

Natural Language Processing
Prof. Pawan Goyal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 05
Text Processing: Basics

Hello everyone. Welcome back to the final lecture of the first week. In the last lecture we were discussing about various empirical laws, in particular Zipf's law and Heap's law; that how the, what is the vocabulary are distributed in a corpus.

We say that the distribution is not very uniform. There are certain words that are very very common. So, we saw that roughly hundred words in the vocabulary made for 50 percent of the corpus that by the time mean that number of tokens. And on the other hand there are 50 percent were words in the vocabulary that occur only once. And we discussed whatever various relationships among my vocabulary size and the number of tokens that I observe in a corpus. And also how they grow with respect to each other, and zipfs law gave me a relation between the frequency and the rank of a word.

So, today in this lecture we will start with the basic key processing in language. So, we will cover the basic concepts, and what are the challenges that one might face while doing the processing. So, we are going to the Basics of Text Processing.

(Refer Slide Time: 01:36)

Text processing: tokenization

What is Tokenization?
Tokenization is the process of segmenting a string of characters into words.

Depending on the application in hand, you might have to perform *sentence segmentation* as well.

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 2 / 26

So, we will start with the problem of Tokenization, as the name would suggest. Remember the name token: token is an individual word in my corpus. So now what happens when I am preprocessing the text in given in any language? What I will face is a string of characters; the sequence of characters. Now I need to indentify what are all the different words that are there in this sequence. Now tokenization is a process by which I convert the string of characters into sequence of various words.

So, I am trying to segment it by the various words that I am observing. Now, before going into what is tokenization I will just talk about slight little problem sentence segmentation. So, this you may or may not have to do always and it depends on what is your application. For example, suppose you are doing classification for the whole document in to certain classes you might not have to go to the individual sentence and you can just talk about what are the various words that are present in this document.

On the other hand, suppose you are trying to find out what are the important sentences in this document; in that application you will have to go to the individual sentence. So now, if you have to go to the individual sentence the first task that you will face is how do I segment these whole documents into a sequence of sentences. So, this is sentence one, sentence two and so on, and this task is called Sentence Segmentation.

(Refer Slide Time: 03:22)

Sentence Segmentation

The problem of deciding where the sentences begin and end.

Challenges Involved

- While '!', '?' are quite unambiguous
- Period "." is quite ambiguous and can be used additionally for
 - Abbreviations (Dr., Mr., m.p.h.)
 - Numbers (2.4%, 4.3)

Approach: build a binary classifier

For each "."

- Decides EndOfSentence/NotEndOfSentence
- Classifiers can be: hand-written rules, regular expressions, or machine learning

Pawan Goyal (IIT Khargpur) Text Processing: Basics Week 1: Lecture 3 3 / 26

Now, you might feel that this is very trivial task, but let us see is it trivial. So what is sentence segmentation? It is a problem of deciding where my sentence begins and ends

so that I have a complete unit of words that I call as a sentence. Now do you think there might be certain challenges involved? Suppose I am talking about the language English, can I always say that wherever I have a dot it is end of the sentence? Let us see.

So, there are many ways in which I can end my sentence. So, I can have exclamation or question mark that ends the sentence and they are mostly unambiguous. So, whenever I have exclamation or question mark I can say probably this is the end of the sentence, but is the case the same with a dot. So, I can think of a scenario where I have a dot in English but it is not the end of the sentence. So, we can find all sorts of abbreviations. They end with a period like, doctor, mister, mph; so you have three dots here. So, you cannot each of this as the end of your sentence.

So, again you have numbers: 2.4, 4.3 and so on. That means the problem of deciding whether a particular dot is the end of the sentence or not is not entirely trivial. So, I need to build certain algorithm for finding out is it my end of the sentence. In text processing we face this kind of problem in nearly every simple task that we are doing. So, even if it looks a trivial task we face with this problem that can I always call dot as end of the sentence.

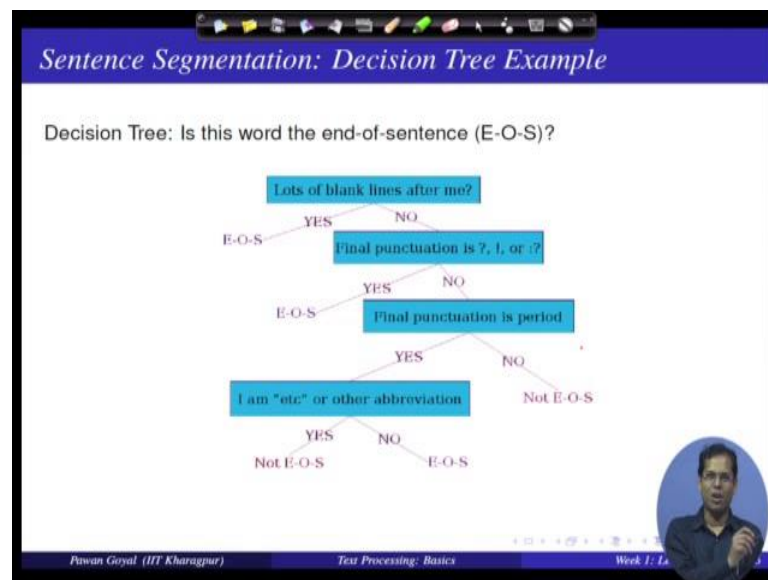
So, how do we go about solving this? Now if you think about it, whenever I see a dot or question mark or exclamation I always have to decide one of the two things: is it the end of the sentence or is not the end of the sentence. Any data point that I am seeing I have to divide into one of these two classes. If you think of these as two classes end of the sentence or not end of the sentence, each point you have to divide into one of the two classes. And this in general, this problem in general is called classification problem. You are classifying into one of the two classes.

