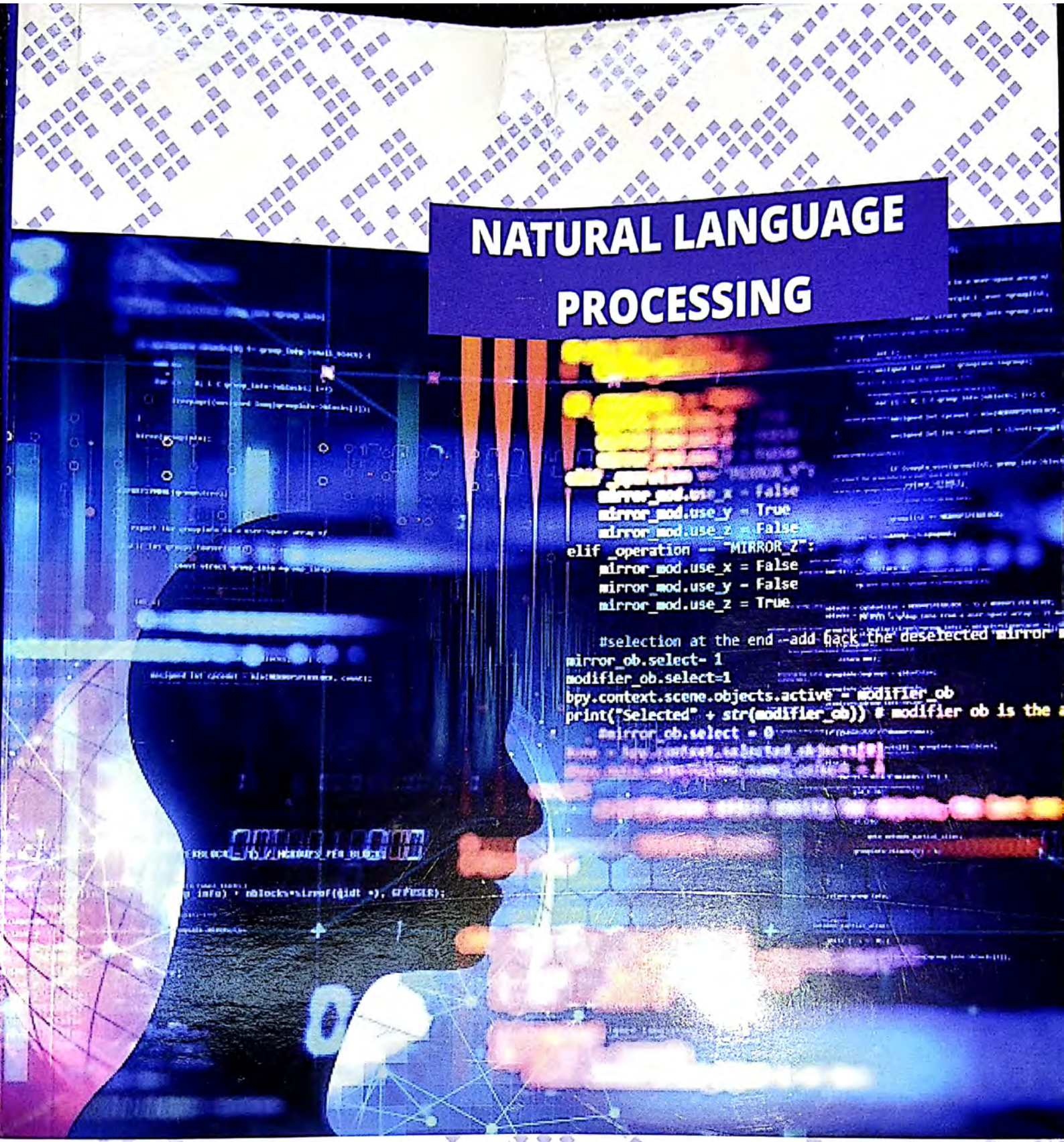
NATURAL LANGUAGE PROCESSING



Sharvari Govilkar
Sagar Kulkarni
Dhiraj Amin

STAREDU SOLUTIONS

www.staredusolutions.org

Contents

Introduction

1

OBJECTIVES

After reading this chapter, the student will be able to Understand:

- Introduction to Natural Language Processing.
- History of NLP .
- Generic NLP Systems .
- Levels of NLP.
- Knowledge in Language processing .
- Ambiguity in NLP .
- Stages in NLP.
- Challenges for NLP.
- Application Areas of NLP:

1. Introduction to Natural Language Processing

Natural Language refers to the language spoken by people, e.g. English, Hindi, Marathi as opposed to artificial/programming languages, like C, C++, Java, etc.

A natural language or ordinary language is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation. Natural languages can take different forms, such as written text, speech or signing etc. They are distinguished from constructed and formal languages such as those used to program computers or to study logic. Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding.

Natural Language Processing (NLP) is a field of research and application that determines the way computers can be used to understand and manage natural language text or speech to do useful things. The term "natural" in the context of the language is used to distinguish human languages (such as Gujarati, English, Spanish and French) from computer languages (such as C, C++, Java and Prolog). The definition of Natural Language Processing clarifies that it is a theoretically induced range of computational techniques (multiple methods or techniques for language analysis) for analyzing and representing naturally occurring text (such as English, Gujarati and Punjabi) at one or more levels of linguistic analysis for the purpose of achieving human like language processing for a range of tasks or applications.

2. Need for Natural Language Processing (NLP)

Significant growth in the volume and variety of data is due to the amount of unstructured text data—in fact, up to 80% of all your data is unstructured text data. Companies collect huge amounts of documents, emails, social media, and other text-based information to get to know their customers better, offer services or market their products .However, most of this data is unused and untouched.

Text analytics, through the use of natural language processing (NLP), holds the key to unlocking the business value within these vast data resources.

In the era of big data, businesses can fully utilize the data potential and take advantage of the latest parallel text analytics and NLP algorithms packaged in a variety of open source software namely R, python etc.

Consider a example given in Figure 1:

```
kJfmmfj mmmvvv nnnffn333
Uj iheale eleee mnster vensi credur
Baboi oi cestnize
Coovoel2^ ekk; ldslik lkdf vnnjfj?
Fgrnglmlik mlfin kfre xnnn!
```

Figure 1: Sample Text in natural form

Computers "see" text in English the same you have seen the figure 1. Normally, People have no trouble understanding natural language as they have Common sense knowledge, Reasoning capacity, Experience for understanding the context of the text. But this is not the case with Computers. Computers don't have inbuilt Common-sense knowledge, Reasoning capacity, Experience. Unless we teach computers to do so, they will not understand any natural language.

3. Goals of Natural Language Processing:

1. The ultimate goal of natural language processing is for computers to achieve human-like comprehension of texts/languages. When this is achieved, computer systems will be able to understand, draw inferences from, summarize, translate and generate accurate and natural human text and language.
2. The goal of natural language processing is to specify a language comprehension and production theory to such a level of detail that a person is able to write a computer program which can understand and produce natural language.
3. The basic goal of NLP is to accomplish human like language processing. The choice of word "processing" is very deliberate and should not be replaced with "understanding". For although the field of NLP was originally referred to as Natural Language Understanding (NLU), that goal has not yet been accomplished. A full NLU system would be able to:
   - Paraphrase an input text.
   - Translate the text into another language.
   - Answer questions about the contents of the text.
   - Draw inferences from the text.

4. Brief overview of NLP

The field of study that focuses on the interactions between human language and computers is called Natural Language Processing, or NLP for short. It sits at the intersection of computer science, artificial intelligence, and computational linguistics

The essence of Natural Language Processing lies in making computers understand the natural language. That's not an easy task though. Computers can understand the structured form of data like spreadsheets and the tables in the database, but human languages, texts, and voices form an unstructured category of data, and it gets difficult for the computer to understand it, and there arises the need for Natural Language Processing.

There's a lot of natural language data out there in various forms and it would get very easy if computers can understand and process that data. We can train the models in accordance with expected output in different ways. Humans have been writing for thousands of years, there are a lot of literature pieces available, and it would be great if we make computers understand that. But the task is never going to be easy. There are various challenges floating out there like understanding the correct meaning of the sentence, correct Named-Entity Recognition(NER), correct prediction of various parts of speech, coreference resolution(the most challenging thing in my opinion).

Computers can't truly understand the human language. If we feed enough data and train a model properly, it can distinguish and try categorizing various parts of speech(noun, verb, adjective, supporters, etc...) based on previously fed data and experiences. If it encounters a new word it tried making the nearest guess which can be embarrassingly wrong a few times.

It's very difficult for a computer to extract the exact meaning from a sentence. For example – The boy radiated fire like vibes. The boy had a very motivating personality or he actually radiated fire? As you can see over here, parsing English with a computer is going to be complicated.

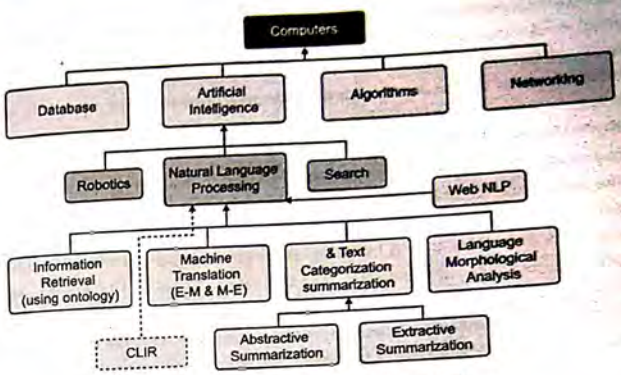


Figure 2: NLP in the Computer science taxonomy

5. History of NLP

NLP began in the 1950s as the intersection of artificial intelligence and linguistics. NLP was originally distinct from text information retrieval (IR), which employs highly scalable statistics-based techniques to index and search large volumes of text efficiently: Manning et al1 provide an excellent introduction to IR. With time, however, NLP and IR have converged somewhat. Currently, NLP borrows from several, very diverse fields, requiring today's NLP researchers and developers to broaden their mental knowledge-base significantly.

Early simplistic approaches, for example, word-for-word Russian-to-English machine translation, were defeated by homographs—identically spelled words with multiple meanings—and metaphor, leading to the apocryphal story of the Biblical, 'the spirit is willing, but the flesh is weak' being translated to 'the vodka is agreeable, but the meat is spoiled.'

Chomsky's 1956 theoretical analysis of language grammars provided an estimate of the problem's difficulty, influencing the creation (1963) of Backus-Naur Form (BNF) notation. BNF is used to specify a 'context-free grammar (CFG), and is commonly used

to represent programming-language syntax. A language's BNF specification is a set of derivation rules that collectively validate program code syntactically. ('Rules' here are absolute constraints, not expert systems' heuristics.) Chomsky also identified still more restrictive 'regular' grammars, the basis of the regular expressions used to specify text-search patterns. Regular expression syntax, defined by Kleene (1956), was first supported by Ken Thompson's grep utility on UNIX.

Subsequently (1970s), lexical-analyzer (lexer) generators and parser generators such as the lex/yacc combination utilized grammars. A lexer transforms text into tokens; a parser validates a token sequence. Lexer/parser generators simplify programming-language implementation greatly by taking regular-expression and BNF specifications, respectively, as input, and generating code and lookup tables that determine lexing/parsing decisions.

While CFGs are theoretically inadequate for natural language, they are often employed for NLP in practice. Programming languages are typically designed deliberately with a restrictive CFG variant, an LALR(1) grammar (LALR, Look-Ahead parser with Left-to-right processing and Rightmost (bottom-up) derivation), to simplify implementation. An LALR(1) parser scans text left-to-right, operates bottom-up (i. e. it builds compound constructs from simpler ones), and uses a look-ahead of a single token to make parsing decisions.

The Prolog language was originally invented (1970) for NLP applications. Its syntax is especially suited for writing grammars, although, in the easiest implementation mode (top-down parsing), rules must be phrased differently (i. e. right-recursively) from those intended for a yacc-style parser. Top-down parsers are easier to implement than bottom-up parsers (they don't need generators), but are much slower.

Recent research has increasingly focused on unsupervised and semi-supervised learning algorithms. Such algorithms are able to learn from data that has not been hand-annotated with the desired answers, or using a combination of annotated and non-annotated data. Generally, this task is much more difficult than supervised learning, and typically produces less accurate results for a given amount of input data. However, there is an enormous amount of non-annotated data available (including, among other things, the entire content of the World Wide Web), which can often make up for the inferior results.

Modern NLP consists of speech recognition, machine learning, machine text reading, and machine translation. These parts when combined would allow for artificial intelligence to gain real knowledge of the world, not just playing chess or moving around an obstacle course. In the near future computers will be able to read all of the information online and learn from it and solve problems and possibly cure diseases. There limit for NLP and AI is humanity, research will not stop until both are at a human level of awareness and understanding.

6. Generic NLP system

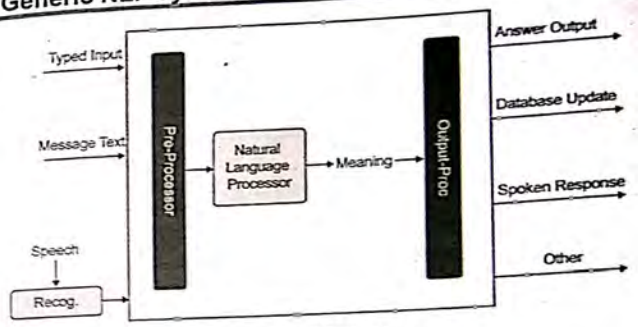
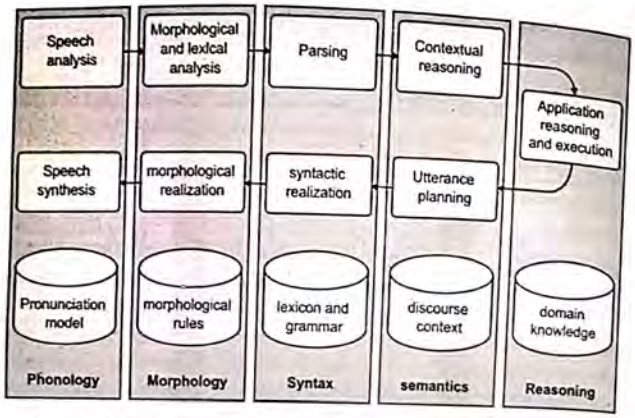


Figure 3: Generic NLP System

Any natural language processing should start with some input and ends with effective and accurate output. The inputs for natural language processor can be text or speech. There are a variety of output that can be generated by the system. Output may be in the form of answer when input is a question. Similarly outputs can be Database update, Spoken response, Semantics, Part of speech, Morphology of word, Semantics of the word/ Sentences etc.

7. Levels of NLP

Natural Language Processing works on multiple levels and most often, these different areas synergize well with each other. The NLP can broadly be divided into various levels as shown in figure.



Phonology: It deals with interpretation of speech sound within and across words.

Morphology: It is a study of the way words are built up from smaller meaning-bearing units called morphemes. For example, the word 'fox' has single morpheme while the word 'cats' have two morphemes 'cat' and morpheme '–s' represents singular and plural concepts.

Morphological lexicon is the list of stem and affixes together with basic information, whether the stem is a Noun stem or a Verb stem [21]. The detailed analysis of this level is discussed in chapter 4. Syntax: It is a study of formal relationships between words. It is a study of: how words are clustered in classes in the form of Part-of-Speech (POS), how they are grouped with their neighbours into phrases, and the way words depend on each other in a sentence.

Semantics: It is a study of the meaning of words that are associated with grammatical structure. It consists of two kinds of approaches: syntax-driven semantic analysis and semantic grammar. The detailed explanation of this level is discussed in chapter 4. In discourse context, the level of NLP works with text longer than a sentence. There are two types of discourse- anaphora resolution and discourse/text structure recognition. Anaphora resolution is replacing of words such as pronouns. Discourse structure recognition determines the function of sentences in the text which adds meaningful representation of the text.

Reasoning: To produce an answer to a question which is not explicitly stored in a database; Natural Language Interface to Database (NLIDB) carries out reasoning based on data stored in the database. For example, consider a database that holds the academic information about student, and user posed a query such as: 'Which student is likely to fail in Maths subject? To answer the query, NLIDB needs a domain expert to narrow down the reasoning process.

8. Knowledge in Language processing

A natural language understanding system must have knowledge about what the words mean, how words combine to form sentences, how word meanings combine to from sentence meanings and so on. The different forms of knowledge required for natural language understanding are given below.

PHONETIC AND PHONOLOGICAL KNOWLEDGE

Phonetics is the study of language at the level of sounds while phonology is the study of combination of sounds into organized units of speech, the formation of syllables and larger units. Phonetic and phonological knowledge are essential for speech based systems as they deal with how words are related to the sounds that realize them.

MORPHOLOGICAL KNOWLEDGE

Morphology concerns word formation. It is a study of the patterns of formation of words by the combination of sounds into minimal distinctive units of meaning called morphemes. Morphological knowledge concerns how words are constructed from morphemes.

SYNTACTIC KNOWLEDGE

Syntax is the level at which we study how words combine to form phrases, phrases combine to form clauses and clauses join to make sentences. Syntactic analysis concerns sentence formation. It deals with how words can be put together to form correct sentences. It also determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.

SEMANTIC KNOWLEDGE

It concerns the meanings of the words and sentences. This is the study of context independent meaning that is the meaning a sentence has, no matter in which context it is used. Defining the meaning of a sentence is very difficult due to the ambiguities involved.

PRAGMATIC KNOWLEDGE

Pragmatics is the extension of the meaning or semantics. Pragmatics deals with the contextual aspects of meaning in particular situations. It concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

DISCOURSE KNOWLEDGE

Discourse concerns connected sentences. It is a study of chunks of language which are bigger than a single sentence. Discourse language concerns inter-sentential links that is how the immediately preceding sentences affect the interpretation of the next sentence. Discourse knowledge is important for interpreting pronouns and temporal aspects of the information conveyed.

WORLD KNOWLEDGE

Word knowledge is nothing but everyday knowledge that all speakers share about the world. It includes the general knowledge about the structure of the world and what each language user must know about the other user's beliefs and goals. This essential to make the language understanding much better.

knowledge representation and reasoning systems have incorporated natural language as interfaces to expert systems or knowledge bases that performed tasks separate from natural language processing. As this book shows, however, the computational nature of representation and inference in natural language makes it the ideal model for all tasks in an intelligent computer system. Natural language processing combines the qualitative characteristics of human knowledge processing with a computer's quantitative advantages, allowing for an in-depth, systematic processing of vast amounts of information. The essays in this interdisciplinary book cover a range of implementations and designs, from formal computational models to large-scale natural language processing systems.

9. Ambiguity in NLP

Natural Language Processing (NLP) is an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things. The Text based NLP has been regarded as consisting of various levels.

