	terjaga
klasik	letak
khas	strategis
pelayanan	tugu
baik	menu
fasilitas	sarapan
lengkap	enak
kolam	beragam
renang	betah
kamar	

D. Daftar Pustaka

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