Now, so the idea is very very simple. So, you have two classes and each data point you have to divide into one of the two classes; that means, you have to build some sort of plural algorithm for doing that. In this case I have to build the binary classifier, what they mean by a binary classifier? There are two classes: end of the sentence or not end of the sentence. In general there can be multiple classes.

So now, for each dot or in general for every word I need to decide whether this is the end of the sentence or not the end of the sentence. So, in general my classifiers that I will build can be some rules that I write by hand, some simple (Refer Time: 06:27) nice rules

or it can be some expressions. I say my particular example matches with this set of expressions it is one class, if I does not match it is of other class. Or I can build a machine learning classifier. So, in this particular scenario what can be the simplest thing to do? Let us see. Can we build a simple rule based classifier?

(Refer Slide Time: 06:49)



So, we will start with example of a simple decision tree. So, by decision tree I mean a set of if-then-else say statements. So, I am at a particular word I want to decide whether this is the end of the sentence or not. So, I can have the simple if-then-else kind of decision tree here. So, met a word and the first thing I check is are there lots of blank lines after that. So, this would happen in a text whenever this is the end of the paragraph and there are some blank lines.

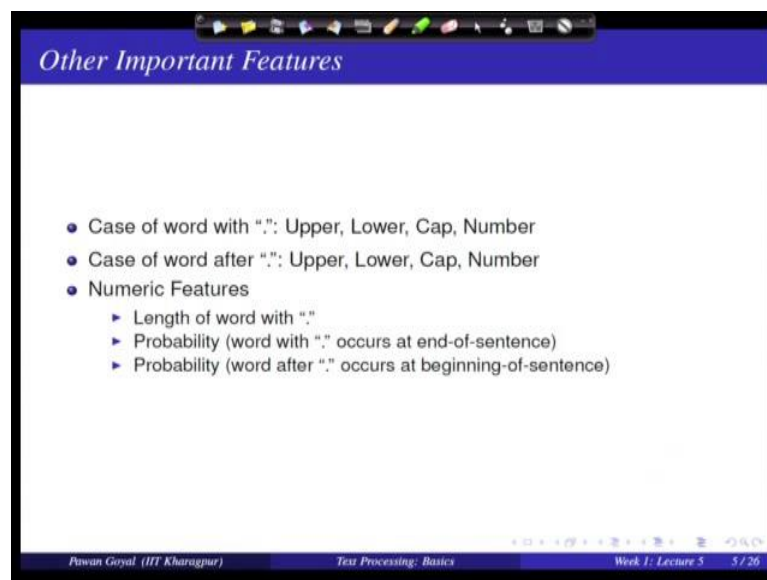
So, if I feel that there are a lot of blank lines after me; that means, after this word I may say this might be the end of the sentence with a good confidence. So, that is why the branch here says yes this is the end of the sentence. But suppose there are not lots of blank lines then I will check if the final punctuation is a question mark exclamation or a colon in that case. So, there are quite unambiguous and I may say this is the end of the sentence.

Now suppose it is not, then I will check if the final punctuation is a period. So, if it is not a period this is easy to say that this is not the end of the sentence, but suppose this is an end of this is a period. So, again I cannot say for certain if it is the end of the sentence, so

I give a again check. For simplicity I might have a list of abbreviations, and I can check if the word that I am correctly facing is one of the abbreviations in my list. If it is there I say this is not end of the sentence, if it is so here I am etcetera or any other abbreviation if the answer is yes I am not end of the sentence, if the answer is no that means this word is not an abbreviation and this will be the end of the sentence. This is very very simple if-then-else rules.

This may not be correct, but this is one particular way in which this problem can be solved. In general you might want to use some other sort of indications; we call them as various features. These are various observations that you make from your corpus. So, what are some examples?

(Refer Slide Time: 09:02)



The slide is titled "Other Important Features" in a blue header. It contains a bulleted list of features:

- Case of word with ".": Upper, Lower, Cap, Number
- Case of word after ".": Upper, Lower, Cap, Number
- Numeric Features
 - ▶ Length of word with "."
 - ▶ Probability (word with "." occurs at end-of-sentence)
 - ▶ Probability (word after "." occurs at beginning-of-sentence)

At the bottom of the slide, there is a footer with the text: "Pawan Goyal (IIT Khuragpur) Text Processing: Basics Week 1: Lecture 5 5 / 26".

Suppose I see the word that is ending with dot, can I use this as a feature whether my word starts with an upper case, lower case, cap, all caps or is it number. How will that help? So, let us see I am here and my word is 4.3. So, I am at dot I want to find out if it is the end of the sentence, if I can say that the current word is a number it is a high probability that this will be in number and it will not be the end of the sentence, so this can be used as another feature.

Again by feature you can think of a simple rule whether the word I am currently at is a number. Or I can use the fact where the case of the word with dots its upper case or lower case. So, what happens generally in abbreviations? We are mostly in upper case.

So, suppose I have doctor and it starts with an upper case I can say that this might be an abbreviation. Saying with the lower case: lower case will give me more probability that this is not an abbreviation.

Similarly I can also use in the case of the word after dot. So is it upper case, lower case, capital or number. So, how will that help? Again whenever I have the end of the sentence the next word in general starts with a capital. So, again this can be used. What can be some other features? So, I can have some numerical features, that is I will have certain thresholds. What is the length of the word ending with dot? Is it if the length is small it might be an abbreviation, if the length is larger it might not be an abbreviation. And I can also use probably. What is the probability that the word that is ending with dot occurs at the end of the sentence. So, if it is really the end of the sentence it might happen then that in a large corpus this end sentence quite often.

Something I can do with the next word after dot, is it the start of the sentence. What is the probability that it occurs in the start of the sentence in a large corpus? So, you might be able to use any of these features to decide given a particular word is it the end of the sentence or not. So now, suppose I ask you this question do you have the same problem in other languages like Hindi?