They are:

- Lexical Analysis:- Analysis of word forms
- Syntactic Analysis:-Structure processing
- Semantic Analysis:- Meaning representation
- Discourse Analysis:- Processing of interrelated sentences
- Pragmatic Analysis:-The purposeful use of sentences in situations.

Ambiguity can occur at all these levels. It is a property of linguistic expressions. If an expression (word/phrase/sentence) has more than one interpretation we can refer it as ambiguous.

For e.g: Consider the sentence, **"The chicken is ready to eat".**

The interpretations in the above phrase can be:

- The chicken(bird) is ready to be fed or
- The chicken (food) is ready to be eaten.

Consider another sentence: **"There was not a single man at the party"**

The interpretations in this case can be:

- Lack of bachelors at the party or
- Lack of men altogether

There are different types of ambiguities

1. **Lexical Ambiguity:** is the ambiguity of a single word. A word can be ambiguous with respect to its syntactic class. Eg: book, study.

   For eg: The word **"silver"** can be used as a noun, an adjective, or a verb.

   - She bagged two silver medals.
   - She made a silver speech.
   - His worries had silvered his hair.

   Lexical ambiguity can be resolved by Lexical category disambiguation i.e. parts-of-speech tagging. As many words may belong to more than one lexical category part-of-speech tagging is the process of assigning a part-of-speech or lexical category such as a noun, verb, pronoun, preposition, adverb, adjective etc. to each word in a sentence.

   Lexical Semantic Ambiguity: The type of lexical ambiguity, which occurs when a single word is associated with multiple senses. Eg: bank, pen, fast, bat, cricket etc.

   For eg: 1. The tank was full of water.
   2. I saw a military tank.

Words have multiple meanings for such sentences. Consider the sentence **"I saw a bat."**

Possible meaning of the words which changes the context of the sentence are:

- bat = flying mammal / wooden club?
- saw = past tense of "see" / present tense of "saw" (to cut with a saw.)

The occurrence of tank in both sentences corresponds to the syntactic category noun, but their meanings are different. Lexical Semantic ambiguity resolved using word sense disambiguation (WSD) techniques,where WSD aims at automatically assigning the meaning of the word in the context in a computational manner.

2. **Syntactic Ambiguity:** The structural ambiguities were syntactic ambiguities. Structural ambiguity is of two kinds: Scope Ambiguity and Attachment Ambiguity.
   - **Scope Ambiguity:** Scope ambiguity involves operators and quantifiers.

     Consider the example: Old men and women were taken to safe locations.

     The scope of the adjective (i.e., the amount of text it qualifies) is ambiguous. That is,whether the structure (old men and women) or ((old men) and women)?

     The scope of quantifiers is often not clear and creates ambiguity.

     Every man loves a woman

     The interpretations can be, For every man there is a woman and also it can be there is one particular woman who is loved by every man.
   - **Attachment Ambiguity**

     A sentence has attachment ambiguity if a constituent fits more than one position in a parse tree. Attachment ambiguity arises from uncertainty of attaching a phrase or clause to a part of a sentence.

     Consider the example:

     The man saw the girl with the telescope.

     It is ambiguous whether the man saw a girl carrying a telescope, or he saw her through his telescope.

     The meaning is dependent on whether the preposition 'with' is attached to the girl or the man.

     Consider the example:

     Buy books for children

     Preposition Phrase 'for children' can be either adverbial and attach to the verb buy or adjectival and attach to the object noun books.

3. **Semantic Ambiguity:** This occurs when the meaning of the words themselves can be misinterpreted. Even after the syntax and the meanings of the individual words have been resolved, there are two ways of reading the sentence.

   Consider the example: **"Seema loves her mother and Sriya does too"**

   The interpretations can be Sriya loves Seema's mother or Sriya likes her own mother.

   Semantic ambiguities born from the fact that generally a computer is not in a position to distinguishing what is logical from what is not.

   Consider the example: **"The car hit the pole while it was moving".**

   The interpretations can be:
   - The car, while moving, hit the pole
   - The car hit the pole while the pole was moving.

   The first interpretation is preferred than the second one because we have a model of the world that helps us to distinguish what is logical (or possible) from what is not. To supply to a computer model of the world is not so easy.

   Consider the example: **"We saw his duck"**

   Duck can refer to the person's bird or to a motion he made.

   Semantic ambiguity happens when a sentence contains an ambiguous word or phrase.

4. **Discourse Ambiguity:** Discourse level processing needs a shared world or shared knowledge and the interpretation is carried out using this context. Anaphoric ambiguity comes under discourse level.
   - **Anaphoric Ambiguity:** Anaphora's are the entities that have been previously introduced into the discourse.

     Consider the example, **The horse ran up the hill. It was very steep. It soon got tired.**

     The anaphoric reference of 'it' in the two situations cause ambiguity. Steep applies to surface hence 'it' can be hill. Tired applies to animate object hence 'it' can be horse.
5. **Pragmatic Ambiguity:** Pragmatic ambiguity refers to a situation where the context of a phrase gives it multiple interpretation. One of the hardest tasks in NLP. The problem involves processing user intention, sentiment, belief world, modals etc. all of which are highly complex tasks.

Consider the example,

"Tourist (checking out of the hotel): Waiter, go upstairs to myroom and see if my sandals are there; do not be late; I have to catch the train in 15 minutes."

Waiter (running upstairs and coming back panting): Yes sir, they are there.

Clearly, the waiter is falling short of the expectation of the tourist, since he does not understand the pragmatics of the situation.

Pragmatic ambiguity arises when the statement is not specific, and the context does not provide the information needed to clarify the statement. Information is missing, and must be inferred. Consider the example: **"I love you too."**

This can be interpreted as:

- I love you (just like you love me)
- I love you (just like someone else does)
- I love you (and I love someone else)
- I love you (as well as liking you)

It is a highly complex task to resolve all these kinds of ambiguities, especially in the upper levels of NLP. The meaning of a word, phrase, or sentence cannot be understood in isolation and contextual knowledge is needed to interpret the meaning. pragmatic and world knowledge is required in higher levels. It is not easy to create a world model for disambiguation tasks. Linguistic tools and lexical resources are needed for the development of disambiguation techniques. Resourceless languages are lagging behind in these fields compared to resourceful languages in implementation of these techniques. Rule based methods are language specific where as stochastic or statistical methods are language independent. Automatic resolution of all these ambiguities contains several long standing problems but again development towards full-fledged disambiguation techniques is required which takes care of all the ambiguities. It is very much necessary for the accurate working of NLP applications such as Machine Translation, Information Retrieval, Question Answering etc.

Statistical Approaches of Ambiguity Resolution in Natural Language Processing are:

1. Probabilistic model
2. Part of Speech Tagging
   - Rule-Based Approaches
   - Markov Model Approaches
   - Maximum Entropy Approaches
   - HMM-Based Taggers
3. Machine Learning Approaches

10. Stages of NLP

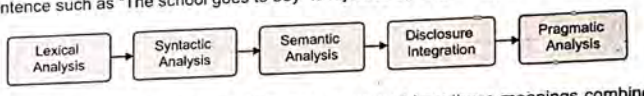
There are general five steps in natural language processing

Lexical Analysis: It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of text into paragraphs, sentences, and words.

The lexical analysis in NLP deals with the study at the level of words with respect to their lexical meaning and part-of-speech. This level of linguistic processing utilizes a language's *lexicon*, which is a collection of individual *lexemes*. A lexeme is a basic unit of lexical meaning; which is an abstract unit of morphological analysis that represents the set of forms or "senses" taken by a single morphemes.

"Duck", for example, can take the form of a noun or a verb but its part-of-speech and lexical meaning can only be derived in context with other words used in the phrase/sentence. This, in fact, is an early step towards a more sophisticated Information Retrieval system where precision is improved through part-of-speech tagging.

Syntactic Analysis (Parsing): It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as "The school goes to boy" is rejected by English syntactic analyzer.



Semantic Analysis: It concerns what words mean and how these meanings combine in sentences to form sentence meanings. It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain. The semantic analyzer disregards sentence such as "hot ice-cream". Another example can be (plant : industrial plant/ living organism)

Discourse Integration: This concerns how the immediately preceding sentences affect the interpretation of the next sentence. The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of the immediately succeeding sentence.

Pragmatic Analysis: This concerns how sentences are used in different situations and how it affects the interpretation of the sentence. During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

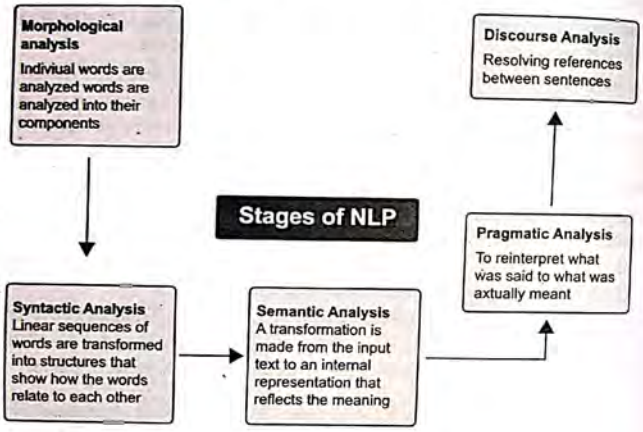
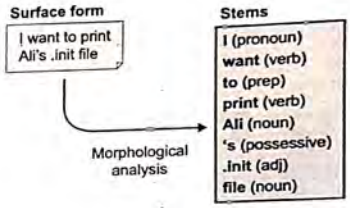


Figure 4: Stages of NLP

Morphological Analysis:

The morphological level of linguistic processing deals with the study of word structures and word formation, focusing on the analysis of the individual components of words. The most important unit of morphology, defined as having the "minimal unit of meaning" is referred to as the *morphemes*. For example, the word: *"unhappiness"*. It can be broken down into three morphemes (prefix, stem, and suffix), with each conveying some form of meaning: the prefix *un-* refers to "not being", while the suffix *-ness* refers to "a state of being". The stem *happy* is considered as a *free morphemes* since it is a "word" in its own right. *Bound morphemes* (prefixes and suffixes) require a free morphemes to which it can be attached to, and can therefore not appear as a "word" on their own.

In Information Retrieval, document and query terms can be stemmed to match the morphological variants of terms between the documents and query; such that the singular form of a noun in a query will match even with its plural form in the document, and vice versa, thereby increasing recall.

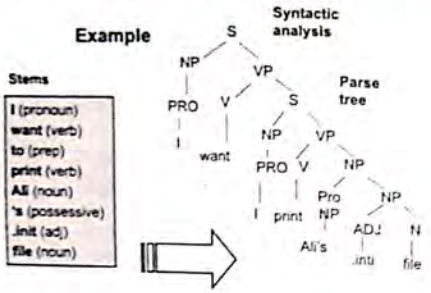


Syntactic Analysis

The part-of-speech tagging output of the lexical analysis can be used at the syntactic level of linguistic processing to group words into phrase and clause brackets. Syntactic Analysis also referred to as *"parsing"*, allows the extraction of phrases which convey more meaning than just the individual words by themselves, such as in a noun phrase.

In Information Retrieval, parsing can be leveraged to improve indexing since phrases can be used as representations of documents which provide better information than just single-word indices. In the same way, phrases that are syntactically derived from the query offers better search keys to match with documents that are similarly parsed.

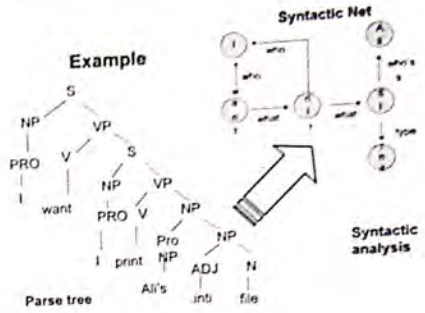
Nevertheless, syntax can still be ambiguous at times as in the case of the news headline *"Boy paralyzed after tumor fights back to gain black belt"* — which actually refers to how a boy was paralyzed because of a tumor but endured the fight against the disease and ultimately gained a high level of competence in martial arts.



Semantic Analysis

The semantic level of linguistic processing deals with the determination of what a sentence really means by relating syntactic features and disambiguating words with multiple definitions to the given context. This level entails the appropriate interpretation of the meaning of sentences, rather than the analysis at the level or individual words or phrases.

In Information Retrieval, the query and document matching process can be performed on a conceptual level, as opposed to simple terms, thereby further increasing system precision. Moreover, by applying semantic analysis to the query, term expansion would be possible with the use of lexical sources, offering improved retrieval of the relevant documents even if exact terms are not used in the query. Precision may increase with query expansion, as with recall probably increasing as well.

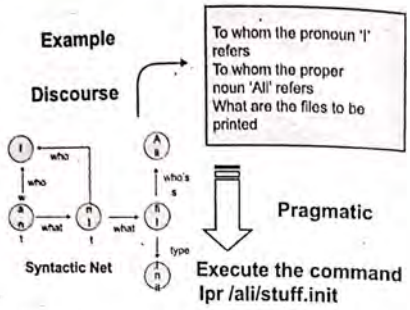


Pragmatic Analysis

The pragmatic level of linguistic processing deals with the use of real-world knowledge and understanding how this impacts the meaning of what is being communicated. By analyzing the contextual dimension of the documents and queries, a more detailed representation is derived.

In Information Retrieval, this level of Natural Language Processing primarily engages query processing and understanding by integrating the user's history and goals as well as the context upon which the query is being made. Contexts may include time and location.

This level of analysis enables major breakthroughs in Information Retrieval as it facilitates the conversation between the IR system and the users, allowing the elicitation of the purpose upon which the information being sought is planned to be used, thereby ensuring that the information retrieval system is fit for purpose.



Discourse Analysis

The discourse level of linguistic processing deals with the analysis of structure and meaning of text beyond a single sentence, making connections between words and sentences. At this level, Anaphora Resolution is also achieved by identifying the entity referenced by an anaphor (most commonly in the form of, but not limited to, a pronoun). An example is shown below.

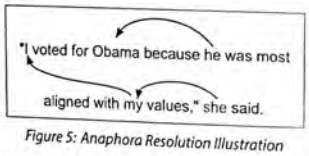


Figure 5: Anaphora Resolution Illustration

With the capability to recognize and resolve anaphora relationships, document and query representations are improved, since, at the lexical level, the implicit presence of concepts is accounted for throughout the document as well as in the query, while at the semantic and discourse levels, an integrated content representation of the documents and queries are generated.

Structured documents also benefit from the analysis at the discourse level since sections can be broken down into (1) title, (2) abstract, (3) introduction, (4) body, (5) results, (6) analysis, (7) conclusion, and (8) references. Information Retrieval systems are significantly improved, as the specific roles of pieces of information are determined as for whether it is a conclusion, an opinion, a prediction, or a fact.

11. Applications of NLP

In the context of Human Computer Interface (HCI), there are many NLP applications such as information retrieval systems, information extraction, machine learning systems, question answering system, dialogue system, email routing, telephone banking, speech recognition system, documentation retrieval system, document summarization, discourse management, multilingual query processing, and natural language interface to database system. Currently interactive applications may be classified into following categories:

Speech Recognition / Speech Understanding and Synthesis / Speech Generation: Speech understanding system attempts to perform a semantic and pragmatic processing of spoken utterance to understand what the user is saying and act on what is being said. The research area in this category includes: linguistic analysis, design & developing efficient and effective algorithms for speech recognition and synthesis.

Language Translator: It is a task of automatically converting one natural language into another preserving the meaning of input text and producing an equivalent text in the output language. The research area in this category includes language modelling.

Information Retrieval (IR): It is a scientific discipline that deals with analysis, design and implementation of a computerized system that addresses representation, organization, and access to large amounts of heterogeneous information encoded in digital format. The search engine is the well known application of IR which accepts query from user and returns the relevant document to user. It returns the document, not the relevant answers; users are left to extract answers from the returned documents. The research area in IR includes: information searching, information extraction, information categorization and information summarization from unstructured information.

Information Extraction: It includes extraction of structured information from unstructured text. It is an activity of filling predefined template from natural language text. The research area in this category includes identifying named entity, resolving anaphora and identifying relationships between entities.

Question Answering (QA): It is passage retrieval in specific domain. It is a process of finding answers for a given question from a large collection of documents.

Natural Language Interface to Database (NLIDB): It is a process of finding answers from database by asking questions in natural language.

Dialog Systems: It is a study of dialog between human and computers. It determines grammar and style of the sentence based on that it gives response to users. The research area in this category includes the design of conventional agent, human-robot dialog and analysis of human-human dialog.

Discourse Management / Story Understanding / Text Generation: The task of identifying the discourse structure is to identify the nature of discourse relationship between sentences such as elaboration, explanation, contrast and also to classify speech acts in a chunk of text (For example, yes-no, statement and assertion).

Expected Questions

1. What is Natural language processing (NLP) ? Discuss various stages involved in NLP process with suitable example.
2. What is Natural Language Understanding? Discuss various levels of analysis under it with example.
3. What do you mean by ambiguity in Natural language? Explain with suitable example. Discuss various ways to resolve ambiguity in NL.
4. What do mean by lexical ambiguity and syntactic ambiguity in Natural language? What are different ways to resolve these ambiguities?
5. List various applications of NLP and discuss any 2 applications in detail.

Word Level Analysis

2

OBJECTIVES

After reading this chapter, the student will be able to understand:

- Morphology analysis
- Stemming and Lemmatization
- Regular expression
- Finite Automata
- Finite-State Morphological Parsing
- Combining FST Lexicon and Rules
- Lexicon free FST Porter stemmer
- N –Grams, N-gram language model, N-gram for spelling correction.

1. Morphology analysis

What are words?

Words are the fundamental building block of language. Every human language, spoken, signed, or written, is composed of words. Every area of speech and language processing, from speech recognition to machine translation to information retrieval on the web, requires extensive knowledge about words. Psycholinguistic models of human language processing and models from generative linguistic are also heavily based on lexical knowledge.

Words are Orthographic tokens separated by white space. In some languages the distinction between words and sentences is less clear.

Chinese, Japanese: no white space between words

nowhitespace → no white space/no whites pace/now hit esp ace

Turkish: words could represent a complete "sentence"

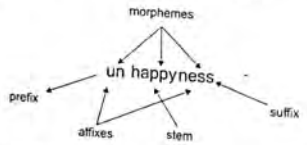
Eg: uygarlastiramadiklarimizdanmissinizcasina

"(behaving) as if you are among those whom we could not civilize"

Morphology deals with the syntax of complex words and parts of words, also called morphemes, as well as with the semantics of their lexical meanings. Understanding how words are formed and what semantic properties they convey through their forms enables human beings to easily recognize individual words and their meanings in discourse. Morphology also looks at parts of speech, intent and stress, and the ways context can change a word's pronunciation and meaning.

Morphology is the study of the structure and formation of words. Its most important unit is the morpheme, which is defined as the "minimal unit of meaning"

Consider a word like: "unhappiness". This has three parts:



Morphemes : un happy ness

Prefix : un

Suffix : ness

Affixes : happy

Stem : happy

There are three morphemes, each carrying a certain amount of meaning. UN means "not", while ness means "being in a state or condition". Happy is a free morpheme because it can appear on its own (as a "word" in its own right). Bound morphemes have to be attached to a free morpheme, and so cannot be words in their own right. Thus, you can't have sentences in English such as "Jason feels very un ness today".

Bound Morphemes: These are lexical items incorporated into a word as a dependent part. They cannot stand alone, but must be connected to another morphemes. Bound morphemes operate in the connection processes by means of derivation, inflection, and compounding. Free morphemes, on the other hand, are autonomous, can occur on their own and are thus also words at the same time. Technically, bound morphemes and free morphemes are said to differ in terms of their distribution or freedom of occurrence. As a rule, lexemes consist of at least one free morpheme

Morphology handles the formation of words by using morphemes base form (stem,lemma), e.g., believe affixes (suffixes, prefixes, infixes), e.g., un-, -able, -ly

Morphological parsing is the task of recognizing the morphemes inside a word e.g., hands, foxes, children and its important for many tasks like machine translation, information retrieval, etc. and useful in parsing, text simplification, etc

Survey of English Morphology

Morphology is the study of the way words are built up from smaller meaning- bearing units, morphemes. A morpheme is often defined as the minimal meaning-bearing unit in a language. So for example the word fox consists of a single morpheme (the morpheme fox) while the word cats consists of two: the morpheme cat and the morpheme -s. As this example suggests, it is often useful to distinguish two broad classes of morphemes: stems and affixes. The exact details of the distinction vary from language to language, but intuitively, the stem is the 'main' morpheme of the word, supplying the main meaning, while the affixes add 'additional' meanings of various kinds. Affixes are further divided into prefixes, suffixes, infixes, and circum- fixes. Prefixes precede the stem, suffixes follow the stem, circumfixes do both, and infixes are inserted inside the stem. For example, the word eats is composed of a stem eat and the suffix -s. The word unbuckle is composed of a stem buckle and the prefix un-. English doesn't have any good examples of circumfixes,

but many other languages do. In German, for example, the past participle of some verbs formed by adding ge- to the beginning of the stem and -t to the end; so the past participle of the verb sagen (to say) is gesagt (said). Infixes, in which a morpheme is inserted in the middle of a word, occur very commonly for example in the Philipine language Tagalog. For example, the affix um, which marks the agent of an action, is infixed to the Tagalog stem hingi 'borrow' to produce humingi.

Prefixes and suffixes are often called concatenative morphology since a word is composed of a number of morphemes concatenated together. A number of languages have extensive non-concatenative morphology, in which morphemes are combined in more complex ways. The Tagalog in- fixation example above is one example of non-concatenative morphology, since two morphemes (hingi and um) are intermingled. Another kind of non-concatenative morphology is called templatic morphology or root-and-pattern morphology. This is very common in Arabic, Hebrew, and other Semitic languages. In Hebrew, for example, a verb is constructed us- ing two components: a root, consisting usually of three consonants (CCC) and carrying the basic meaning, and a template, which gives the ordering of consonants and vowels and specifies more semantic information about the resulting verb, such as the semantic voice (e.g. active, passive, middle). For example the Hebrew tri-consonantal root lmd, meaning 'learn' or 'study', can be combined with the active voice CaCaC template to produce the word lamad, 'he studied', or the intensive CiCeC template to produce the word limed, 'he taught', or the intensive passive template CuCaC to produce the word lumad, 'he was taught'.

A word can have more than one affix. For example, the word rewrites have the prefix re-, the stem write, and the suffix -s. The word unbelievably has a stem (believe) plus three affixes (un-, -able, and -ly). While English doesn't tend to stack more than 4 or 5 affixes, languages like Turkish can have words with 9 or 10 affixes, as we saw above. Languages that tend to string affixes together like Turkish does are called agglutinative languages.

There are two broad (and partially overlapping) classes of ways to form words from morphemes: inflection and derivation. Inflection is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem, and usually filling some syntac- tic function like agreement. For example, English has the inflectional mor- pheme -s for marking the plural on nouns, and the inflectional morphemes -ed for marking the past tense on verbs. Derivation is the combination of a

word stem with a grammatical morpheme, usually resulting in a word of a different class, often with a meaning hard to predict exactly. For example, the verb computerizes can take the derivational suffix -ation to produce the noun computerization.

Inflectional morphology & Derivational morphology

Morphemes are defined as smallest meaning-bearing units. Morphemes can be classified in various ways. One common classification is to separate those morphemes that mark the grammatical forms of words (-s, -ed, -ing and others) from those that form new lexemes conveying new meanings, e.g. un- and -ment. The former morphemes are inflectional morphemes and form a key part of grammar, the latter are derivational morphemes and play a role in word-formation, as we have seen. The following criteria help you to distinguish the two types:

- Effect: Inflectional morphemes encode grammatical categories and relations, thus marking word-forms, while derivational morphemes create new lexemes.
- Position: Derivational morphemes are closer to the stem than inflectional morphemes, cf. amendments (amend[stem] – ment[derivational] – s[inflectional]) and legalized (legal[stem] – ize[derivational] – ed[inflectional]).
- Productivity: Inflectional morphemes are highly productive, which means that they can be attached to the vast majority of the members of a given class (say, verbs, nouns or adjectives), whereas derivational morphemes tend to be more restricted with regard to their scope of application. For example, the past morpheme can in principle be attached to all verbs; suffixation by means of the adjective-forming derivational morphemes -able, however, is largely restricted to dynamic transitive verbs, which excludes formations such as *bleedable or *lieable.
- Class properties: Inflectional morphemes make up a closed and fairly stable class of items which can be listed exhaustively, while derivational morphemes tend to be much more numerable and more open to changes in their inventory.

Both inflectional and derivational morphemes must be attached to other morphemes; they cannot occur by themselves, in isolation, and are therefore known as bound morphemes

Inflected words are variations of already existing lexemes that were only changed in their grammatical shape. Therefore many of the Inflectional Morphemes are not listed in the dictionary. If you know the word surprise and look it up you will also find in the same entry the word surprise -s which simply expresses the plural.

Derivational Morphology on the other hand uses affixes to create new words out of already existing lexemes. Typical affixes are -ness, -ish, -ship and so on. These affixes do not change the grammatical form of a word such as inflectional affixes do, but instead often create a new meaning of the base or change the word class of the base. An Example would be the word light. The plural form light-s would be considered Inflectional Morphology, but if we consider de-light the prefix -de has changed the meaning of the word completely. We now do not think of light in the form of sunshine or lamps anymore but instead about a feeling. Also if we consider en-light the suffix -en has changed the word class of light from noun to verb.

INFLECTIONAL MORPHOLOGY

Inflection is a morphological process that adapts existing words so that they function effectively in sentences without changing the category of the base morphemes.

Inflection can be seen as the "realization of morphosyntactic features through morphological means" . But what exactly does that mean? It means that inflectional morphemes of a word do not have to be listed in a dictionary since we can guess their meaning from the root word. We know when we see the word what it connects to and most times can even guess the difference to its original. For example let us consider help-s, help-ed and help-er. According to what I have said about words listed in the dictionary, all of these variants might be inflectional morphemes, but then on the other hand does help-s really need an extra listing or can we guess from the root help and the suffix -s what it means? Does our natural feeling and instinct for language not tell us, that the suffix –s indicates the third person singular and that help-s therefore only is a variant from help (considering help as a verb and not a noun here)? Yes it does. As native speaker one instantly knows that –s, as also the past form indicator –ed only show a grammatical variant of the root lexeme help.

So why is help-er or even help-less-ness different? The answer is actually very simple. The suffixes in these last two words change the word class and therefore form a new lexeme. Help-er can now be the new root for additional suffixes such help-er-s which would then be an inflectional morpheme again, the root here being the smallest free morpheme after you remove all affixes.

After establishing this we still have a problem if we consider the word help as a noun and as a verb. How do we distinguish these two? The answer is context and the phenomenon is called a zero morphemes. Only threw context can we say if help is a verb or a noun.

To illustrate this consider the following two sentences:

1. I help my grandmother in her garden.
2. He is my grandmother's help.

Here our general knowledge of words and their meaning shows us that in 1. help is used as a verb and expresses us working with our grandmother in order to support her. In 2. help is a noun and stands for the person that regularly supports my grandmother. This variation of a word without actually changing its form is called zero morphemes and cannot only distinguish verb and noun (which makes it a derivational morphemes) but also singular and plural, which makes it an inflectional morpheme. I will talk about this later in 2.2.: Inflection in nouns, though.

"We may define inflectional morphology as the branch of morphology that deals with paradigms. It is therefore concerned with two things: on the one hand, with the semantic oppositions among categories; and on the other, with the formal means, including inflections, that distinguish them." (Matthews, 1991).

Inflectional morphology is that it changes the word form, it determines the grammar and it does not form a new lexeme but rather a variant of a lexeme that does not need its own entry in the dictionary.

word stem + grammatical morphemes

cat + s only for nouns, verbs, and some adjectives

Nouns

- plural:
  - regular: +s, +es irregular: mouse - mice; ox - oxen
  - many spelling rules: e.g. -y -> -ies like: butterfly - butterflies
- possessive: +'s, +'

Verbs

- main verbs (sleep, eat, walk)
- modal verbs (can, will, should)
- primary verbs (be, have, do)

VERB INFLECTIONAL SUFFIXES

1. The suffix –s functions in the Present Simple as the third person marking of the verb : to work – he work-s
2. The suffix –ed functions in the past simple as the past tense marker in regular verbs: to love – lov-ed

3. The suffixes –ed (regular verbs) and –en (for some regular verbs) function in the marking of the past partciple and, in general, in the marking of the perfect aspect: To study studied studied / To eat ate eaten
4. The suffix –ing functions in the marking of the present participle, the gerund and in the marking of the continuous aspect: To eat – eating / To study - studying

NOUN INFLECTIONAL SUFFIXES

1. The suffix –s functions in the marking of the plural of nouns: dog – dogs
2. The suffix –s functions as a possessive marker (saxon genitive): Laura – Laura's book.

ADJECTIVE INFLECTIONAL SUFFIXES

The suffix –er functions as comparative marker: quick – quicker

The suffix –est functions as superlative marker: quick - quickest

DERIVATIONAL MORPHOLOGY

Derivation is concerned with the way morphemes are connected to existing lexical forms as affixes. Derivational morphology is a type of word formation that creates new lexemes, either by changing syntactic category or by adding substantial new meaning (or both) to a free or bound base. Derivation may be contrasted with inflection on the one hand or with compounding on the other. The distinctions between derivation and inflection and between derivation and compounding, however, are not always clear-cut. New words may be derived by a variety of formal means including affixation, reduplication, internal modification of various sorts, subtraction, and conversion. Affixation is best attested cross-linguistically, especially prefixation and suffixation. Reduplication is also widely found, with various internal changes like ablaut and root and pattern derivation less common. Derived words may fit into a number of semantic categories. For nouns, event and result, personal and participant, collective and abstract noun are frequent. For verbs, causative and applicative categories are well-attested, as are relational and qualitative derivations for adjectives. Languages frequently also have ways of deriving negatives, relational words, and evaluative. Most languages have derivation of some sort, although there are languages that rely more heavily on compounding than on derivation to build their lexical stock. A number of topics have dominated the theoretical literature on derivation, including productivity (the extent to which new words can be created with a given affix or morphological process), the principles that determine the ordering of affixes, and the place of derivational morphology with respect to other components of the grammar. The

study of derivation has also been important in a number of psycholinguistic debates concerning the perception and production of language.

Derivational morphology is defined as morphology that creates new lexemes, either by changing the syntactic category (part of speech) of a base or by adding substantial, non-grammatical meaning or both. On the one hand, derivation may be distinguished from inflectional morphology, which typically does not change category but rather modifies lexemes to fit into various syntactic contexts; inflection typically expresses distinctions like number, case, tense, aspect, person, among others. On the other hand, derivation may be distinguished from compounding, which also creates new lexemes, but by combining two or more bases rather than by affixation, reduplication, subtraction, or internal modification of various sorts. Although the distinctions are generally useful, in practice applying them is not always easy.

We can distinguish affixes in two principal types:

1. Prefixes - attached at the beginning of a lexical item or base-morpheme – ex: un-, pre-, post-, dis, im-, etc.
2. Suffixes – attached at the end of a lexical item ex: -age, -ing, -ful, -able, -ness, -hood, -ly, etc.

EXAMPLES OF MORPHOLOGICAL DERIVATION

a. Lexical item (free morpheme): like (verb) + prefix (bound morpheme) dis- = dislike (verb)	b. Lexical item: like + suffix –able = likeable + prefix un- =unlikeable + suffix –ness = unlikeableness

Derivational affixes can cause semantic change:

1. Prefix pre- means before; post- means after; un- means not, re- means again.
2. Prefix = fixed before; Unhappy = not happy = sad; Retell = tell again.
3. Prefix de- added to a verb conveys a sense of subtraction; dis- and un- have a sense of negativity.
4. To decompose; to defame; to uncover; to discover.

Derivation Versus Inflection

The distinction between derivation and inflection is a functional one rather than a formal one, as Booij (2000, p. 360) has pointed out. Either derivation or inflection may be affected by formal means like affixation, reduplication, internal modification of bases, and other morphological processes. But derivation serves to create new lexemes while inflection prototypically serves to modify lexemes to fit different grammatical contexts. In the clearest cases, derivation changes category, for example taking a verb like *employ* and making it a noun (*employment, employer, employee*) or an adjective (*employable*), or taking a noun like *union* and making it a verb (*unionize*) or an adjective (*unionish, unionesque*). Derivation need not change category, however. For example, the creation of abstract nouns from concrete ones in English (*king ~ kingdom; child ~ childhood*) is a matter of derivation, as is the creation of names for trees (*poirier* 'pear tree') from the corresponding fruit (*poire* 'pear') in French, even though neither process changes category. Derivational prefixation in English tends not to change category, but it does add substantial new meaning, for example creating negatives (*unhappy, inconsequential*), various quantitative or relational forms (*tricycle, preschool, submarine*) or evaluative (*megastore, miniskirt*). Inflection typically adds grammatical information about number (singular, dual, plural), person (first, second, third), tense (past, future), aspect (perfective, imperfective, habitual), and case (nominative, accusative), among other grammatical categories that languages might mark.

Nevertheless, there are instances that are difficult to categorize, or that seem to fall somewhere between derivation and inflection. Many of the difficult cases hinge on determining the necessary and sufficient criteria in defining derivation. Certainly, category change is not necessary, as there are many obvious cases of derivation that do not involve category change. But further, Haspelmath (1996) has argued that there are cases of inflection (participles of various sorts, for example) that are category-changing, so that this criterion cannot even be said to be sufficient. Productivity has also been used as a criterion—inflection typically being completely productive, derivation being only sporadically productive—but some derivation is just as productive as inflection, for example, forming nouns from adjectives with *-ness* in English, and some inflection displays lexical gaps (Booij, 2000, p. 363 says, e.g., that the Dutch infinitive *bloemlezen* 'to make an anthology' does not correspond to any finite forms), so productivity is neither necessary nor sufficient for distinguishing inflection and derivation. Anderson (1982) makes the criterion of relevance to syntax the most important one for distinguishing inflection from derivation; inflection is invariably relevant to syntax, derivation not. But Booij (1996) has argued that even this criterion is problematic unless we are clear what we mean by relevance to syntax. Case inflections, for example, mark grammatical context, and are therefore clearly inflectional. Number-marking on verbs is arguably inflectional when it is triggered by the number of subject or object, but number on nouns or tense and aspect on verbs is a matter of semantic choice, independent of grammatical configuration. Booij therefore distinguishes what he calls contextual inflection, inflection triggered by distinctions elsewhere in a sentence, from inherent inflection, inflection that does not depend on the syntactic context, the latter being closer to derivation than the former. Some theorists (Bybee, 1985; Dressler, 1989) postulate a continuum from derivation to inflection, with no clear dividing line between them. Another position is that of Scalise (1984), who has argued that evaluative morphology is neither inflectional nor derivational but rather constitutes a third category of morphology.

2. Stemming and Lemmatization

In natural language processing, there may come a time when you want your program to recognize that the words "ask" and "asked" are just different tenses of the same verb. This is the idea of reducing different forms of a word to a core root. Words that are derived from one another can be mapped to a central word or symbol, especially if they have the same core meaning.

Maybe this is in an information retrieval setting and you want to boost your algorithm's recall. Or perhaps you are trying to analyze word usage in a corpus and wish to condense related words so that you don't have as much variability. Either way, this technique of text normalization may be useful to you.

This is where something like stemming or lemmatization comes in.

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 words with 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

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. Stemming is definitely the simpler of the two approaches. With stemming, words are reduced to their word stems. A word stem need not be the same root as a dictionary-based morphological root, it just is equal to or smaller form of the word.

Stemming algorithms are typically rule-based. You can view them as heuristic process that sort-of lops off the ends of words. A word is looked at and run through a series of conditionals that determine how to cut it down.

For example, we may have a suffix rule that, based on a list of known suffixes, cuts them off. In the English language, we have suffixes like "-ed" and "-ing" which may be useful to cut off in order to map the words "cook," "cooking," and "cooked" all to the same stem of "cook."

Over stemming comes from when too much of a word is cut off. This can result in nonsensical stems, where all the meaning of the word is lost or muddled. Or it can result in words being resolved to the same stems, even though they probably should not be.

Take the four words university, universal, universities, and the universe. A stemming algorithm that resolves these four words to the stem "univers" has over stemmed. While it might be nice to have universal and universe stemmed together and university and universities stemmed together, all four do not fit. A better resolution might have the first two resolve to "univers" and the latter two resolve to "universi." But enforcing rules that make that so might result in more issues arising.

Under stemming is the opposite issue. It comes from when we have several words that actually are forms of one another. It would be nice for them to all resolve to the same stem, but unfortunately, they do not. This can be seen if we have a stemming algorithm that stems the words data and datum to "dat" and "datu."

Stemming Algorithms Examples

Porter stemmer: This stemming algorithm is an older one. It's from the 1980s and its main concern is removing the common endings to words so that they can be resolved to a common form. It's not too complex and development on it is frozen. Typically, it's a nice starting basic stemmer, but it's not really advised to use it for any production/ complex application. Instead, it has its place in research as a nice, basic stemming algorithm that can guarantee reproducibility. It also is a very gentle stemming algorithm when compared to others.