So, in Hindi you will see that in general there is only agenda that you use to indicate the end of the sentence and this is not used for any other purpose. So this problem you will see is again language dependent. This problem is there for English, but not so for Hindi. But you will see there are other problems that do not exist for English language, but are there for other Indian languages. We will see some of those examples in the same (Refer Time: 12:14).

(Refer Slide Time: 12:15)

Implementing Decision Trees

- Just an if-then-else statement
- Choosing the features is more important
- For numeric features, thresholds are to be picked
- With increasing features including numerical ones, difficult to set up the structure by hand
- Decision Tree structure can be learned using machine learning over a training corpus

Basic Idea
Usually works top-down, by choosing a variable at each step that best splits the set of items.
Popular algorithms: ID3, C4.5, CART

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 6 / 26

So, how do we implement a decision tree? As you have seen this is simple if-then-else statement. So now what is important is that you choose the correct set of features. So, how do you go about choosing the set of features? You will see in your from your data what are some observations that can separate my two classes here. So, my two classes here are; end of the sentence and non end of the sentence. And what are the observations we were having? In general it might be an abbreviation in the case of the word and that is before the dot: maybe upper case or lower case and one of these might indicate one class the other might indicate other class. So, all these are my observations that I use as my features.

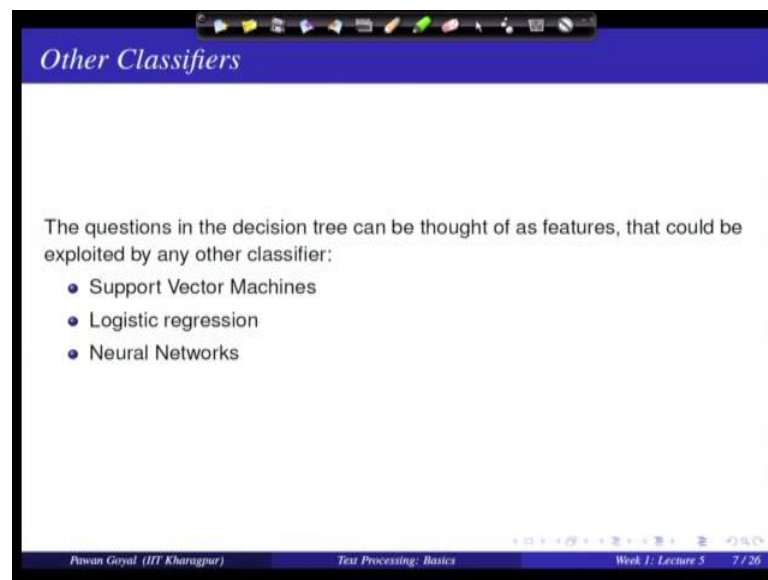
Now, whenever I am using numerical features like the length of the word before dot, I need to pick some sort of threshold; that is whether the length of the word is between 2 to 3 or say more than 3 between 5 to 7 like that. So, my tree can be if the length of the word is between 5 to 7 I could one class otherwise I could another class.

Now here is one problem; suppose I keep on increasing my features it can be both numerical and non numerical features. It might be difficult to set up my if-then-else rules by hand. So, in that scenario I can try to use some sort of machine learning technique to learn this decision tree. In the literature there are lots of such algorithms available that given a data and a set of features we will construct a decision tree for you.

So, I will just give you the names of some of the algorithms. And the basic idea on this they work is that at every point we have to choose a particular split. So, you have to choose a feature value that it splits my data into certain parts. And I have certain criteria to find out what is the best way to split. So, one particular criterion is what is the information given by this. So, these algorithms that we have mentioned here like ID3, C4.5, CART they all use one of these criteria.

In general once you have identified what are your interesting features for these tasks. You are not limited to only one classifier a decision tree, you can also try out some other classifiers like.

(Refer Slide Time: 14:53)



Support vector machines, logistic regression and neural networks. These all these are quite popular classifiers for various analytic applications. So, we will talk about some of these as we will go to some advanced topics in this course

(Refer Slide Time: 15:11)

Word Tokenization

What is Tokenization?
Tokenization is the process of segmenting a string of characters into words.

I have a can opener; but I can't open these cans.

Word Token

- An occurrence of a word
- For the above sentence, 11 word tokens.

Word Type

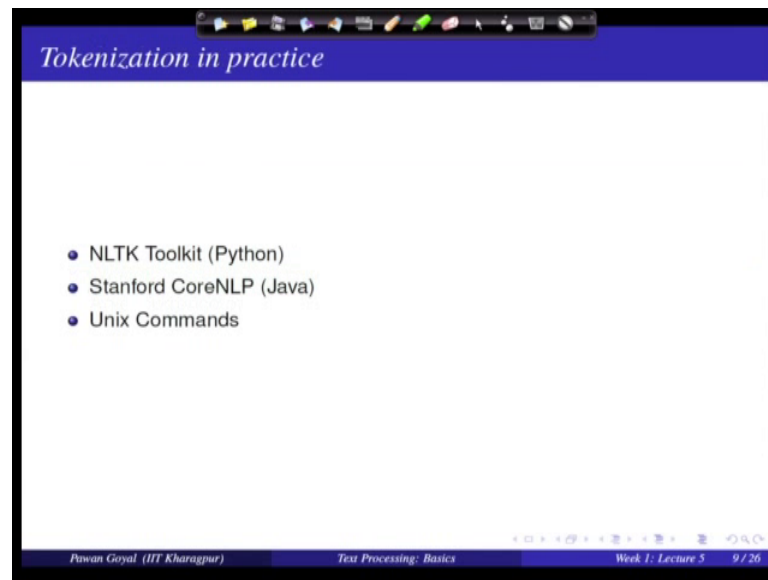
- A different realization of a word
- For the above sentence, 10 word types.