Snowball stemmer: This algorithm is also known as the Porter2 stemming algorithm. It is almost universally accepted as better than the Porter stemmer, even being acknowledged as such by the individual who created the Porter stemmer. That being said, it is also more aggressive than the Porter stemmer. A lot of the things added to the Snowball stemmer were because of issues noticed with the Porter stemmer. There is about a 5% difference in the way that Snowball stems versus Porter.

Lancaster stemmer: Just for fun, the Lancaster stemming algorithm is another algorithm that you can use. This one is the most aggressive stemming algorithm of the bunch. However, if you use the stemmer in NLTK, you can add your own custom rules to this algorithm very easily. It's a good choice for that. One complaint around this stemming algorithm though is that it sometimes is overly aggressive and can really transform words into strange stems. Just make sure it does what you want it to before you go with this option!

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* . lemmatization is a more calculated process. It involves resolving words to their dictionary form. In fact, a lemma of a word is its dictionary or canonical form!

Because lemmatization is more nuanced in this respect, it requires a little more to actually make work. For lemmatization to resolve a word to its lemma, it needs to know its part of speech. That requires extra computational linguistics power such as a part of speech tagger. This allows it to do better resolutions (like resolving is and are to "be").

Another thing to note about lemmatization is that it's often times harder to create a lemmatizer in a new language than it is a stemming algorithm. Because lemmatizers

require a lot more knowledge about the structure of a language, it's a much more intensive process than just trying to set up a heuristic stemming algorithm.

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.

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:

(F)	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.

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, Hindi and Finnish).

3. Regular expression

Regular expression (RE), a language for specifying text search strings or search pattern. The regular expression languages used for searching texts in UNIX (vi, Perl, Emacs, grep) and Microsoft. Usually search patterns are used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory.

Words are almost identical, and many RE features exist in the various Web search engines. Besides this practical use, the regular expression is an important theoretical tool throughout computer science and linguistics. A regular expression is a formula in a special language that is used for specifying simple classes of strings. A string is a sequence of symbols; for the purpose of most text-based search techniques, a string is a sequence of alphanumeric characters (letters, numbers, spaces, tabs, and punctuation). For these purposes a space is just a character like any other, and we represent it with the symbol ␣. Formally, a regular expression is an algebraic notation for characterizing a set of strings. Thus, they can be used to specify search strings as well as to define a language in a formal way.

Basically, a regular expression is a pattern describing a certain amount of text. Their name comes from the mathematical theory on which they are based. But we will not dig into that. You will usually find the name abbreviated to "regex" or "regexp". Regular expressions (regex or regexp) are extremely useful in extracting information from any text by searching for one or more matches of a specific search pattern (i.e. a specific sequence of ASCII or unicode characters).

```
Natural Language Processing, or [redacted] for short, is broadly defined as the
automatic manipulation of natural language, like speech and text, by
software.Statistical [redacted] aims to do statistical inference for the field of
natural language.
```

```
/\b(\w*NLP\w*)\b/g                                                    2 matches
```

Figure 1: shows matching of the string NLP in the given text on the site https://www.regextester.com/

Basic Regular Expression Patterns

Anchors — ^ and $

^The matches any string that **starts with The -> Try it!**

end$ matches a string that **ends with end**

^The end$ **exact string match** (starts and ends with **The end**)

roar matches any string that **has the text roar in it**

Quantifiers — * + ? and {}

abc* matches a string that has **ab followed by zero or more c**

abc+ matches a string that has **ab followed by one or more c**

abc? matches a string that has **ab followed by zero or one c**

abc{2} matches a string that has **ab followed by 2 c**

abc{2,} matches a string that has **ab followed by 2 or more c**

abc{2,5} matches a string that has **ab followed by 2 up to 5 c**

a(bc)* matches a string that has **a followed by zero or more copies of the sequence bc**

a(bc){2,5} matches a string that has **a followed by 2 up to 5 copies of the sequence bc**

OR operator — | or []

a(b|c) matches a string that has **a followed by b or c**

a[bc] same as previous

Character classes — \d \w \s and .

\d matches a **single character** that is a **digit**

\w matches a **word character** (alphanumeric character plus underscore)

\s matches a **whitespace character** (includes tabs and line breaks)

. matches **any character**

\d, \w and \s also present their negations with \D, \W and \S respectively.

For example, \D will perform the inverse match with respect to that obtained with \d.

\D matches a **single non-digit character**

In order to be taken literally, you must escape the characters ^.[$()|*+?{\with a backslash \ as they have special meaning.

\$\d matches a string that has a **$ before one digit**

We can match also non-printable characters like tabs \t, new-lines \n, carriage returns \r

Flags

We are learning how to construct a regex but forgetting a fundamental concept: **flags**.

A regex usually comes within this form */abc/*, where the search pattern is delimited by two slash characters /. At the end we can specify a flag with these values (we can also combine them each other):

- **g** (global) does not return after the first match, restarting the subsequent searches from the end of the previous match
- **m** (multi-line) when enabled ^ and $ will match the start and end of a line, instead of the whole string
- **i** (insensitive) makes the whole expression case-insensitive (for instance **/aBc/i** would match **AbC**)

Grouping and capturing — ()

a(bc) parentheses create a **capturing group with value bc**

a(?:bc)* using **?:** we **disable the capturing group**

a(?<foo>bc) using **?<foo>** we put a name to the group

This operator is very useful when we need to extract information from strings or data using your preferred programming language. Any multiple occurrences captured by

several groups will be exposed in the form of a classical array: we will access their values specifying using an index on the result of the match.

If we choose to put a name to the groups (using (?<foo>...)) we will be able to retrieve the group values using the match result like a dictionary where the keys will be the name of each group.

Bracket expressions—[]

[abc] matches a string that has **either an a or a b or a c** -> is the **same as a|b|c**

[a-c] matches a string that has **either an a or a b or a c** -> is the **same as a|b|c**

[a-fA-F0-9] a string that represents **a single hexadecimal digit, case insensitively**

[0-9]% a string that has a character **from 0 to 9 before a % sign**

[^a-zA-Z] a string that has **not a letter from a to z or from A to Z.** In this case the **^** is used as **negation of the expression**

Greedy and Lazy match

The quantifiers (* + {}) are greedy operators, so they expand the match as far as they can through the provided text.

For example, <.+> matches simple div in This is a simple div test. In order to catch only the div tag we can use a ? to make it lazy:

<.+?> matches **any character one or more** times included **inside <** and **>, expanding as needed**

Boundaries — \b and \B

\babc\b performs a **"whole words only" search**

\b represents an anchor like caret (it is similar to $ and ^) matching positions where one side is a word character (like \w) and the other side is not a word character (for instance it may be the beginning of the string or a space character).

It comes with its negation, \B. This matches all positions where \b doesn't match and could be if we want to find a search pattern fully surrounded by word characters.

\Babc\B matches only if the pattern is **fully surrounded by word** characters

Look-ahead and Look-behind — (?=) and (?<=)

d(?=r) matches a **d** only if is **followed by r, but r will not be** part of the overall regex **match**

(?<=r)d matches a **d** only if is **preceded by an r, but r will not be** part of the overall regex **match**

4. Finite Automata

The regular expression is more than just a convenient metalanguage for text searching. First, a regular expression is one way of describing a finite-state automaton (FSA). Finite-state automata are the theoretical foundation of a good deal of the computational work we will describe in this book, Any FSA regular expression can be implemented as a finite-state automaton (except regular expressions that use the memory feature; more on this later). Symmetrically, any finite-state automaton can be described with a regular expression. Second, a regular expression is one way of characterizing a particular kind of formal language called a regular language. Both regular expressions and finite-state automata can be used to describe regular languages. The relation among these three theoretical constructions is sketched out in the figure below.

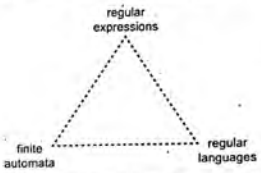


Figure 2: The relationship between finite automata, regular expressions,and regular languages

A formal language is completely determined by the 'words in the dictionary', rather than by any grammatical rules

A (formal) ***language*** L over alphabet Σ is just a set of strings in Σ^*.

Thus any subset $L \subseteq \Sigma^*$ determines a language over Σ

The ***languaage determined by a regular expression*** r over Σ is

$$L(r) \underline{\text{def}} \{w \in \Sigma^* | w \text{ matches } r\}.$$

Two regular expressions *r* and *s* (over the same alphabet) are ***equivalent*** iff *L*(r) and *L*(*s*) are equal sets (i.e. have exactly the same members.)

A finite automaton has a finite set of states with which it accepts or rejects strings.

Finite State Automata (FSA) can be:

Deterministic

On each input there is one and only one state to which the automaton can transition from its current state

Nondeterministic

An automaton can be in several states at once

Deterministic finite state automaton

1. A finite set of states, often denoted Q
2. A finite set of input symbols, often denoted Σ
3. A transition function that takes as arguments a state and an input symbol and returns a state. The transition function is commonly denoted δ If q is a state and a is a symbol, then δ(q, a) is a state p (and in the graph that represents the automaton there is an arc from q to p labeled a)
4. A start state, one of the states in Q
5. A set of final or accepting states F ($F \subseteq Q$)

A DFA is a tuple A = (Q, Σ, δ, q_0, F)

Other notations for DFAs

Transition diagrams

- Each state is a node
- For each state $q \in Q$ and each symbol $a \in \Sigma$, let $\delta(q, a) = p$
- Then the transition diagram has an arc from q to p, labeled a
- There is an arrow to the start state q_0
- Nodes corresponding to final states are marked with doubled circle

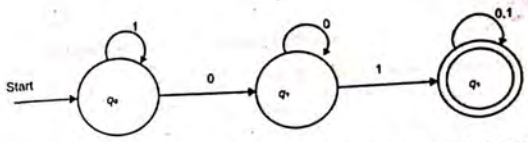
Transition tables

- Tabular representation of a function
- The rows correspond to the states and the columns to the input s
- The entry for the row corresponding to state q and the column corresponding to input a is the state δ(q, a)

A = ({ q_0, q_1, q_2 }, { 0, 1 }, δ, q_0, { q_1 })

where the transition function δ is given by the table

		0	1
→	q_0	q_2	q_0
*	q_1	q_1	q_1
	q_2	q_2	q_1



The DFA define a language: the set of all strings that result in a sequence of state transitions from the start state to an accepting state.

Extended transition function

Describes what happens when we start in any state and follow any sequence of inputs

If δ is our transition function, then the extended transition function is denoted by δ

The extended transition function is a function that takes a state q and a string w and returns a state p (the state that the automaton reaches when starting in state q and processing the sequence of inputs w)

Formal definition of the extended transition function

Definition by induction on the length of the input string

Basis: $\delta(q, \epsilon) = q$

If we are in a state q and read no inputs, then we are still in state q Induction: Suppose w is a string of the form xa; that is a is the last symbol of w, and x is the string consisting of all but the last symbol

Then: $\delta(q, w) = \delta(\delta(q,x), a)$

To compute $\delta(q, w)$, first compute $\delta(q,x)$, the state that the automation is in after procesing all but the last symbol of *w*

Suppose this state is *p*, *i.e.*, $\delta(q,x) = p$

Then $\delta(q, w)$ is what we get by making a transition from state *p* on input *a* - the last symbol of *w*

Design a DFA to accept the language

L = {w | w has both an even number of 0 and an even number of 1}

The Language of a DFA

The language of a DFA A = (Q, Σ, δ, qo, F), denoted $L(A)$ is defined by

$$L(A) = \{w \mid \delta(qo, w) \text{ is in } F\}$$

The language of A is the set of strings w that take the start state qo to the one of the accepting states .

If L is a $L(A)$ from some DFA, then L is a regular language

Nondeterministic Finite Automata (NFA)

A NFA has the power to be in several states at once. This ability is often expressed as an ability to "guess" something about its input . Each NFA accepts a language that is also accepted by some DFA. NFA are often more succinct and easier than DFAs . We can always convert an NFA to a DFA, but the latter may have exponentially more states than the NFA (a rare case) .The difference between the DFA and the NFA is the type of transition function δ

For a NFA

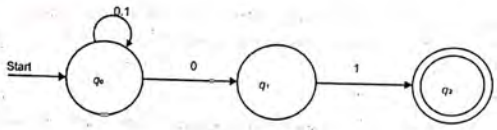
δ is a function that takes a state and input symbol as arguments

(like the DFA transition function), but returns a set of zero or more states

(rather than returning exactly one state, as the DFA must)

Example: An NFA accepting strings that end in 01

Nondeterministic automaton that accepts all and only the strings of 0s and 1s that end in 01



NFA: Formal definition

A nondeterministic finite automaton (NFA) is a tuple A = (Q, Σ, δ, q_0, F) where:

1. Q is a finite set of states
2. Σ is a finite set of input symbols
3. $q_0 \in Q$ is the start state
4. F ($F \subseteq Q$) is the set of final or accepting states
5. δ, the transition function is a function that takes a state in Q and an input symbol in Δ as arguments and returns a subset of Q

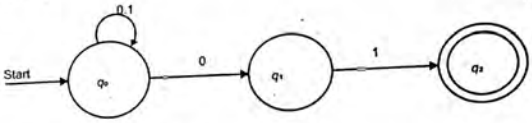
The only difference between a NFA and a DFA is in the type of value that δ returns

Example: An NFA accepting strings that end in 01

$A = (\{q0, q1, q2\}, \{0,1\}, \delta, q0, \{q2\})$

where the transition function δ is given by the table

		0	1
→	$q0$	$\{q0, q1\}$	$\{q0\}$
	$q1$	θ	$\{q2\}$
★	$q2$	θ	θ



The Extended Transition Function

Basics: : $\hat{\delta}(q, \varrho) = \{q\}$

Without reading any input symbols, we are only in the state we began in

Induction

Suppose w is a string of the form xa; that is a is the last symbol of w, and x is the string consisting of all but the last symbol

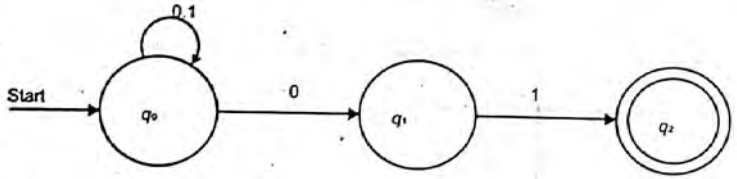
Also suppose that $\delta, (q, x) = \{P1, P2, ... Pk,\}$

Let $\bigcup_{i=1}^{k} \delta(Pi, a) = \{r_1, r_2, ...r_n\}$

Then: $\delta(q, w) = \{r_1, r_2, ...r_n\}$

We compute $\delta^{\wedge}(q, w)$ by first computing $\delta^{\wedge}(q, x)$ and by then following any transition from any of these states that is labeled a

Example: An NFA accepting strings that end in 01



Processing $w = 00101$

1. $\delta(q0, \epsilon) = \{q0\}$
2. $\delta(q0, 0) = \delta(q0, 0) = \{q0, q1\}$
3. $\delta(q0, 00) = \delta(q0, 0) \cup \delta(q1, 0) = \{q0, q1\} \cup \theta = \{q0, q1\}$
4. $\delta(q0, 001) = \delta(q0, 1) \cup \delta(q1, 1) = \{q0\} \cup \{q2\}\ \{q0, q2\}$
5. $\delta(q0, 0010) = \delta(q0, 0) \cup \delta(q2, 0) = \{q0, q1\} \cup \theta = \{q0, q1\}$
6. $\delta(q0, 00101) = \delta(q0, 1) \cup \delta(q1, 1) = \{q0\} \cup \{q2\} = \{q0, q2\}$

The Language of a NFA

The language of a NFA A = (Q, Σ, δ, q_0, F), denoted L(A) is defined by

$L(A) \cong \{w \mid \delta(q0, w) \cap F \neq \theta\}$

The language of A is the set of strings $w \in \Sigma^*$ such that $\delta^{\wedge}(q_0, w)$ contains at least one accepting state.

The fact that choosing using the input symbols of w lead to a non-accepting state, or do not lead to any state at all, does not prevent w from being accepted by a NFA as a whole.

Equivalence of Deterministic and Nondeterministic Finite Automata

Every language that can be described by some NFA can also be described by some DFA. The DFA in practice has about as many states as the NFA, although it has more transitions. In the worst case, the smallest DFA can have 2^n (for a smallest NFA with n state)

5. Finite-State Morphological Parsing

Consider a simple example: parsing just the productive nominal plural (-s) and the verbal progressive (-ing). Our goal will be to take input forms like those in the first column below and produce output forms like those in the second column.

Input	Morphological Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
goose	(goose +N +SG) or (goose +V)
gooses	goose +V 3SG
merging	merge +V +PRES – PART
caught	(catch +V +PAST – PART) or (catch +V +PAST)

The second column contains the stem of each word as well as assorted morphological features. These features specify additional information about the stem. For example the feature +N means that the word is a noun; +SG means it is singular, +PL that it is plural. consider +SG to be a primitive unit that means 'singular'. Note that some of the input forms (like caught or goose) will be ambiguous between different morphological parses.

In order to build a morphological parser, we'll need at least the following:

1. a lexicon: The list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc).
2. morphotactics: the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the rule that the English plural morpheme follows the noun rather than preceding it.

3. orthographic rules: these spelling rules are used to model the changes that occur in a word, usually when two morphemes combine (for example the y ! ie spelling rule discussed above that changes city + -s to cities rather than cities).

6. Building a Finite-State Lexicon

A lexicon is a repository for words. The simplest possible lexicon would consist of an explicit list of every word of the language (every word, i.e. including abbreviations ('AAA') and proper names ('Jane' or 'Beijing') as follows: a, AAA, AA, Aachen, aardvark, aardwolf, aba, abaca, aback

Since it will often be inconvenient or impossible, for the various rea- sons we discussed above, to list every word in the language, computational lexicons are usually structured with a list of each of the stems and affixes of the language together with a representation of the morphotactics that tells us how they can fit together. There are many ways to model morphotactics; one of the most common is the finite-state automaton. A very simple finite-state model for English nominal inflection might look like Figure 3.

The FSA in Figure 3 assumes that the lexicon includes regular nouns (reg-noun) that take the regular -s plural (e.g. cat, dog, fox, aardvark). These are the vast majority of English nouns since for now we will ignore the fact that the plural of words like fox have an inserted e: foxes. The lexicon also includes irregular noun forms that don't take -s, both singular irreg-sg-noun (goose, mouse) and plural irreg-pl-noun (geese, mice).

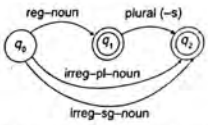


Figure 3: A finite-state automaton for English nominal inflection.

reg-noun	irreg-pi-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
dog	mice	mouse	
aardvark			

A similar model for English verbal inflection might look like Figure 4.