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 8 / 26

Now coming back to our problem tokenization; we said that tokenization is a process of segmenting a string of characters into words, finding out what are the different words in this question. Now remember we talked about token and type distinction; suppose I give you a simple sentence here I have a can opener, but I cannot open these cans.

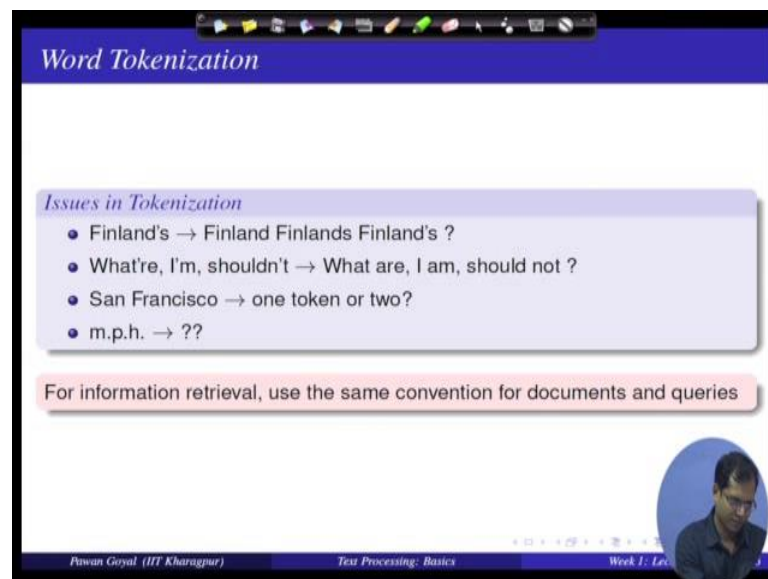
How many tokens are there? If you count there are 11 different occurrences of words. So, you have 11 word tokens, but how many unique words are there. So, you will find there are only 10 unique words, which word repeats, is the word I repeats twice. So, there are 10 types and 11 tokens. So, my tokenization is to find out each of the 11 word tokens from the sentence.

(Refer Slide Time: 16:04)



In practice at least for English you can use certain toolkits that are available like NLTK in Python, CoreNLP in Java and you can also use the Unix commands. So, in this course you will mainly be using NLTK toolkit for doing all pre processing task and in some other tasks as well, but in general you can use any of these three possibilities.

(Refer Slide Time: 16:34)



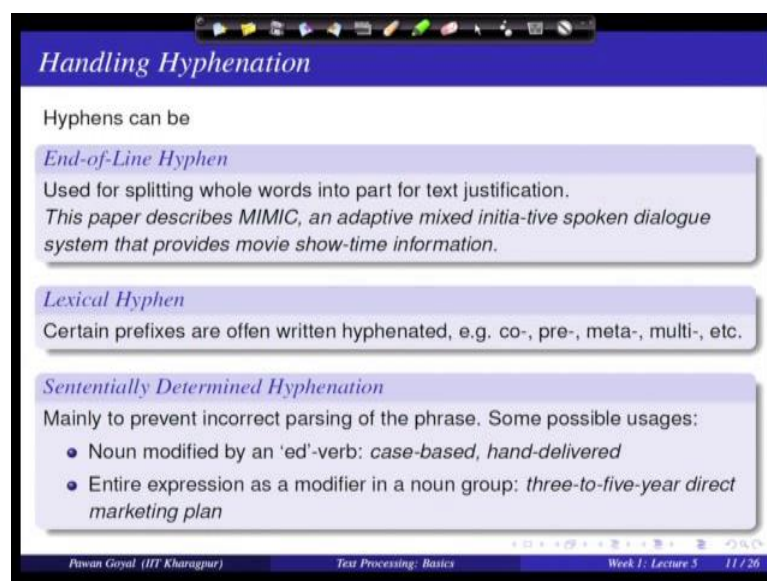
So for English most of the problems that we will see are taken care of the tokenizers that we have discussed previously but, still it is good to know; what are the challenges that are involved when I tried to design a tokenization algorithm.

See for example, here you will see that if I encounter a word like Finland's in my data. So, one question that I have is whether I treat it as simple Finland, as it is Finland's or I convert it to Finland's by removing the apostrophe. So, this question you might also try to defer to the next processing step that you will see, but sometimes you might want to tackle this in the same (Refer Time: 17:14). Similarly, if you see what are, do I treat it as a single token or two tokens what are? This trouble you might have to solve in the same step, whether I treat it as a single token or multiple tokens; same with I am, should not and so on.

Similarly whenever your name end at each like San Francisco, should I treat it as a single token or two separate tokens? Now remember when we were talking about some of the cases why (Refer Time: 17:45) hard. So, you might have to find out that this particular sequence of tokens is a single entity and treat it as a single entity, not as multiple different tokens. So, this problem is related. Similarly if you find m dot p dot h whether you call it a single token or multiple tokens.

So now, there are no fixed answers to these and some of these might depend on what is the application for which you are doing this pre processing. But one thing you can always keep in mind; suppose you are doing for the application of information retrieval if the same sort of steps that you apply for your documents should be applied to your query as well otherwise you will not be able to match them perfectly. Suppose, if I am using it for information retrieval, so I should use the same convention for both my documents as well as the queries.

(Refer Slide Time: 18:39)



Handling Hyphenation

Hyphens can be

- End-of-Line Hyphen**
Used for splitting whole words into part for text justification.
This paper describes MIMIC, an adaptive mixed initia-tive spoken dialogue system that provides movie show-time information.
- Lexical Hyphen**
Certain prefixes are often written hyphenated, e.g. co-, pre-, meta-, multi-, etc.
- Sententially Determined Hyphenation**
Mainly to prevent incorrect parsing of the phrase. Some possible usages:
 - Noun modified by an 'ed'-verb: *case-based, hand-delivered*
 - Entire expression as a modifier in a noun group: *three-to-five-year direct marketing plan*

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 11 / 26

So then another problem can be; how do I handle hyphens in my day? This looks again a simple problem, but we will see it is not that simple. So, let us see some kind of examples: what are the various sorts of hyphens that can be there in my corpus?

So, here I have a sentence from a research paper abstract and the sentence says this paper describes MIMIC an adaptive mixed initia-tive spoken dialogue system that provides movie show-time information. So, in this sentence itself you see two different hyphens: one is with initia-tive another is show hyphen time.

So, now can you see that these two are different hyphens; the first hyphen is not in general that I will use in my text, second hyphen I can use in my text I can write show time with an hyphen, but how did this hyphen initiative come into the corpus. So, we have given this a title end of line hyphen. So, what happens in research papers for example, whenever you write a sentence you might have to do some sort of justification and that is where you end the line even if it is not the end of this of the word. So, you will end up with in hyphen.

So now, when you are trying to pre process and when you are retrieving such kind of hyphens you might have to join these together, and you should, you have to say that this is a single word initiative and not initia hyphen tive. But again this is not trivial because for show time you will not do the same; show time you might want to keep it as it is.

Then there are some other kinds of hyphens like; lexical hyphens. So, you might have these hyphens with various prefixes like co-, pre-, meta-, multi-, etcetera. Sometimes they are sententially determined hyphens also, that is they put hyphens so that it becomes easier to interpret the sentence. Like here case-based, hand-delivered etcetera are optional.

Similarly, if you see in the next sentence three-to-five-year direct mark marketing plan; three to five year can be written perfectly without keeping the hyphens, but here you are putting it so that it becomes easier to interpret that particular occurrence. Again when you are doing tokenization your problem that how do I handle all these hyphens.

(Refer Slide Time: 21:00)

Language Specific Issues: French and German

French
l'ensemble: want to match with un ensemble

German
Noun coumpounds are not segmented

- Lebensversicherungsgesellschaftsangeestlter
- 'life insurance company employee' ✓
- Compound splitter required for German information retrieval

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 12/26

Further, there are various issues that you might face for certain languages, but not others. For an example like in French if you have a token like ensemble, so you might want to match it with ensemble. So, that might be a similar problem that we are facing in English, but let us take something in German. So I have this big sentence here, but the problem is that this is not a single word. This is a compound composed of four different words and the corresponding English meaning is this one. So, you have four words in English. So, when you are putting in French they make a compound.

So, now what is the problem that you will face when you are processing the German text? And you are trying to tokenize it? So, you might want to find out what are the

individual words in this particular compound. So, you need some sort of compound split up for German. So, the problem is there for German not so much for English.

(Refer Slide Time: 22:26)

The slide is titled "Language Specific Issues: Chinese and Japanese". It contains the following content:

No space between words

莎拉波娃现在居住在美国东南部的佛罗里达。

莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

Sharapova now lives in US southeastern Florida

Japanese: further complications with multiple alphabets intermingled.

フォーチュン500社は情報不足のため時間あた\$500K(約6,000万円)

Below the Japanese sentence, four boxes are labeled: Katakana, Hiragana, Kanji, and Romaji. Arrows point from these labels to the corresponding parts of the sentence: Katakana points to "フォーチュン", Hiragana points to "社", Kanji points to "は情報不足のため時間", and Romaji points to "あた\$500K(約6,000万円)".

At the bottom of the slide, there is a footer with the text: "Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Les."

So now, what happens if I am making a language like Chinese or Japanese? So, here is a sentence in Chinese. So, what do you see in Chinese words are written without any spaces in between. Now, when you are doing the pre processing your task is to find out what are the individual word tokens in this Chinese sentence. This problem is also difficult because in general for a given utterance of a sequence of characters there might be more than one possible ways of breaking into sequence of words and both might be perfectly valid possibilities.

So, in Chinese we will not have not have any space between words and I have to find out what are the places where I have to break these words; and this problem is called word tokenization. Same problem happens with Japanese and here for the complications because they are using four different steps like Katakana, Hiragana, Kanji and Romaji. So, these problems become a bit more severe.

(Refer Slide Time: 23:30)

Language Specific Issues: Sanskrit

सत्यम्ब्रूयात्प्रियम्ब्रूयान्नब्रूयात्सत्यमप्रियमिष्यन्चनानृतम्ब्रूयादेषधर्मःसनातनः

satyambrūyātpriyambrūyānnabrūyātsatyamapriyampriyamcanāṇṛtambrūyād-
eṣadharmāḥsanātanaḥ.

"One should tell the truth, one should say kind words; one should neither tell
harsh truths, nor flattering lies; this is a rule for all times."

Segmented Text:
satyam brūyāt priyam brūyāt na brūyāt satyam apriyam priyam ca na anṛtam
brūyāt eṣaḥ dharmāḥ sanātanaḥ.

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lex

Now, the same problem is there even for Sanskrit. So, if some of you have taken a Sanskrit course in your class 8th or 10th you might be familiar with the rules of Sandians in Sanskrit language. So, that is it.