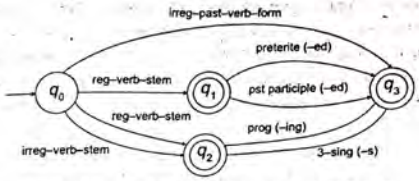


Figure 4: A finite-state automaton for English verbal inflection

This lexicon has three stem classes (reg-verb-stem, irreg-verb-stem, and irreg-past-verb-form), plus 4 more affix classes (-ed past, -ed participle, -ing participle, and 3rd singular -s):

reg-verb-stem	irreg-verb-stem	irreg-past-verb	past	past-part	pres-part	3sg
walk	cut	caught	-ed	-ed	-ing	-s
fry	speak	ate				
talk	sing	eaten				
impeach	sang					
	cut					
	spoken					

English derivational morphology is significantly more complex than English inflectional morphology, and so automata for modelling English derivation tend to be quite complex. Some models of English derivation, in fact, are based on the more complex context-free grammars

As a preliminary example, though, of the kind of analysis it would require, we present a small part of the morphotactics of English adjectives, taken from Antworth (1990). Antworth offers the following data on English adjectives:

big, bigger, biggest
cool, cooler, coolest, coolly
red, redder, reddest
clear, clearer, clearest, clearly, unclear, unclearly
happy, happier, happiest, happily
unhappy, unhappier, unhappiest, unhappily
real, unreal, really

An initial hypothesis might be that adjectives can have an optional pre- fix (un-), an obligatory root (big, cool, etc) and an optional suffix (-er, -est, or -ly). This might suggest the FSA in Figure 5. Alas, while this FSA will recognize all the adjectives in the table above, it will also recognize ungrammatical forms like unbig, redly, and realest. We need to set up classes of roots and specify which can occur with which suffixes. So adj-root1 would include adjectives that can occur with un- and -ly (clear, happy, and real) while adj-root2 will include adjectives that can't (big, cool, and red). Antworth (1990) presents Figure 6 as a partial solution to these problems. This gives an idea of the complexity to be expected from English derivation.

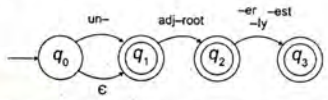


Figure 5: An FSA for a fragment of English adjective morphology: Antworth's Proposal #1.

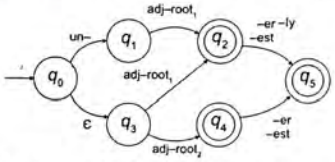


Figure 6: An FSA for a fragment of English adjective morphology: Antworth's Proposal #2

This gives an idea of the complexity to be expected from English derivation. For a further example, we give in Figure 7 another fragment of an FSA for English nominal and verbal derivational morphology, based on Sproat (1993), Bauer (1983), and Porter (1980). This FSA models a number of derivational facts, such as the well-known generalization that any verb ending in -ize can be followed by the nominalizing suffix -ation (Bauer, 1983; Sproat, 1993)). Thus since there is a word fossilize, we can predict the word fossilization by following states q0, q1, and q2. Similarly, adjectives ending in -al or -able at q5 (equal, formal, realizable) can take the suffix -ity, or sometimes the suffix -ness to state q6 (naturalness, casualness).

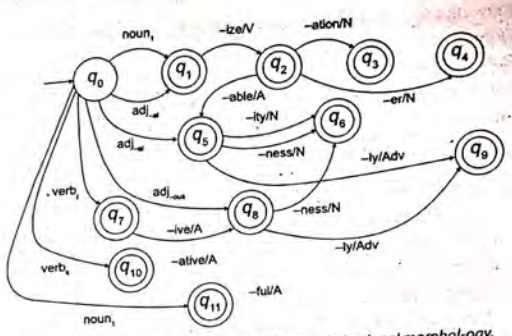


Figure 7: An FSA for another fragment of English derivational morphol-ogy.

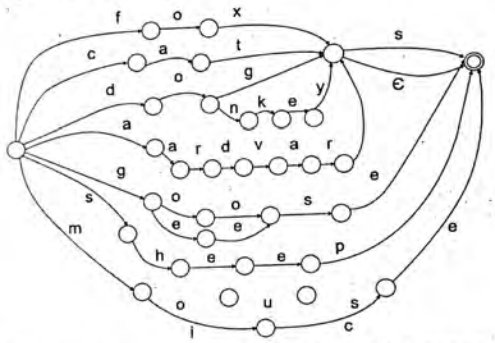


Figure 8: Compiled FSA for a few English nouns with their inflection. Note that this automaton will incorrectly accept the input foxs.

Figure 8 shows the noun-recognition FSA produced by expanding the Nominal Inflection FSA of Figure 9 with sample regular and irregular nouns for each class. We can use Figure 8 to recognize strings like aardvarks by simply starting at the initial state, and comparing the input letter by letter with each word on each outgoing arc, etc.,

Finite-State Transducers

We've now seen that FSAs can represent the morphotactic structure of a lexicon, and can be used for word recognition. A transducer maps between one representation and another; a finite-state transducer or FST is a type of finite automaton which maps between two sets of symbols. We can visualize an FST as a two-tape automaton which recognizes or generates pairs of strings. Intuitively, we can do this by labeling each arc in the finite-state machine with two symbol strings, one from each tape. Fig. 9 shows an example of an FST where each arc is labeled by an input and output string, separated by a colon.

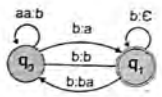


Figure 9: A finite-state transducer

The FST thus has a more general function than an FSA; where an FSA defines a formal language by defining a set of strings, an FST defines a relation between sets of strings. Another way of looking at an FST is as a machine that reads one string and generates another. Here's a summary of this four-fold way of thinking about transducers:

- FST as recognizer: a transducer that takes a pair of strings as input and outputs accept if the string-pair is in the string-pair language, and reject if it is not.
- FST as generator: a machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.
- FST as translator: a machine that reads a string and outputs another string.
- FST as set relater: a machine that computes relations between sets.

All of these have applications in speech and language processing. For morphological parsing (and for many other NLP applications), we will apply the FST as translator metaphor, taking as input a string of letters and producing as output a string of morphemes.

An FST can be formally defined in a number of ways; we will rely on the following definition, based on what is called the Mealy machine MEALY MACHINE extension to a simple FSA:

Q	a finite set of states $q0, q1, \ldots, qn-1$
Σ	a finite set corresponding to the input alphabet
Δ	a finite set corresponding to the output alphabet
$q_0 \in Q$	the start state
$F \subseteq Q$	the set of final states
$\delta(q, w)$	the transition function or transition matrix between states; Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\delta(q, w)$ returns a set of new states $Q' \in Q$. δ is thus a function from $Q \times \Sigma^*$ to 2^Q (because there are 2^Q possible subsets of Q). δ returns a set of states rather than a single state because a given input may be ambiguous in which state it maps to.
$\sigma(q, w)$	the output function giving the set of possible output strings for each state and input. Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\sigma(q, w)$ gives a set of output strings, each a string $o \in \Delta^*$. σ is thus a function from $Q \times \Sigma^*$ to 2^{Δ^*}.

Where FSAs are isomorphic to regular languages, FSTs are isomorphic to regular relations. Regular relations are sets of pairs of strings, a natural extension of the regular languages, which are sets of strings. Like FSAs and regular languages, FSTs and regular relations are closed under union, although in general they are not closed under difference, complementation and intersection (although some useful subclasses of FSTs are closed under these operations; in general, FSTs that are not augmented with the ε are more likely to have such closure properties). Besides union, FSTs have two additional closure properties that turn out to be extremely useful: inversion: The inversion of a transducer T (T^{-1}) simply switches the input and output labels. Thus, if T maps from the input alphabet I to the output alphabet O, T^{-1} maps from O to I.

composition: If T1 is a transducer from I1 to O1 and T2 a transducer from O1 to O2, then T1 ∘ T2 maps from I1 to O2.

Inversion is useful because it makes it easy to convert a FST-as-parser into an FST-as-generator. Composition is useful because it allows us to take two transducers that run in series and replace them with one more complex transducer. Composition works as in algebra; applying T1 ∘ T2 to an input sequence S is identical to applying T1 to S and then T2 to the result; thus T1 ∘ T2(S) = T2(T1(S)).

Fig. 10, for example, shows the composition of [a:b]+ with [b:c]+ to produce [a:c]+.

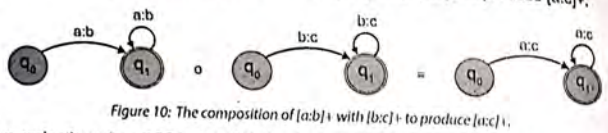


Figure 10: The composition of [a:b]+ with [b:c]+ to produce [a:c]+.

The projection of an FST is the FSA that is produced by extracting only one side of the relation. We can refer to the projection to the left or upper side of the relation as the upper or first projection and the projection to the lower or right side of the relation as the lower or second projection.

Morphological Parsing with Finite-State Transducers

Let's now turn to the task of morphological parsing. Given the input cats, for instance, we'd like to output cat +N +Pl, telling us that cat is a plural noun. Given the Spanish input bebo ('I drink'), we'd like beber +V +PInd +1P +Sg, telling us that bebo is the present indicative first-person singular form of the Spanish verb beber, 'to drink'. In the finite-state morphology paradigm that we will use, we represent a word as a correspondence between a lexical level, which represents a concatenation of mor- Surface phemes making up a word, and the surface level, which represents the concatenation of letters which make up the actual spelling of the word. Fig. 11 shows these two levels for (English) cats.

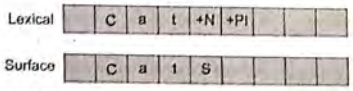


Figure 11: Schematic examples of the lexical and surface tapes;

The actual transducers will involve intermediate tapes as well.

For finite-state morphology it's convenient to view an FST as having two tapes. The upper or lexical tape, is composed from characters from one alphabet Σ. The lower or surface tape, is composed of characters from another alphabet Δ. In the two-level morphology of

Koskenniemi (1983), we allow each arc only to have a single symbol from each alphabet. We can then combine the two symbol alphabets Σ and Δ to create a new alphabet, Σ', which makes the relationship to FSAs quite clear. Σ' is a finite alphabet of complex symbols. Each complex symbol is composed of an input output pair i : o; one symbol i from the input alphabet Σ, and one symbol o from an output alphabet Δ, thus $\Sigma' \subseteq \Sigma \times \Delta$. Σ and Δ may each also include the epsilon symbol ε. Thus, where an FSA accepts a language stated over a finite alphabet of single symbols,

such as the alphabet of our sheep language:

(3.2) Σ = {b,a,!}

an FST defined this way accepts a language stated over pairs of symbols, as in:

(3.3) Σ' = {a : a, b : b, ! :!, a :!, a : ε, ε :!}

In two-level morphology, the pairs of symbols in Σ' Feasible pair are also called feasible pairs. Thus each symbol a : b in the transducer alphabet Σ'expresses how the symbol a from one tape is mapped to the symbol b on the other tape. For example a : ǫ means that an a on the upper tape will correspond to nothing on the lower tape. Just as for an FSA, we can write regular expressions in the complex alphabet Σ'. Since it's most common for symbols to map to themselves, in two-level morphology we call pairs like a : a default pairs, and just refer to them by the single letter a. We are now ready to build an FST morphological parser out of our earlier morphotactic FSAs and lexica by adding an extra "lexical" tape and the appropriate morphological features. Fig. 12 shows an augmentation of Fig. 13 with the nominal morphological features (+Sg and +Pl) that correspond to each morpheme. The symbol ^ indicates a morpheme boundary, while the symbol # indicates a word boundary. The morphological features map to the empty string ǫ or the boundary symbols since there is no segment corresponding to them on the output tape.

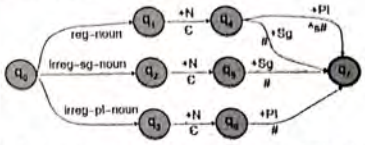


Figure 12: A schematic transducer for English nominal number inflection Tnum.

The sym- bols above each arc represent elements of the morphological parse in the lexical tape; the symbols below each arc represent the surface tape (or the intermediate

lexical, intermediate, and surface. So for example we could write an E-insertion rule that performs the mapping from the intermediate to surface levels shown in Figure 15.

Figure 15: An example of the lexical, intermediate and surface tapes.

Between each pair of tapes is a 2-level transducer; the lexical transducer between the lexical and intermediate levels, and the E-insertion spelling rule between the intermediate and surface levels. The E-insertion spelling rule inserts an e on the surface tape when the intermediate tape has a morpheme boundary ^ followed by the morpheme

Such a rule might say something like "insert an e on the surface tape just when the lexical tape has a morpheme ending in x (or z, etc) and the next morpheme is -s.

$$\varepsilon \rightarrow e / \begin{Bmatrix} x \\ s \\ z \end{Bmatrix} \hat{} \text{-}s\# \qquad \text{(T..1)}$$

This is the rule notation of Chomsky and Halle (1968); a rule of the form a -> b/ c__d means 'rewrite a as b when it occurs between c and d'. Since the symbol ε means an empty transition, replacing it means inserting something. The symbol ^ indicates a morpheme boundary. These boundaries are deleted by including the symbol ^:ε in the default pairs for the transducer; thus morpheme boundary markers are deleted on the surface level by default. (Recall that the colon is used to separate symbols on the intermediate and surface forms). The # symbol is a special symbol that marks a word boundary. Thus (T..1) means 'insert an e after a morpheme-final x, s, or z, and before the morpheme s'. Figure 15 shows an automaton that corresponds to this rule.

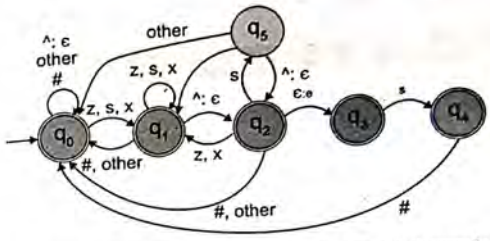


Figure 16: The transducer for the E-insertion rule of (T..1), extended from a similar transducer in Antworth

The idea in building a transducer for a particular rule is to express only the constraints necessary for that rule, allowing any other string of symbols to pass through unchanged. This rule is used to insure that we can only see the ε:e pair if we are in the proper context. So state q0, which models having seen only default pairs unrelated to the rule, is an accepting state, as is q_1, which models having seen a z, s, or x. q_2 models having seen the morpheme boundary after the z, s, or x, and again is an accepting state. State q_3 models having just seen the E-insertion; it is not an accepting state, since the insertion is only allowed if it is followed by the s morpheme and then the end-of-word symbol #.

The other symbol is used in Figure 16 to safely pass through any parts of words that don't play a role in the E-insertion rule. other means 'any feasible pair that is not in this transducer'; it is thus a version of @:@ which is context-dependent in a transducer-by-transducer way. So for exam- ple when leaving state q_0, we go to q_1 on the z, s, or x symbols, rather than following the other arc and staying in q_0. The semantics of other depends on what symbols are on other arcs; since # is mentioned on some arcs, it is (by definition) not included in other, and thus, for example, is explicitly mentioned on the arc from q_2 to q_0. A transducer needs to correctly reject a string that applies the rule when it shouldn't. One possible bad string would have the correct environment for the E-insertion, but have no insertion. State q_5 is used to insure that the e is always inserted whenever the environment is appropriate; the transducer reaches q_5 only when it has seen an s after an appropriate morpheme boundary. If the machine is in state q_5 and the next symbol is #, the machine rejects the string (because there is no legal transition

on # from q_3). Figure 17 shows the transition table for the rule which makes the illegal transitions explicit with the '–' symbol.

State\Input	5:5	x:x	z:z	^:ε	ε:e	#	other
q_0:	1	1	1	0	-	0	0
q_1:	1	1	1	2	-	0	0
q_2:	5	1	1	0	3	0	0
q_3:	4	-	-	-	-	-	-
q_4:	-	-	-	-	-	0	-
q_5:	1	1	1	2	-	-	0

Figure 17: The state-transition table for E-insertion rule of Figure T.2.

7. Lexicon free FST Porter stemmer

While building a transducer from a lexicon plus rules is the standard algorithm for morphological parsing, there are simpler algorithms that don't require the large on-line lexicon demanded by this algorithm. These are used especially in Information Retrieval (IR) tasks in which a user needs some information, and is looking for relevant documents (perhaps on the web, perhaps in a digital library database). User gives the system a query with some important characteristics of documents she desires, and the IR system retrieves what it thinks are the relevant documents. One common type of query is Boolean combinations of relevant keywords or phrases, e.g. (marsupial OR kangaroo OR koala). The system then returns documents that have these words in them. Since a document with the word marsupials might not match the keyword marsupial, some IR systems first run a stemmer on the keywords and on the words in the document. Since morphological parsing in IR is only used to help form equivalence classes, the details of the suffixes are irrelevant; what matters is determining that two words have the same stem.

One of the most widely used such stemming algorithms is the simple and efficient Porter (1980) algorithm, which is based on a series of simple cascaded rewrite rules. Since cascaded rewrite rules are just the sort of thing that could be easily implemented as an FST, we think of the Porter algorithm as a lexicon-free FST stemmer . The algorithm contains rules like:

Rule: ATIONAL ! ATE (e.g. relational ! relate)

Rule: ING ! ε if stem contains vowel (e.g. motoring ! motor)

Do stemmers really improve the performance of information retrieval engines? One problem is that stemmers are not perfect. Following kinds of errors of omission and of commission in the Porter algorithm:

Errors of Commission

organization - organ

Doing - doe

generalization - generic

numerical - numerous

policy - police

university - universe

negligible - negligent

Errors of Omission

European - Europe

analysis - analyzes

matrices - matrix

noise - noisy

sparse - sparsity

explain - explanation

urgency - urgent

Porter Stemmer

A *consonant* in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term 'consonant' is defined to some extent in terms of itself does not make it ambiguous.) So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a *vowel*.

A consonant will be denoted by c, a vowel by v. A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:

CVCV ... C

CVCV ... V

VCVC ... C

VCVC ... V

These may all be represented by the single form

[C]VCVC ... [V]

where the square brackets denote arbitrary presence of their contents. Using (VC)^m to denote VC repeated m times, this may again be written as

[C](VC)^m[V].

m will be called the *measure* of any word or word part when represented in this form. The case m = 0 covers the null word. Here are some examples:

m=0 TR, EE, TREE, Y, BY.

m=1 TROUBLE, OATS, TREES, IVY.

m=2 TROUBLES, PRIVATE, OATEN, ORRERY.

The *rules* for removing a suffix will be given in the form

(condition) S1 -> S2

This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m, e.g.

(m > 1) EMENT ->

Here S1 is 'EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

The 'condition' part may also contain the following:

*S - the stem ends with S (and similarly for the other letters).

v - the stem contains a vowel.

*d - the stem ends with a double consonant (e.g. -TT, -SS).

*o - the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP).

And the condition part may also contain expressions with *and*, *or* and *not*, so that

(m>1 and (*S or *T))

tests for a stem with m>1 ending in S or T, while

(*d and not (*L or *S or *Z))

tests for a stem ending with a double consonant other than L, S or Z. Elaborate conditions like this are required only rarely.

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with

SSES -> SS

IES -> I

SS -> SS

S ->

(here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (S1='SS') and CARES to CARE (S1='S').

In the rules below, examples of their application, successful or otherwise, are given on the right in lower case. The algorithm now follows:

Step 1a

SSES	->	SS	caresses	->	caress
IES	->	I	ponies	->	poni
			ties	->	ti
SS	->	SS	caress	->	caress
S	->		cats	->	cat

Step 1b

(m>0) EED	->	EE	feed	->	feed
			agreed	->	agree
(*v*) ED	->		plastered	->	plaster
			bled	->	bled
(*v*) ING	->		motoring	->	motor
			sing	->	sing

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

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

The rule to map to a single letter causes the removal of one of the double letter pair. The -E is put back on -AT, -BL and -IZ, so that the suffixes -ATE, -BLE and -IZE can be recognised later. This E may be removed in step 4.

Step 1c

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

Step 1 deals with plurals and past participles. The subsequent steps are much more straightforward.

Step 2

(m>0) ATIONAL	->	ATE	relational	->	relate
(m>0) TIONAL	->	TION	conditional	->	condition
			rational	->	rational
(m>0) ENCI	->	ENCE	valenci	->	valence
(m>0) ANCI	->	ANCE	hesitanci	->	hesitance
(m>0) IZER	->	IZE	digitizer	->	digitize

(m>0) ABLI	->	ABLE	conformabli	->	conformable
(m>0) ALLI	->	AL	radicalli	->	radical
(m>0) ENTLI	->	ENT	differentli	->	different
(m>0) ELI	->	E	vileli	->	vile
(m>0) OUSLI	->	OUS	analogousli	->	analogous
(m>0) IZATION	->	IZE	vietnamization	->	vietnamize
(m>0) ATION	->	ATE	predication	->	predicate
(m>0) ATOR	->	ATE	operator	->	operate
(m>0) ALISM	->	AL	feudalism	->	feudal
(m>0) IVENESS	->	IVE	decisiveness	->	decisive
(m>0) FULNESS	->	FUL	hopefulness	->	hopeful
(m>0) OUSNESS	->	OUS	callousness	->	callous
(m>0) ALITI	->	AL	formaliti	->	formal
(m>0) IVITI	->	IVE	sensitiviti	->	sensitive
(m>0) BILITI	->	BLE	sensibiliti	->	sensible

The test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps.

Step 3

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

Step 4

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

(m>1) OUS	->		homologous	->	homolog
(m>1) IVE	->		effective	->	effect
(m>1) IZE	->		bowdlerize	->	bowdler

The suffixes are now removed. All that remains is a little tidying up.

Step 5a

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

Step 5b

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

8. N –Grams

Imagine listening to someone as they speak and trying to guess the next word that they are going to say. For example, what word is likely to follow this sentence fragment?

I'd like to make a collect. . .

Probably the most likely word is call, although it's possible the next word could be telephone, or person-to-person or international. (Think of some others). Guessing the next word (or word prediction) is an essential subtask of speech recognition, hand-writing recognition, augmentative communication for the disabled, and spelling error detection. In such tasks, word-identification is difficult because the input is very noisy and ambiguous. Thus, looking at previous words can give us an important clue about what the next ones are going to be.

N-gram is simply a sequence of N words. For instance, let us take a look at the following examples.

1. San Francisco (is a 2-gram)
2. The Three Musketeers (is a 3-gram)
3. She stood up slowly (is a 4-gram)

Now which of these three N-grams have you seen quite frequently? Probably, "San Francisco" and "The Three Musketeers". On the other hand, you might not have seen "She stood up slowly" that frequently. Basically, "She stood up slowly" is an example of an N-gram that does not occur as often in sentences as Examples 1 and 2.

Now if we assign a probability to the occurrence of an N-gram or the probability of a word occurring next in a sequence of words, it can be very useful. Why?

First of all, it can help in deciding which N-grams can be chunked together to form single entities (like "San Francisco" chunked together as one word, "high school" being chunked as one word).

It can also help make next word predictions. Say you have the partial sentence "Please hand over your". Then it is more likely that the next word is going to be "test" or "assignment" or "paper" than the next word being "school".

We formalize this idea of word prediction with probabilistic models called N-gram models, which predict the next word from the previous N − 1 words. Such statistical models of word sequences are also called language models or LMs. Computing the probability of the next word will turn out to be closely related to computing the probability of a sequence of words. The following sequence, for example, has a non-zero probability of appearing in a text:

. . . all of a sudden, I notice three guys standing on the sidewalk...

while this same set of words in a different order has a much lower probability:

on guys all I of notice sidewalk three a sudden standing the

As we will see, estimators like N-grams that assign a conditional probability to possible next words can be used to assign a joint probability to an entire sentence. Whether estimating probabilities of next words or of whole sequences, the N-gram model is one of the most important tools in speech and language processing. N-grams are essential in any task in which we have to identify words in noisy, ambiguous input. In speech recognition, for example, the input speech sounds are very confusable and many words sound extremely similar. N-gram models are also essential in statistical machine translation.

It can also help to make spelling error corrections. For instance, the sentence "drink coffee" could be corrected to "drink coffee" if you knew that the word "coffee" had a high probability of occurrence after the word "drink" and also the overlap of letters between "coffee" and "coffee" is high. In spelling correction, we need to find and correct spelling errors like the following that accidentally result in real English words:

They are leaving in about fifteen minutes to go to her house.

The design and construction of the system will take more than a year.

Since these errors have real words, we can't find them by just flagging words that aren't in the dictionary. But note that in about fifteen minutes is a much less probable sequence than in about fifteen minutes. A spellchecker can use a probability estimator both to detect these errors and to suggest higher-probability corrections. Word prediction is also important for augmentative communication systems that help the disabled. People who are unable to use speech or sign language to communicate, like the physicist Steven Hawking, can communicate by using simple body movements to select words from a menu that are spoken by the system. Word prediction can be used to suggest likely words for the menu. Besides these sample areas, N-grams are also crucial in NLP tasks like part-of- speech tagging, natural language generation, and word similarity, as well as in applications from authorship identification and sentiment extraction to predictive text input systems for cell phones.

Language model

Problem of Modelling Language

Formal languages, like programming languages, can be fully specified. All the reserved words can be defined and the valid ways that they can be used can be precisely defined. We cannot do this with natural language. Natural languages are not designed; they emerge, and therefore there is no formal specification. There may be formal rules for parts of the language, and heuristics, but natural language that does not confirm is often used. Natural languages involve vast numbers of terms that can be used in ways that introduce all kinds of ambiguities, yet can still be understood by other humans. Further, languages change, word usages change: it is a moving target. Nevertheless, linguists try to specify the language with formal grammars and structures. It can be done, but it is very difficult and the results can be fragile. An alternative approach to specifying the model of the language is to learn it from examples.

Statistical Language Modelling, or Language Modelling and LM for short, is the development of probabilistic models that are able to predict the next word in the sequence given the words that precede it. It is a probability distribution over sequences of words.

Given such a sequence, say of length m, it assigns a probability P(w1,....,wm) to the whole sequence.

The goal of probabilistic language modelling is to calculate the probability of a sentence of sequence of words:

P(W) = P(w1, w2, w3,...., wn)

and can be used to find the probability of the next word in the sequence:

P(w5 | w1, w2, w3, w4)

A model that computes either of these is called a Language Model

Method for Calculating Probabilities:

Conditional Probability:

let A and B be two events with P(B) =/= 0, the conditional probability of A given B is:

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

$$P(x_1, x_2,....x_n) = P(x_1)P(x_2|x_1)...P(x_n|x_1,...,x_{n-1})$$

Chain Rule

The chain rule applied to compute the joined probability of words in a sequence is therefore:

$$P(w_1, w_2,....w_n) = \prod_i P(w_i|w_1, w_2...w_{i-1})$$

For example,

$$P(\text{"its water is so transparent"}) =$$
$$P(its)*$$
$$P(water|its)*$$
$$P(is|its\ water)*$$
$$P(so|its\ water\ is)*$$
$$P(transparent|its\ water\ is\ so)$$

This is a lot to calculate, could we not simply estimate this by counting and dividing the results as shown in the following formula:

$$P(transparent|its\ water\ is\ so) = \frac{count(its\ water\ is\ so\ transparent)}{count(its\ water\ is\ so)}$$

In general, no! There are far to many possible sentences in this method that would need to be calculated and we would like have very sparse data making results unreliable.

Markov Property

A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A process with this property is called a Markov process.

In other words, the probability of the next word can be estimated given only the previous k number of words.

For example, if k=1:

$P(transparent|\ its\ water\ is\ so) \approx P(transparent|\ so)$

or if k=2:

$P(transparent|\ its\ water\ is\ so) \approx P(transparent|\ is\ so)$

General equation for the Markov Assumption, k=i :

$P(w_i|\ w_1w_2...w_{i-1}) \approx P(w_i|\ w_{i-k}...\ w_{i-1})$

The language model provides context to distinguish between words and phrases that sound similar. For example, in American English, the phrases "recognize speech" and "wreck a nice beach" sound similar, but mean different things.

Data sparsity is a major problem in building language models. Most possible word sequences are not observed in training. One solution is to make the assumption that the probability of a word only depends on the previous n words. This is known as an n-gram model or unigram model when n = 1. The unigram model is also known as the bag of words model.

Estimating the relative likelihood of different phrases is useful in many natural language processing applications, especially those that generate text as an output. Language modeling is used in speech recognition, machine translation, part-of-speech tagging, parsing, Optical Character Recognition, handwriting recognition, information retrieval and other applications.

In speech recognition, sounds are matched with word sequences. Ambiguities are easier to resolve when evidence from the language model is integrated with a pronunciation model and an acoustic model.

Language models are used in information retrieval in the query likelihood model. There a separate language model is associated with each document in a collection. Documents are ranked based on the probability of the query Q in the document's language model P(Q| Md). Commonly, the unigram language model is used for this purpose.

N-gram language model

Statistical language models, in its essence, are the type of models that assign probabilities to the sequences of words. In this article, we'll understand the simplest model that assigns probabilities to sentences and sequences of words, the n-gram

You can think of an N-gram as the sequence of N words, by that notion, a 2-gram (or bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (or trigram) is a three-word sequence of words like "please turn your", or "turn your homework"

Let's start with equation P(w|h), the probability of word w, given some history, h. For example,

P(The|its water is so transparent that)

Here,

w = The

h = its water is so transparent that

And, one way to estimate the above probability function is through the relative frequency count approach, where you would take a substantially large corpus, count the number of times you see its water is so transparent that, and then count the number of times it is followed by the. In other words, you are answering the question:

Out of the times you saw the history h, how many times did the word w follow it

P(the| its water is so transparent that) = C(its water is so transparent that the)/C(its water is so transparent that)

Now, you can imagine it is not feasible to perform this over an entire corpus; especially if it is of a significant size.

This shortcoming and ways to decompose the probability function using the chain rule serves as the base intuition of the N-gram model. Here, you, instead of computing probability using the entire corpus, would approximate it by just a few historical words

The Bigram Model

As the name suggests, the bigram model approximates the probability of a word given all the previous words by using only the conditional probability of one preceding word. In other words, you approximate it with the probability: P(the | that)

And so, when you use a bigram model to predict the conditional probability of the next word, you are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

This assumption that the probability of a word depends only on the previous word is also known as Markov assumption.

Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far in the past. The Markov assumption is the presumption that the future behavior of a dynamical system only depends on its recent history. In particular, in a kth-order Markov model, the next state only depends on the k most recent states, therefore an N-gram model is a (N−1)-order Markov model.

Maximum Likelihood Estimate

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)}{count(w_{i-1})}$$

$$P(w_i|w_{i-1}) = \frac{c(w_i|w_{i-1})}{c(w_{i-1})}$$

Example

$$P(w_i|w_{i-1}) = \frac{c(w_i|w_{i-1})}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> I am Sam </s>
<s> I do not like green eggs and ham </s>

$P(I|<s>) = \frac{2}{3} = .67$ $P(Sam|<s>) = \frac{1}{3} = .33$ $P(am|I) = \frac{2}{3} = .67$

$P(</s>|Sam) = \frac{1}{2} = 0.5$ $P(Sam|am) = \frac{1}{2} = .5$ $P(do|I) = \frac{1}{3} = .33$

You can further generalize the bigram model to the trigram model which looks like two words into the past and can thus be further generalized to the N-gram model. Basically, an N-gram model predicts the occurrence of a word based on the occurrence of its N – 1 previous words. An N-gram model uses the previous N-1 words to predict the next one: P(wn | wn-N+1 wn-N+2... wn-1)

P(phone | Please turn off your cell)

Number of parameters required grows exponentially with the number of words of prior context.

Unigram: P(phone)

Bigram: P(phone | cell)

Trigram: P(phone | your cell)

Another example for the sentence boy is chasing the big dog

unigrams: P(dog)

bigrams: P(dog | big)

trigrams: P(dog | the big)

quadrigrams: P(dog | chasing the big)

Assume a language has T word types in its lexicon, how likely is word x to follow word y?

Simplest model of word probability: 1/T

Alternative 1: estimate likelihood of x occurring in new text based on its general frequency of occurrence estimated from a corpus (unigram probability)

popcorn is more likely to occur than unicorn

Alternative 2: condition the likelihood of x occurring in the context of previous words (bigrams, trigrams,...) Estimate the probability of each word given prior context.

mythical unicorn is more likely than mythical popcorn

So here we are answering the question – how far back in the history of a sequence of words should we go to predict the next word? For instance, a bigram model (N = 2) predicts the occurrence of a word given only its previous word (as N – 1 = 1 in this case). Similarly, a trigram model (N = 3) predicts the occurrence of a word based on its previous two words (as N – 1 = 2 in this case).

Let us see a way to assign a probability to a word occurring next in a sequence of words. First of all, we need a very large sample of English sentences (called **a corpus**).

For the purpose of our example, we'll consider a very small sample of sentences, but in reality, a corpus will be extremely large. Say our corpus contains the following sentences:

1. He said thank you.
2. He said bye as he walked through the door.
3. He went to San Diego.

4. San Diego has nice weather.
5. It is raining in San Francisco.

Let's assume a bigram model. So we are going to find the probability of a word based only on its previous word. In general, we can say that this probability is (the number of times the previous word 'wp' occurs before the word 'wn') / (the total number of times the previous word 'wp' occurs in the corpus) =

(Count (wp wn))/(Count (wp))

Let's work this out with an example.

To find the probability of the word "you" following the word "thank", we can write this as P (you | thank) which is a conditional probability.

This becomes equal to:

=(No. of times "Thank You" occurs) / (No. of times "Thank" occurs)

= 1/1

= 1

We can say with certainty that whenever "Thank" occurs, it will be followed by "You" (This is because we have trained on a set of only five sentences and "Thank" occurred only once in the context of "Thank You"). Let's see an example of a case when the preceding word occurs in different contexts.

Let's calculate the probability of the word "Diego" coming after "San". We want to find the P (Diego | San). This means that we are trying to find the probability that the next word will be "Diego" given the word "San". We can do this by:

=(No of times "San Diego" occurs) / (No. of times "San" occurs)

= 2/3

= 0.67

This is because in our corpus, one of the three preceding "San"s was followed by "Francisco". So, the P (Francisco | San) = 1 / 3.

In our corpus, only "Diego" and "Francisco" occur after "San" with the probabilities 2 / 3 and 1 / 3 respectively. So if we want to create a next word prediction software based on our corpus, and a user types in "San", we will give two options: "Diego" ranked most likely and "Francisco" ranked less likely.

Generally, the bigram model works well and it may not be necessary to use trigram models or higher N-gram models.

Challenges

There are, of course, challenges, as with every modeling approach, and estimation method. Let's look at the key ones affecting the N-gram model, as well as the use of Maximum Likelihood Estimation (MLE)

Sparse data

Not all N-grams found in the training data, need smoothing

Change of domain

Train on WSJ, attempt to identify Shakespeare – won't work

Sensitivity to the training corpus

The N-gram model, like many statistical models, is significantly dependent on the training corpus. As a result, the probabilities often encode particular facts about a given training corpus. Besides, the performance of the N-gram model varies with the change in the value of N.

Moreover, you may have a language task in which you know all the words that can occur, and hence we know the vocabulary size V in advance. The closed vocabulary assumption assumes there are no unknown words, which is unlikely in practical scenarios.

Smoothing

A notable problem with the MLE approach is sparse data. Meaning, any N-gram that appeared a sufficient number of times might have a reasonable estimate for its probability. But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.

As a result of it, the N-gram matrix for any training corpus is bound to have a substantial number of cases of putative "zero probability N-grams"

N-gram for spelling correction

Let's define the job of a spell checker and an auto corrector. A word needs to be checked for spelling correctness and corrected if necessary, many a time in the context of the surrounding words. A spellchecker points to spelling errors and possibly suggests alternatives. An auto corrector usually goes a step further and automatically picks the most likely word. In case of the correct word already having been typed, the same is retained. So, in practice, an auto correct is a bit more aggressive than a spellchecker, but this is more of an implementation detail — tools allow you to configure the behaviour.

There is not much difference between the two in theory. So, the discussion in the rest of the blog post applies to both.

There are different types of spelling errors. Let's try to classify them a bit formally. Note that the below is for the purpose of illustration, and is not an rigorous linguistic approach to classifying spelling errors.

Non-word Errors: These are the most common type of errors. You either miss a few keystrokes or let your fingers hurtle a bit longer. E.g., typing langage when you meant language; or hurryu when you meant hurry

Real Word Errors: If you have fat fingers, sometimes instead of creating a non-word, you end up creating a real word, but one you didn't intend. E.g. typing buckled when you meant bucked. Or your fingers are a tad wonky, and you type in three when you meant there.

Cognitive Errors: The previous two types of errors result not from ignorance of a word or its correct spelling. Cognitive errors can occur due to those factors. The words piece and peace are homophones (sound the same). So you are not sure which one is which. Sometimes your damn sure about your spellings despite a few grammar nazis claim you're not.

Short forms/Slang/Lingo: These are possibly not even spelling errors. May be u r just being kewl. Or you are trying hard to fit in everything within a text message or a tweet and must commit a spelling sin. We mention them here for the sake of completeness.