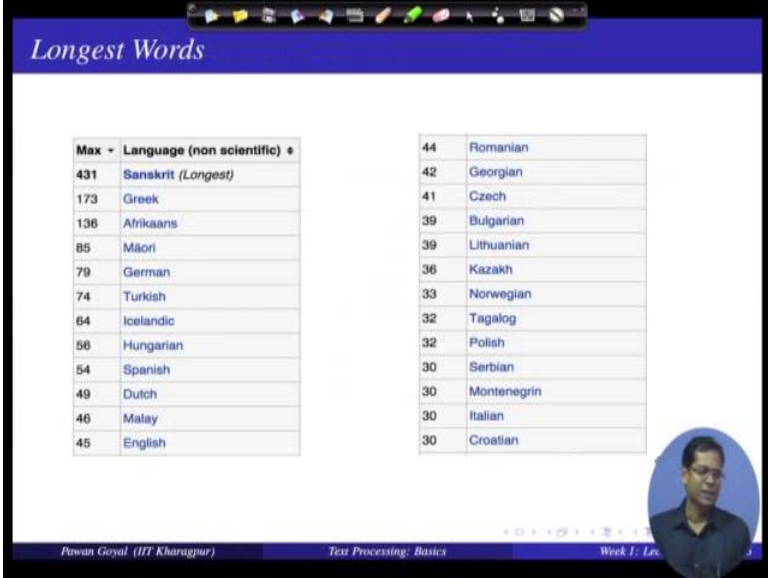
This is a simple single sentence in Sanskrit, but this is a huge this looks like a single word, it is not a single word. It is composed of multiple words in Sanskrit and they are combined with a Sandi relation. This stands for nice proverb in Sanskrit that translates in English as one should tell the truth, one should say kind words, one should neither tell harsh truths nor flattering lies; this is a rule for all times - this is a proverb.

And this is a single sentence that talks about this proverb, but there all the words are combined with Sandi relation. So, if we try to undo the Sandi this is what you will find at the segmented text. So, there are multiple words in this sentence they are combined to make a single, it looks like a single word.

So, this problem we saw in Chinese, Japanese and Sanskrit, but in Sanskrit the problem is slightly more complicated and why is that. So, in Japanese and in Chinese when you try to combine various words together you simply concatenate them, you put them one after another without making any changes at the boundary. It does not happen in Sanskrit when you combine two words you also make certain changes at the boundary and this is called the Sandi operation.

So, in this particular case since see here I have the word 'bruyat' and the word 'na', but when I am combining I am writing it 'bruyanna'. So, you see here the letter 't' gets changed to 'n' that means when I am trying to analyze the sentence, so this particular sentence in Sanskrit I need to find out not only what are the breaks, but what is the corresponding word from which this sentence you derived. So, from here to find out the actual words are bruyat lesna that gives me this 'bruyat'. And this is very very common in Sanskrit that you are always combining words by doing a Sandi operation. So, this further complicates my problem of word tokenization or segmentation.

(Refer Slide Time: 26:14)



Max	Language (non scientific)
431	Sanskrit (Longest)
173	Greek
136	Afrikaans
85	Māori
79	German
74	Turkish
64	Icelandic
56	Hungarian
54	Spanish
49	Dutch
46	Malay
45	English
44	Romanian
42	Georgian
41	Czech
39	Bulgarian
39	Lithuanian
36	Kazakh
33	Norwegian
32	Tagalog
32	Polish
30	Serbian
30	Montenegrin
30	Italian
30	Croatian

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lec.

So this is just a list from Wikipedia what are the longest words in various languages. Then note this sentence is about the words. You see in Sanskrit the longest word is composed of 431 characters, it is a compound. And then you have Greek and Afrikaans and other languages, in English you will see that the longest word is of 45 characters is non-scientific.

(Refer Slide Time: 26:36)

Longest Words

Compound word composed of 431 letters, from the Varadāmbikā Parinaya Campū by Tirumalāmba

निरन्तरान्धकारिता-दिगन्तर-कन्दलबमन्द-सुधारस-बिन्दु-सान्द्रतर-धनाधन-वृन्द-सन्वेहकर-
स्यन्दमान-मकरन्द-बिन्दु-बन्धुरतर-माकन्द-तरु-कुल-तल्प-कल्प-मृदुल-सिकता-जाल-जटिल-
मूल-तल-मरुक्क-मिलबलघु-लघु-लय-कलित-रमणीय-पानीय-शालिका-बालिका-करार-विन्द-
गलन्तिका-गलबेला-लवङ्ग-पाटल-धनसार-कस्तूरिकातिसौरभ-मेदुर-लघुतर-मधुर-शीतलतर-
सलिलधारा-निराकरिष्णु-तवीय-विमल-विलोचन-मयूख-रेखापसरित-पिपासायास-पथिक-
लोकान्

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lec. 1

So, what is the particular word in Sanskrit that is composed of 431 letters? So, this was from the Varadambika Parinaya Campu by Tirumalamba; this is a single compound from his book.

(Refer Slide Time: 26:51)

Word Tokenization in Chinese or Sanskrit

Also called 'Word Segmentation'.

Greedy Algorithm for Chinese

Maximum Matching (Greedy Algorithm)

- Start a pointer at the beginning of the string
- Find the largest word in dictionary that matches the string starting at pointer
- Move the pointer over the word in string

Think of the cases when word segmentation would be required for English Text.

Finding constituent words in a compound hashtags: #ThankYouSachin, #musicmonday etc.

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 17 / 26

So now, when I talk about this problem of tokenization in Sanskrit or in English this problem is also called word segmentation, have a sequence of characters and you segment it to find out individual words. Now what is the simplest algorithm that you can think off? Let us take as in the case of Chinese. So, the simplest algorithm that works is a

greedy algorithm that is called maximum matching algorithm. So, whenever you are given a string you start you point to it at the beginning of the string. Now suppose that you have the dictionary and the words that you are currently seeing all should be the in the dictionary.

So, you will find out what is the maximum match as per my dictionary in the string, you break there and put the pointer from at the next character and again do the same thing. So, this greedily chooses what are actual words by taking the maximum matches. And this works nicely for most of the cases.

So, this (Refer Time: 27:55), now can you think of some cases where the segmentation will also be required for the English text? In English in general we do not combine words to make a single word. We do not do that, but what is the scenario where we are doing that right now. So does, do hash tags come into mind. For example, suppose I have hash tags like ThankYouSachin, and musicmonday. So, here different words are combined together without putting a boundary in between.

So, if you are given a hash tag and you have to analyze that you have to actually segment it into various words.

(Refer Slide Time: 28:30)

Text Segmentation for Sanskrit

1

General assumption behind the design

Sentences from Classical Sanskrit may be generated by a regular relation R of the Kleene closure W^* of a regular set W of words over a finite alphabet Σ .

- W : vocabulary of (inflected) words (*padas*) and
- R : sandhi

Analysis of a sentence

A candidate sentence w is analyzed by inverting relation R to produce a finite sequence w_1, w_2, \dots, w_n of word forms, together with a proof that $w \in R(w_1 \cdot w_2 \cdot \dots \cdot w_n)$.

¹<http://sanskrit.inria.fr>

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 18 / 26

So, when I talk about Sanskrit, so this we have a segment to available at the site Sanskrit dot inria dot fr. So, we will just briefly see what is the design principle of building a

Now, when I have a set of words I can now combine them together with an operation of Sandi; that is what I mean by sigma star here; so w^* here. So, I have a set of words w and I will do a cleaner closure; that means, I can combine any number of words together, but whenever I am combining words I am doing them by a Sandi operation. This is the relation between the words.

(Refer Slide Time: 30:14)

The screenshot shows the application interface with the sentence: "सत्यम्ब्रूयाप्रियम्ब्रूयान्ब्रूयात्सत्यमप्रियम्ब्रूयचानातृम्ब्रूयावैषधर्मःसनातनः". Below the sentence, the word "Undo (120 Solutions)" is displayed. The segmentation options are shown as a sequence of words: सत्यम्, प्रियम्, त्, सत्यम्, अ, प्रियम्, प्रियम्, सना, तम्, त्, न्, धर्मः, सनातनः. Each word is followed by a red box containing a checkmark or an 'X', indicating the correctness of the segmentation. The words are color-coded: सत्यम् (blue), प्रियम् (green), त् (red), सत्यम् (blue), अ (yellow), प्रियम् (green), प्रियम् (green), सना (blue), तम् (green), त् (red), न् (red), धर्मः (blue), सनातनः (blue).

Now this is a snapshot from the segmentor. So, I gave the same sentence there and it gave me all the possible ways of analyzing the Sandi's. And it says that there are 120

different solutions. So, here whenever I have bruyana, so you see there are two possibilities bruyat and bruyam. That is like that it gives me all the possible ways in which this sentence can be broken into individual word tokens.

Now this is another problem that I will have to find out what is the most likely word sequence among all these 120 possibilities. But we can use many many different models that we will not talk about in this lecture probably in some other lectures.

(Refer Slide Time: 30:52)

Normalization

Why to "normalize"?

Indexed text and query terms must have the same form.

- U.S.A. and USA should be matched
- We implicitly define equivalence classes of terms

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 5 20/26

So coming back to normalization; we talked about this problem that the same word might be doing multiple different ways like U dot S dot A versus USA. Now I should be able to match them together. Especially, if you are doing information retrieval we are giving a query and you are retrieving from some document. Suppose your query contains U dot S dot A if the document contains USA, if you are only doing the surface match you will not be able to map on to each other. So you will have to consider this problem in advance and do the pre processing accordingly of either your documents or the query, but using the same sort same sentence.

So, what we are doing by this? We are defining some sort of equivalence classes. We are saying USA and U dot S dot A should go to one class, and the other same type.

(Refer Slide Time: 31:52)

Case Folding

- Reduce all letters to lower case
- Possible exceptions (Task dependent):
 - ▶ Upper case in mid sentence, may point to named entities (e.g. General Motors)
 - ▶ For MT and information extraction, some cases might be helpful (*US* vs. *us*)

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 1

We also do some sort of case folding that is we can reduce all the letters to lower case. So, whenever I have the word like `w o r d` I will always write small `w o r d`, so that whenever even if it is starting the sentence and it occurs in capitals because of that in general I know that this is a word `w o r d`. But this is not a generic rule sometimes depending on application you might have certain exceptions. For example, you might put treat the name and it is separately. So, if you have entity General Motors you might want to keep it as it is without case folding.

Similarly, you might want to keep `US` for United States in upper case and not do the case folding. And this is important for the application of machine transition also, because if you do a case folding here you will know `u s` in lower case that means something else versus `US` that is in United States; excuse me.

(Refer Slide Time: 32:58)

Lemmatization

- Reduce inflections or variant forms to base form:
 - ▶ am, are, is → be
 - ▶ car, cars, car's, cars' → car
- Have to find the correct dictionary headword form

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lec. 1

We also have the problem of lemmatization; that is you have individual words like am, are, is; and you want to convert them to their lemma; that means, what is the base form from which they are derived. Similarly car, cars, car's cars'; so all these are derived from car. Again this is some sort of normalization we are saying all these are some sort of equivalence class because they come from the same word from.

So, in the problem of lemmatization is that you have to find out the actual dictionary head word from which they have derived.

(Refer Slide Time: 33:32)

Morphology

Morphology studies the internal structure of words, how words are built up from smaller meaningful units called **morphemes**

Morphemes are divided into two categories

- Stems: The core meaning bearing units
- Affixes: Bits and pieces adhering to stems to change their meanings and grammatical functions
 - ▶ Prefix: un-, anti-, etc (a-, ati-, pra- etc.)
 - ▶ Suffix: -ity, -ation, etc (-taa, -ke, -ka etc.)
 - ▶ Infix: 'n' in 'vindati' (he knows), as contrasted with vid (to know).