Intentional Typos: Well, because you are clever. You type in teh and pwned and zomg carefully and frown if they get autocorrected. It could be a marketing trick, one that probably even backfired.

The word N-gram approach to spelling error detection and correction is to generate every possible misspelling of each word in a sentence either just by typographical modifications (letter insertion, deletion, substitution), or by including homophones as well, (and presumably including the correct spelling), and then choosing the spelling that gives the sentence the highest prior probability. That is, given a sentence $W = \{w_1, w_2, \ldots, w_k, \ldots, w_n\}$, where wk has alternative spelling w'k , w''k , etc, we choose the spelling among these possible spellings that max- imizes P(W), using the N-gram grammar to compute P(W). A class-based N-gram can be used instead, which can find unlikely part-of-speech combinations, although it may not do as well at finding unlikely word combinations.

Expected Questions:

1. What is morphology. Why do we need to do Morphological Analysis? Discuss various application domains of Morphological Analysis.
2. Explain derivational & inflectional morphology in detail with suitable examples.
3. What are morphemes? What are different ways to create words from morphemes?
4. What is language model? Explain the use of Language model?
5. Write a note on N-Gram language Model.
6. What is the role of FSA in Morphological analysis? Explain FST in detail.

Syntax Analysis

3

OBJECTIVES

After reading this chapter, the student will be able to understand:

- Part-Of-Speech tagging (POS)- Tag set for English (Penn Treebank) .
- Rule based POS tagging, Stochastic POS tagging.
- Issues –Multiple tags & words, Unknown words.
- Introduction to CFG.
- Sequence labelling: Hidden Markov Model (HMM), Maximum Entropy, and Conditional Random Field (CRF).

Syntax Analysis

Syntax refers to the arrangement of words in a sentence such that they make grammatical sense. In NLP, syntactic analysis is used to assess how the natural language aligns with the grammatical rules. Computer algorithms are used to apply grammatical rules to a group of words and derive meaning from them.

Syntactic analysis helps us understand the roles played by different words in a body of text. The words themselves are not enough. Consider Innocent peacefully children sleep little vs Innocent little children sleep peacefully. There is some evidence that human understanding of language is, in part, based on structural analysis. Consider "Twas brillig and the slithy toves did gyre and gimble in the wabe." Here we can understand the sentence on some level, even though most of the words make no sense to us. Colorless green ideas sleep furiously makes more sense to us than Ideas green furiously colorless sleep Finally, consider the sentence the old dog the footsteps of the young. In reading this sentence, you might have found yourself having to re-examine the first part. In all likelihood, you thought that the word "dog" was being used as a noun, when, in fact, it is a verb.

Here are some syntax techniques that can be used:

Lemmatization: It entails reducing the various inflected forms of a word into a single form for easy analysis.

Morphological segmentation: It involves dividing words into individual units called morphemes.

Word segmentation: It involves dividing a large piece of continuous text into distinct units.

Part-of-speech tagging: It involves identifying the part of speech for every word.

Parsing: It involves undertaking grammatical analysis for the provided sentence.

Sentence breaking: It involves placing sentence boundaries on a large piece of text.

Stemming: It involves cutting the inflected words to their root form.

1. Part-Of-Speech tagging(POS)

The part of speech tagging is a process of assigning corresponding part of speech like noun, verb, adverb, adjective, verb to each word in a sentence. It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

Automatic assignment of descriptors to the given tokens is called Tagging. The descriptor is called tag. The tag may indicate one of the parts-of-speech, semantic information, and so on. POS tagging is applied to language grammatical rules to parse meanings of sentences and phrases. Assume we have A tag set, A dictionary that gives you the possible set of tags for each entry, A text to be tagged. Output: is a Single best tag for each word , E.g., Book/VB that/DT flight/NN /The main challenge in POS tagging is to resolving the ambiguity in possible POS tags for a word. The word bear in the following sentences have the same spelling but different meanings. She saw a bear. Your efforts will bear fruit.

Words are traditionally grouped into equivalence classes called parts of speech, word classes, morphological classes, or lexical tags. In traditional grammar there were generally only a few parts of speech (noun, verb, adjective, preposition, adverb, conjunction, etc.). More recent models have much larger numbers of word classes (45 for the Penn Treebank ,87 for the Brown corpus, and 146 for the C7 tag set). The part of speech for a word gives a significant amount of information about the word and its neighbours. This is clearly true for major categories, (verb versus noun), but is also true for the many finer distinctions. For example, these tag sets distinguish between possessive pronouns (my, your, his, her, its) and personal pronouns (I, you, he, me). Knowing whether a word is a possessive pronoun or a personal pronoun can tell us what words are likely to occur in its vicinity (possessive pronouns are likely to be followed by a noun, personal pronouns by a verb).

A word's part-of-speech can tell us something about how the word is pronounced. A noun or an adjective are pronounced differently (the noun is pronounced CONtent and the adjective conTENT). Thus, knowing the part of speech can produce more natural pronunciations in a speech synthesis system and more accuracy in a speech recognition system.

Parts of speech can also be used in stemming for informational retrieval (IR), since knowing a word's part of speech can help tell us which morphological affixes it can take. They can also help an IR application by helping select out nouns or other important words from a document. Automatic part-of-speech taggers can help in building automatic word-sense disambiguation algorithms, and POS taggers are also used in advanced ASR language models such as class based N-grams. Parts of speech are very often used for 'partial parsing' texts, for example for quickly finding names or other phrases for the information extraction applications .Finally, corpora that have been marked for part-of-speech are very useful for linguistic research, for example to help find instances or frequencies of particular constructions in large corpora.

Tag set for English

Parts of speech can be divided into two broad super categories: closed class types and open class types. Closed classes are those that have relatively fixed membership. For example, prepositions are a closed class because there is a fixed set of them in English. By contrast nouns and verbs are open classes because new nouns and verbs are continually coined or borrowed from other languages (e.g. the new verb to fax or the borrowed noun futon). It is likely that any given speaker or corpus will have different open class words, but all speakers of a language, and corpora that are large enough, will likely share the set of closed class words. Closed class words are generally also function words; function words are grammatical words like of, it, and, or you, which tend to be very short, occur frequently, and play an important role in grammar. There are four major open classes that occur in the languages of the world: nouns, verbs, adjectives, and adverbs. It turns out that English has nouns verbs adjectives adverbs all four of these, although not every language does. Many languages have no adjectives. Every known human language has at least two categories noun and verb. Noun is the name given to the lexical class in which the words for most people, places, or things occur. But since lexical classes like noun are defined functionally (morphological and syntactically) rather than semantically, some words for people, places, and things may not be nouns, and conversely some nouns may not be words for people, places, or things. Thus nouns include concrete terms like ship and chair, abstractions like band- width and relationship, and verb-like terms like pacing in His pacing to and fro became quite annoying). What defines a noun in English, then, are things like its ability to occur with determiners (a goat, its bandwidth, Plato's Republic), to take possessives (IBM's annual revenue), and for most but not all nouns, to occur in the plural form (goats, abaci).

Nouns are traditionally grouped into proper nouns and common nouns. Proper nouns, like Regina, Colorado, and IBM, are the names of specific persons or entities. In English, they generally aren't preceded by articles (e.g. the book is upstairs, but Regina is upstairs). In written English, proper nouns are usually capitalized. In many languages, including English, common nouns are divided into count nouns and mass nouns. Count nouns are those that allow gram- count nouns mass nouns matical enumeration; that is, they can occur in both the singular and plural (goat/goats, relationship/relationships) and they can be counted (one goat, two goats). Mass nouns are used when something is conceptualized as a homogeneous group. So words like snow, salt, and communism are not counted (i.e. two snows or two communisms). Mass nouns can also appear without articles where singular count nouns cannot (Snow is white but not Goat is white). The verb class includes most of the words referring to actions and processes, including main verbs like draw, provide, differ, and go. English verbs have a number of morphological forms (non- 3rd-person-sg (eat), 3d-person-sg (eats), progressive (eating), past participle eaten). A subclass of English verbs called auxiliaries will be discussed when we turn to closed class forms. The third open class English form is adjectives; semantically this class includes many terms that describe properties or qualities. Most languages have adjectives for the concepts of color (white, black), age (old, young), and value (good, bad), but there are languages without adjectives. As we discussed above, many linguists argue that the Chinese family of languages uses verbs to describe such English-adjectival notions as color and age.

The final open class form is adverbs. What coherence the class has semantically may be solely that each of these words can be viewed as modifying something (often verbs, hence the name 'adverb', but also other adverbs and entire verb phrases). Directional adverbs or locative adverbs (home, here, downhill) specify the direction or location of some action; degree adverbs (extremely, very, somewhat) specify the extent of some action, process, or property; manner adverbs (slowly, slinkily, delicately) describe the manner of some action or process and temporal adverbs describe the time that some action or event took place (yesterday, Monday). Because of the heterogeneous nature of this class, some adverbs (for example temporal adverbs like Monday) are tagged in some tagging schemes as nouns. The closed classes differ from language to language than do the open classes.

Prepositions occur before noun phrases; semantically they are relational, often indicating spatial or temporal relations, whether literal (on it, before then, by the house) or metaphorical (on time, with gusto, beside her- self). But they often indicate other relations as well (Hamlet was written by Shakespeare, and (from Shakespeare) "And I did laugh sans intermission an hour by his dial").

A particle is a word that resembles a preposition or an adverb, and that often combines with a verb to form a larger unit called a phrasal verb, as in the following examples from Thoreau: So I went on for some days cutting and hewing timber. . . Moral reform is the effort to throw off sleep. . . We can see that these are particles rather than prepositions, for in the first example, on is followed, not by a noun phrase, but by a true preposition phrase. With transitive phrasal verbs, as in the second example, we can tell that off is a particle and not a preposition because particles may appear after their objects (throw sleep off as well as throw off sleep). This is not possible for prepositions (The horse went off its track, but *The horse went its track off).

Articles a, an, and the often begin a noun phrase. A and an mark a noun phrase as indefinite, while the can mark it as definite. Conjunctions are used to join two phrases, clauses, or sentences. Coordinating conjunctions like and, or, or but, join two elements of equal status. Subordinating conjunctions are used when one of the elements is of some sort of embedded status. For example in 'I thought that you might like some milk' is a subordinating conjunction that links the main clause I thought with the subordinate clause you might like some milk. This clause is called subordinate because this entire clause is the 'content' of the main verb thought. Subordinating conjunctions like that which link a verb to its argument in this way are also called complementizers.

Pronouns are forms that often act as a kind of shorthand for referring to some noun phrase or entity or event. Personal pronouns refer to per- sons or entities (you, she, I, it, me, etc). Possessive pronouns are forms of personal pronouns that indicate either actual possession or more often just an abstract relation between the person and some object (my, your, his, her, WH its, one's, our, their). Wh-pronouns (what, who, whom, whoever) are used in certain question forms, or may also act as complementizers (Frieda, who I met five years ago. . .).

A closed class subtype of English verbs is the auxiliary verbs. Cross- AUXILIARY linguistically, auxiliaries are words (usually verbs) that mark certain semantic features of a main verb, including whether an action takes place in the present, past or future

(tense), whether it is completed (aspect), whether it is negated (polarity)? ... an action is necessary, possible, suggested, desired, etc. (mood).

English also has many words of more or less unique function, including interjections (oh, ah, hey, man, alas), negatives (no, not), politeness markers (please, thank you), greetings (hello, goodbye), and the existential there (there are two on the table) among others. Whether these classes are assigned particular names or lumped together (as interjections or even adverbs) depends on the purpose of the labelling.

Penn Treebank

There are a small number of popular tag sets for English, many of which evolved from the 87-tag tagset used for the Brown corpus. Three of the most commonly used are the small 45-tag Penn Treebank tagset , the medium-sized 61 tag C5 tagset used by the Lancaster UCREL project's CLAWS (the Constituent Likelihood Automatic Word-tagging System) tagger to tag the British National Corpus (BNC) and the larger 146-tag C7 tagset . The Penn Treebank tagset, has been applied to the Brown corpus and a number of other corpora. Here is an example of a tagged sentence from the Penn Treebank version of the Brown corpus (in a flat ASCII file, tags are often represented after each word, following a slash, but the tags can also be represented in various other ways). The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/ NNS ./.

The Penn Treebank tagset was culled from the original 87-tag tagset for the Brown corpus. This reduced set leaves out information that can be recovered from the identity of the lexical item. For example the original Brown tagset and other large tagsets like C5 include a separate tag for each of the different forms of the verbs do (e.g. C5 tag 'VDD' for did and 'VDG' for doing), be, and have. These were omitted from the Penn set.

Table : Penn tree bank set

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*

Number	Tag	Description
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRPS	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*

Number	Tag	Description
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

2. Rule based POS tagging, Stochastic POS tagging

Part-of-speech tagging is the process of assigning a part-of-speech or other lexical class marker to each word in a corpus. Tags are also usually applied to punctuation markers; thus, tagging for natural language is the same process as tokenization for computer languages, although tags for natural languages are much more ambiguous. As we suggested at the beginning of the chapter, taggers play an increasingly important role in speech recognition, natural language parsing and information retrieval. The input to a tagging algorithm is a string of words and a specified TAGSET tagset of the kind described in the previous section. The output is a single best tag for each word.

VB DT NN.

Book that flight.

VBZ DT NN VB NN?

Does that flight serve dinner?

Even in these simple examples, automatically assigning a tag to each word is not trivial. For example, book is ambiguous. That is, it has more than one possible usage and part of speech. It can be a verb (as in book that flight or to book the suspect) or a noun (as in hand me that book, or a book of matches). Similarly that can be a determiner (as in Does that flight serve dinner), or a complementizer (as in I thought that your flight was earlier). The problem of POS-tagging is to resolve these ambiguities, choosing the proper tag for the context. Part-of-speech tagging is thus one of the many disambiguation tasks we will see in this book. How hard is the tagging problem? Most words in English are unambiguous; i.e. they have only a single tag. But many of the most common words of English are ambiguous (for example can can be an auxiliary ('to be able'), a noun ('a metal container'), or a verb ('to put something in such a metal container')).

Most tagging algorithms fall into one of two classes: rule-based taggers and stochastic taggers. Rule-based taggers generally involve a large database of handwritten disambiguation rules which specify, for example, that an ambiguous word is a noun rather than a verb if it follows a determiner. Stochastic taggers generally resolve tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context.

Rule-Based Part-of-Speech Tagging

One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag. Disambiguation can also be performed in rule-based tagging by analyzing the linguistic features of a word along with its preceding as well as following words. For example, suppose if the preceding word of a word is article then word must be a noun.

As the name suggests, all such kind of information in rule-based POS tagging is coded in the form of rules. These rules may be either Context-pattern rules Or, as Regular expression compiled into finite-state automata, intersected with lexically ambiguous sentence representation.

We can also understand Rule-based POS tagging by its two-stage architecture .In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech. In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.

The rules can be categorized in to two parts:

a. **Lexical:** Word level

b. **Context Sensitive Rules:** Sentence level

The transformation-based approaches use a predefined set of handcrafted rules as well as automatically induced rules that are generated during training

Basic Idea:

1. Assign all possible tags to words
2. Remove tags according to set of rules of type:
3. if word+1 is an adj, adv, or quantifier and the following is a sentence boundary
4. And
5. word-1 is not a verb like "consider"
6. then
7. eliminate non-adv else eliminate adv.
8. Typically more than 1000 hand-written rules, but may be machine-learned.

First Stage:

FOR each word

Get all possible parts of speech using a morphological analysis algorithm

Example

	NM			
	RB			
VBN		JJ		VB
PRP	VBD	TO	VB	DT NM
She	**promised**	**to**	**back**	**the bill**

Apply rules to remove possibilities

Example Rule:

IF VBD is an option and VBN|VBD follows "<start>PRP"

THEN Eliminate VBN

NM

RB

VBN JJ VB

PRP VBD TO VB DT NM

She promised to back the bill

ENGTWOL Rule-Based Tagger

A Two-stage architecture

Use lexicon FST (dictionary) to tag each word with all possible POS

Apply hand-written rules to eliminate tags.

The rules eliminate tags that are inconsistent with the context, and should reduce the list of POS tags to a single POS per word.

Det-Noun Rule:If an ambiguous word follows a determiner, tag it as a noun

ENGTWOL Adverbial-that Rule

Given input "that"

If the next word is adj, adverb, or quantifier, and following that is a sentence boundary, and the previous word is not a verb like "consider" which allows adjs as object complements,

Then eliminate non-ADV tags,

Else eliminate ADV tag

I consider that odd. (that is NOT ADV)

It isn't that strange. (that is an ADV)

This approach does work and produces accurate results.

What are the drawbacks?

Extremely labor-intensive

Word	POS	Additional POS features
smaller	ADJ	COMPARATIVE
entire	ADJ	ABSOLURE ATTRIBUTIVE
fast	ADV	SUPERLATIVE
that	DET	CENTRAL DEMONSTRATIVE SG
all	DET	PREDETERMINER SG/PL QUANTIFIER
dog's	N	GENITIVE SG

furniture	N	NOMINATIVE SG NOINDEFDETERMINER
one-third	NUM	SG
she	PRON	PERSONAL FEMININE NOMINATIVE SG3
show	V	IMPERATIVE VFIN
show	V	PRESENT-SG3 VFIN
show	N	NOMINATIVE SG
shown	PCP2	SVOO SVO SV
occurred	PCP2	SV
occurred	V	PAST VFIN SV

Sample ENGTWOL Lexicon

Stage 1 of ENGTWOL Tagging

First Stage: Run words through Kimmo-style morphological analyzer to get all parts of speech.

Example: *Pavlov had shown that salivation ...*

Pavlov **PAVLOV N NOM SG PROPER**

had **HAVE V PAST VFIN SVO**

HAVE PCP2 SVO

shown **SHOW PCP2 SVOO SVO SV**

that ADV

PRON DEM SG

DET CENTRAL DEM SG

CS

salivation **N NOM SG**

Stage 2 of ENGTWOL Tagging

Second Stage: Apply constraints.

Constraints used in negative way.

Example: Adverbial "that" rule
Given input: "that"
If
(+1 A/ADV/QUANT)
(+2 SENT-LIM)
(NOT -1 SVOC/A)
Then eliminate non-ADV tags
Else eliminate ADV

Properties of Rule-Based POS Tagging

Rule-based POS taggers possess the following properties –
These taggers are knowledge-driven taggers.
The rules in Rule-based POS tagging are built manually.
The information is coded in the form of rules.
We have some limited number of rules approximately around 1000.
Smoothing and language modelling is defined explicitly in rule-based taggers.
Advantages of Rule Based Taggers: -

a. Small set of simple rules.
b. Less stored information.

Drawbacks of Rule Based Taggers

a. Generally less accurate as compared to stochastic taggers.