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lec. 1

And for that we use morphology. So, what is morphology? I am trying to find out the structure of word by seeing what is the particular stem the headword and what is the affix that is applied to it. So, these individual units are called various morphemes.

So, you have a stems that are the (Refer Time: 33:55) hybrids and the affixes that are what are the different units like as for plural etcetera you are applying to them to make the individual word. So, my examples are like for prefix you have un-, anti-, etcetera for English and a-, ati-, pra- etcetera for Hindi or Sanskrit. Suffix like -ity, -ation etcetera and -taa -ka -ke etcetera for Hindi. And in general you can also have some infix, like you have the word like vid and you can infix n in between this is in Sanskrit. So, we will discuss in detail about it in morphology later.

So, there is another concept you have lemmatization where you are finding the actual dictionary headword. So, there is also a concept called stemming where you do not try to find the actual dictionary headword, but you just try to remove certain suffixes and whatever you obtain is called a stem, so this crude chopping of various affixes in that word.

(Refer Slide Time: 34:59)

Stemming

- Reducing terms to their stems, used in information retrieval
- Crude chopping of affixes
 - ▶ language dependent
 - ▶ *automate(s), automatic, automation* all reduced to *automat*

for example compressed and compression are both accepted as equivalent to compress.

→

for example compress and compression are both accepted as equivalent to compress

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lec...

So, this is again language dependent. So, what we are doing here words like automate, automatic, automation all will be reduced to a single lemma automatically. So, this is stemming, so you know the actual lemma is automate with an e, but here I am just

chopping off the affixes at the end. So, I am removing here this ic, ion all and putting it to automate.

So, this is one example; if you try to do a stemming here see you will find from example e is removed, from compressed ed is removed and so on. So, what is the algorithm that is used for this stemming?

(Refer Slide Time: 35:48)

The slide is titled "Porter's algorithm" and is divided into two sections: "Step 1a" and "Step 1b".

Step 1a

- sses → ss (caresses → caress)
- ies → i (ponies → poni)
- ss → ss (caress → caress)
- s → ϕ (cats → cat)

Step 1b

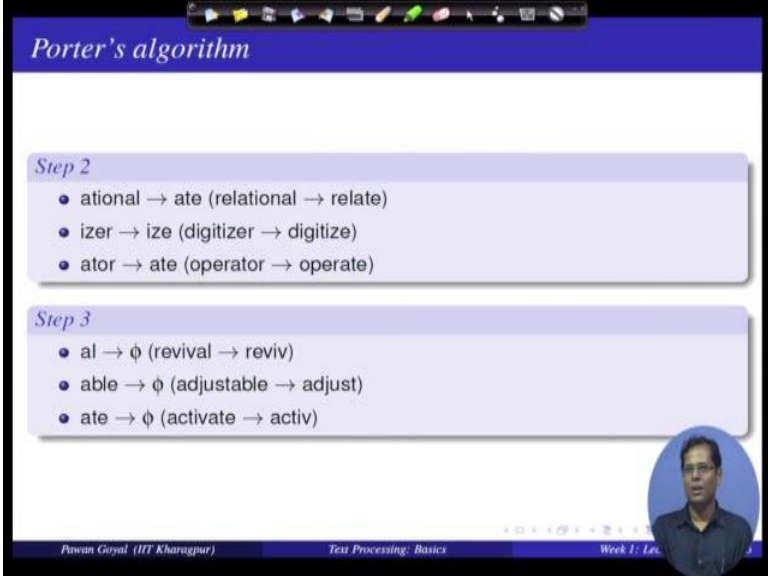
- (*v*)ing → ϕ (walking → walk, king → king)
- (*v*)ed → ϕ (played → play)

At the bottom of the slide, there is a footer with the text "Pawan Goyal (IIT Khargpur)", "Text Processing: Basics", and "Week 1: Lex". A small circular inset image of a man is visible in the bottom right corner of the slide.

So, we have the Porter's algorithm that is very very famous. And this is again some sort of if-then-else rules. So, what are some examples here? What is the first step? I take a word if it ends with sses I remove es from there and I end with ss, so example is caresses goes to caress. If not then I see whether the words end with ies I put it to i - like ponies goes to poni. If not I see if the word ends with ss I keep it as ss, if not I see if the word ends with s I remove that s. Cats goes to cat but caress does not go to caress with only one s because this is step comes before. If there is a double s and in the word I written it otherwise if there is a single s I remove it that.

Like that there are some other steps. So, if there is a vowel in my word and the word ends with ing I remove ing. So, walking goes to walk, but what about king you see in k there is no vowel. So, king will be written as it is. Same is a vowel and there is an ed I remove this ed. And I have this word played to play. So, you can see that what is the use of this heuristic of having this vowel. If you did not have this vowel you would have converted king to k.

(Refer Slide Time: 37:17)



Porter's algorithm

Step 2

- ational → ate (relational → relate)
- izer → ize (digitizer → digitize)
- ator → ate (operator → operate)

Step 3

- al → ϕ (revival → reviv)
- able → ϕ (adjustable → adjust)
- ate → ϕ (activate → activ)

Pawan Goyal (IIT Kharagpur) Text Processing: Basics Week 1: Lecture 1

And like that there are some other ways like if the word ends with ational then I will put it put ate, so rational; so relational to relate. And if the word ends with izer I convert I remove that r digitizer to digitize ator to ate. And if the word ends with al I remove that al, if the word ends with able I remove that able, if the word ends with ate I remove that ate. So, like that these are some steps that I take from my corpus from each word I convert it to its step, it does not give me the correct dictionary headword, but still this is a good practice in principle for information retrieval, if you want to match the query with the documents.

This is for this week. Next week we will start with another pre processing task that is a spelling correction.

Thank you.