Stochastic Part of Speech Taggers

The use of probabilities in tags is quite old. Stochastic taggers use probabilistic and statistical information to assign tags to words. These taggers might use 'tag sequence probabilities', 'word frequency measurements' or a combination of both.

Based on probability of certain tag occurring given various possibilities. Requires a training corpus. No probabilities for words not in corpus. Training corpus may be different from test corpus.

E.g.

1. The tag encountered most frequently in the training set is the one assigned to an ambiguous instance of that word (word frequency measurements).

2. The best tag for a given word is determined by the probability that it occurs with the n previous tags (tag sequence probabilities)

Advantages of Stochastic Part of Speech Taggers

1. Generally more accurate as compared to rule based taggers.

Drawbacks of Stochastic Part of Speech Taggers

1. Relatively complex.
2. Require vast amounts of stored information.

Stochastic taggers are more popular as compared to rule based taggers because of their higher degree of accuracy. However, this high degree of accuracy is achieved using some sophisticated and relatively complex procedures and data structures.

For a given sentence or word sequence, Hidden Markov Models (HMM) taggers choose the tag sequence that maximizes the following formula:

$$P(\text{word} \mid \text{tag}) * P(\text{tag} \mid \text{previous n tags}) \quad ..(1)$$

HMM taggers generally choose a tag sequence for a whole sentence rather than for a single word, but for pedagogical purposes, let's first see how an HMM tagger assigns a tag to an individual word. We first give the basic equation, then work through an example, and, finally, give the motivation for the equation. A bigram-HMM tagger of this kind chooses the tag t_i for word w_i that is most probable given the previous tag t_{i-1} and the current word w_i

$$t_i = \underset{j}{\operatorname{argmax}} P(t_{j-1}, w_i) \quad ...(2)$$

Through some simplifying Markov assumptions that we will give below, we restate Equation 2 to give the basic HMM equation for a single tag as follows:

$$t_i = \underset{j}{\operatorname{argmax}} P(t_j \mid t_{j-1}) P(w_i \mid t_j) \quad ...(3)$$

An Example

Let's work through an example, using an HMM tagger to assign the proper tag to the single word race in the following examples (both shortened slightly from the Brown corpus):

Secretariat/NNP is/VBZ expected/VBN to/TO race/VB tomorrow/NN ..(4)

People/NNS continue/VBP to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/JJ space/NN ..(5)

In the first example race is a verb (VB), in the second noun (NN). For the purposes of this example, let's pretend that some other mech- anism has already done the best tagging job possible on the surrounding words, leaving only the word race untagged. A bigram version of the HMM tagger makes the simplifying assumption that the tagging problem can be solved by looking at nearby words and tags. Consider the problem of as- signing a tag to race given just these subsequences: to/TO race/??? the/DT race/??? Let's see how this equation applies to our example with race; Equation 3 says that if we are trying to choose between NN and VB for the sequence to race, we choose the tag that has the greater of these two probabilities:

P(VB|TO)P(race|VB) ..(6)

And

P(NN|TO)P(race|NN) ..(7)

Equation 3 and its instantiations Equations 6 and 7 each have two probabilities: a tag sequence probability $P(t_i|t_i\text{-}1)$ and a word-likelihood $P(w_i|t_i)$. For race, the tag sequence probabilities P(NN|TO) and P(VB|TO) give us the answer to the question "how likely are we to expect a verb (noun) given the previous tag?". They can just be computed from a corpus by count- ing and normalizing. We would expect that a verb is more likely to follow TO than a noun is, since infinitives (to race, to run, to eat) are common in English. While it is possible for a noun to follow TO (walk to school, related to hunting), it is less common. Sure enough, a look at the combined Brown and Switchboard corpora gives us the following probabilities, showing that verbs are fifteen times as likely as nouns after TO:

P(NN|TO) = .021

P(VB|TO) = .34

The second part of Equation 3 and its instantiations Equations 6 and 7 is the lexical likelihood: the likelihood of the noun race given each tag, P(race|VB) and P(race|NN). Note that this likelihood term is not asking 'which is the most likely tag for this word'. That is, the likelihood term is not P(VB|race). Instead we are computing P(race|VB). The probability, slightly counterintuitively, answers the question "if we were expecting a verb, how likely is it that this verb would be race". Here are the lexical likelihoods from the combined Brown and Switchboard corpora: P(race|NN) = .00041

P(race|VB) = .00003

If we multiply the lexical likelihoods with the tag sequence probabili- ties, we see that even the simple bigram version of the HMM tagger correctly tags race as a VB despite the fact that it is the less likely sense of race:

P(VB|TO)P(race|VB) = :00001

P(NN|TO)P(race|NN) = :000007

Transformation-Based Tagging

Transformation-Based Tagging, sometimes called Brill tagging, is an in- stance of the Transformation-Based Learning (TBL) approach to machine learning , and draws inspiration from both the rule-based and stochastic taggers. Like the rule-based taggers, TBL is based on rules that specify what tags should be assigned to what words. But like the stochastic taggers, TBL is a machine learning technique, in which rules are automatically induced from the data. Like some but not all of the HMM taggers, TBL is a supervised learning technique; it assumes a pre-tagged training corpus. Samuel et al. (1998a) offer a useful analogy for understanding the TBL paradigm, which they credit to Terry Harvey. Imagine an artist painting a picture of a white house with green trim against a blue sky. Suppose most of the picture was sky, and hence most of the picture was blue. The artist might begin by using a very broad brush and painting the entire canvas blue. Next she might switch to a somewhat smaller white brush, and paint the entire house white. She would just color in the whole house, not worrying about the brown roof, or the blue windows or the green gables. Next she takes a smaller brown brush and colors over the roof. Now she takes up the blue paint on a small brush and paints in the blue windows on the barn. Finally she takes a very fine green brush and does the trim on the gables. The painter starts with a broad brush that covers a lot of the canvas but colors a lot of areas that will have to be repainted. The next layer colors less of the canvas, but also makes less 'mistakes'. Each new layer uses a finer brush that corrects less of the picture, but makes fewer mistakes. TBL uses somewhat the same method as this painter. The TBL algorithm has a set of tagging rules. A corpus is first tagged using the broadest rule, i.e. the one that applies to the most cases. Then a slightly more specific rule is chosen, which changes some of the original tags. Next an even narrower rule, which changes a smaller number of tags (some of which might be previously-changed tags).

How TBL rules are applied

Let's look at one of the rules used by Brill's tagger. Before the rules apply, the tagger labels every word with its most-likely tag. We get these most-likely tags from a tagged corpus. For example, in the Brown corpus, race is most likely to be a noun:

P(NN|race) = .98

P(VB|race) = .02

This means that the two examples of race that we saw above will both be coded as NN. In the first case, this is a mistake, as NN is the incorrect tag:

is/VBZ expected/VBN to/TO race/NN tomorrow/NN ..8

In the second case this race is correctly tagged as an NN:

the/DT race/NN for/IN outer/JJ space/NN ..9

After selecting the most-likely tag, Brill's tagger applies its transformation rules. As it happens, Brill's tagger learned a rule that applies exactly to this mistagging of race: Change NN to VB when the previous tag is TO This rule would change race/NN to race/VB in exactly the following situation, since it is preceded by to/TO:

expected/VBN to/TO race/NN --> expected/VBN to/TO race/VB..10

How TBL Rules are Learned

Brill's TBL algorithm has three major stages. It first labels every word with its most-likely tag. It then examines every possible transformation, and selects the one that results in the most improved tagging. Finally, it then re-tags the data according to this rule. These three stages are repeated until some stopping criterion is reached, such as insufficient improvement over the previous pass. Note that stage two requires that TBL knows the correct tag of each word; i.e., TBL is a supervised learning algorithm. The output of the TBL process is an ordered list of transformations; these then constitute a 'tagging procedure' that can be applied to a new corpus. In principle the set of possible transformations is infinite, since we could imagine transformations such as "transform NN to VB if the previous word was 'IBM' and the word 'the' occurs between 17 and 158 words before that". But TBL needs to consider every possible transformation, in order to pick the best one on each pass through the algorithm. Thus the algorithm needs a way to limit the set of transformations. This is done by designing a small set of templates, abstracted transformations. Every allowable transformation is an instantiation of one of the templates.

The preceding (following) word is tagged **z**.
The word two before (after) is tagged **z**.
One of the two preceding (following) words is tagged **z**.
One ofhte three preceding (following) words is tagged **z**.
The preceding word is tagged **z** and the following word is tagged **w**.
The Preceding (following) word is tagged **z** and the word.
two before (after) is tagged **w**.

	Change tags			
#	From	To	Condition	Example
1	NN	VB	Previous tag is TO	to/TO race/NN → VB
2	VBP	VB	One of the previous 3 tags is MD	might/MD vanish/VBP → VB
3	NN	VB	One of the previous 2 tags is MD	might/MD not replay/NN → VB
4	VB	NN	One of the previous 2 tags is DT	
5	VBD	VBN	One of the previous 3 tags is VBZ	

Figure 1: Figure Templates for TBL

TBL problems and advantages

Problems

Infinite loops and rules may interact

The training algorithm and execution speed is slower than HMM

Advantages

It is possible to constrain the set of transformations with "templates"

IF tag Z or word W is in position *-k

THEN replace tag X with tag

Learns a small number of simple, non-stochastic rules

Speed optimizations are possible using finite state transducers

TBL is the best performing algorithm on unknown words

The Rules are compact and can be inspected by humans

3. Issues

Multiple tags and multiple words

Two issues that arise in tagging are tag indeterminacy and multi-part words. Tag indeterminacy arises when a word is ambiguous between multiple tags and it is impossible or very difficult to disambiguate. In this case, some taggers allow the use of multiple tags. This is the case in the Penn Treebank and in the British National Corpus. Common tag indeterminacies include adjective versus preterite versus past participle (JJ/VBD/VBN), and adjective versus noun as prenominal modifier (JJ/NN).

The second issue concerns multi-part words. The C5 and C7 tagsets, for example, allow prepositions like 'in terms of' to be treated as a single word by adding numbers to each tag:

in/II31 terms/II32 of/II33

Finally, some tagged corpora split certain words; for example the Penn Treebank and the British National Corpus splits contractions and the 's-genitive from their stems:

would/MD n't/RB

children/NNS 's/POS

Unknown words

All the tagging algorithms we have discussed require a dictionary that lists the possible parts of speech of every word. But the largest dictionary will still not contain every possible word. Proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate. Therefore in order to build a complete tagger we need some method for guessing the tag of an unknown word. The simplest possible unknown-word algorithm is to pretend that each unknown word is ambiguous among all possible tags, with equal probability. Then the tagger must rely solely on the contextual POS-trigrams to suggest the proper tag. A slightly more complex algorithm is based on the idea that the probability distribution of tags over unknown words is very similar to the distribution of tags over words that occurred only once in a training set. These words that only occur once are known as hapax legomena (singular hapax legomenon). For example, unknown words and hapax legomena are similar in that they are both most likely to be nouns, followed by verbs, but are very unlikely to be determiners or interjections. Thus the likelihood P(wi |ti) for an unknown word is

determined by the average of the distribution over all singleton words in the training set. that this idea of using 'things we've seen once' as an estimator for 'things we've never seen' proved useful as key concept.

The most powerful unknown-word algorithms make use of information about how the word is spelled. For example, words that end in the letters are likely to be plural nouns (NNS), while words ending with -ed tend to be past participles (VBN). Words starting with capital letters are likely to be nouns. Four specific kinds of orthographic features: 3 inflectional endings (-ed, -s, -ing), 32 derivational endings (such as -ion, -al, -ive, and -ly), 4 values of capitalization (capitalized initial, capitalized non- initial, etc.), and hyphenation. They used the following equation to compute the likelihood of an unknown word:

P(wi|ti) = p(unknown-word|ti) p(capital|ti) p(endings/hyph |ti)

Other researchers, rather than relying on these hand-designed features, have used machine learning to induce useful features. Brill used the TBL algorithm, where the allowable templates were defined orthographically (the first N letters of the words, the last N letters of the word, etc). His algorithm induced all the English inflectional features, hyphenation, and many derivational features such as -ly, al. Franz uses a loglinear model which includes more features, such as the length of the word and various prefixes, and furthermore includes interaction terms among various features.

4. Introduction to CFG

A context-free grammar (CFG) is a list of rules that define the set of all well-formed sentences in a language. Each rule has a left-hand side, which identifies a syntactic category, and a right-hand side, which defines its alternative component parts, reading from left to right.

The word syntax comes from the Greek syntaxiś , meaning 'setting out together or arrangement', and refers to the way words are arranged together. various syntactic notions part-of-speech categories as a kind of equivalence class for words.. There are three main new ideas: constituency, grammatical relations, and subcategorization and dependencies. The fundamental idea of constituency is that groups of words may behave as a single unit or phrase, called a constituent. For example we will see that a group of words called a noun phrase often acts as a unit; noun phrases include single words like she or Michael and phrases like the house, Russian Hill, and a well-weathered three-story structure. context-free grammars, a formalism that will allow us to model these

constituency facts. Grammatical relations are a formalization of ideas from traditional grammar about SUBJECTS and OBJECTS. In the sentence: She ate a mammoth breakfast. The noun phrase She is the SUBJECT and a mammoth breakfast is the OBJECT. Subcategorization and dependency relations refer to certain kinds of relations between words and phrases.

For example the verb want can be followed by an infinitive, as in I want to fly to Detroit or a noun phrase, as in I want a flight to Detroit. But the verb find cannot be followed by an infinitive (*I found to fly to Dallas). These are called facts about the subcategory of the verb. All of these kinds of syntactic knowledge can be modelled by various kinds of grammars that are based on context-free grammars. Context-free grammars are the backbone of many models of the syntax of natural language (and, for that matter, of computer languages). As such they are integral to most models of natural language understanding, of grammar checking, and more recently of speech understanding. They are powerful enough to express sophisticated relations among the words in a sentence, yet computationally tractable enough that efficient algorithms exist for parsing sentences with them

Here are some properties of language that we would like to be able to analyze: Structure and Meaning–A sentence has a semantic structure that is critical to determine for many tasks. For instance, there is a big difference between Foxes eat rabbits and Rabbits eat foxes. Both describe some eating event, but in the first the fox is the eater, and in the second it is the one being eaten. We need some way to represent overall sentence structure that captures such relationships. Agreement–In English as in most languages there are constraints between forms of words that should be maintained, and which are useful in analysis for eliminating implausible interpretations. For instance, English requires number agreement between the subject and the verb (among other things). Thus we can say Foxes eat rabbits but "Foxes eats rabbits" is ill formed. In contrast, The Fox eats a rabbit is fine whereas "The fox eat a rabbit" is not. Recursion–Natural languages have a recursive structure that is important to capture. For instance, Foxes that eat chickens eat rabbits is best analyzed as having an embedded sentence (i.e. that eat chickens) modifying the noun phases Foxes. This one additional rule allows us to also interpret Foxes that eat chickens that eat corn eat rabbits and a host of other sentences. Long Distance Dependencies–Because of the recursion, word dependencies

like number agreement can be arbitrarily far apart from each other, e.g., "Foxes that eat chickens that eat corn eats rabbits" is awkward because the subject (foxes) is plural while the main verb (eats) is singular. In principle, we could find sentences that have an arbitrary number of words between the subject and its verb.

Noam Chomsky, in 1957, used the following notation called productions, to define the syntax of English. The terms used here, sentence, noun phrase,etc. plus the following rules describe a very small subset of English sentences. The articles a, and the have been categorized as adjectives for simplicity.

<sentence> → <noun phrase> <verb phrase>
<noun phrase> → <adjective> <noun phrase> | <adjective> <singular noun>
<verb phrase> → <singular verb> <adverb>
<adjective> → a | the |little
<singular noun> → boy
<singular verb> → ran
<adverb> → quickly

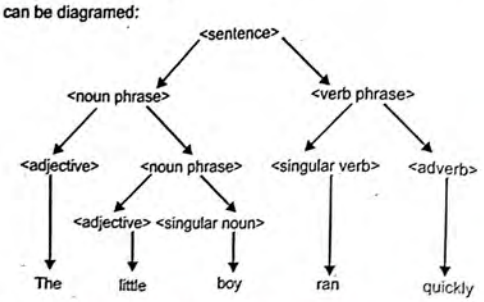
Here, the arrow, →, might be read as "is defined as" and the vertical bar, "|", as "or". Thus, a noun phrase is defined as an adjective followed by another noun phrase or as an adjective followed by a singular noun. This definition of noun phrase is recursive because noun phrase occurs on both sides of the production. Grammars are recursive to allow for infinite length strings. This grammar is said to be context-free because only one syntactic category, e.g., , occurs on the left of the arrow. If there were more than one syntactic category, this would describe a context and be called context-sensitive. A grammar is an example of a metalanguage- a language used to describe another language. Here, the metalanguage is the context-free grammar used to describe a part of the English language. Figure X shows a diagram called a parse tree or structure tree for the sentence the little boy ran quickly. The sentence: "quickly, the little boy ran" is also a syntactically correct sentence, but it cannot be derived from the above grammar.

EXAMPLE 2: Finding the structure of a sentence

Using the grammer above, the sentence:

The little boy ran quickly

can be diagramed:



In fact, it is impossible to describe all the correct English sentences using a context free grammar. On the other hand, it is possible, using the grammar above, to derive a syntactically correct, but semantically incorrect string, "little the boy ran quickly." Context free grammars cannot describe semantics.

E.g., the rule s --> np vp means that "a sentence is defined as a noun phrase followed by a verb phrase." Figure X1 shows a simple CFG that describes the sentences from a small subset of English.

```
s  → np vp
np → det n
vp → tv np
   → iv
det → the
    → a
    → an
n   → giraffe
    → apple
iv  → dreams
tv  → eats
    → dreams
```

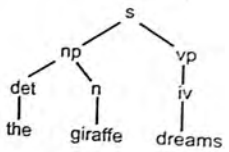


Figure 2: A grammar and a parse tree for "the giraffe dreams"

A sentence in the language defined by a CFG is a series of words that can be derived by systematically applying the rules, beginning with a rule that has s on its left-hand side. A parse of the sentence is a series of rule applications in which a syntactic category is replaced by the right-hand side of a rule that has that category on its left-hand side, and the final rule application yields the sentence itself. E.g., a parse of the sentence "the giraffe dreams" is: s => np vp => det n vp => the n vp => the giraffe vp => the giraffe iv => the giraffe dreams. A convenient way to describe a parse is to show its parse tree, which is simply a graphical display of the parse. Note that the root of every subtree has a grammatical category that appears on the left-hand side of a rule, and the children of that root are identical to the elements on the right-hand side of that rule.

Potential Problems in CFG

- Agreement
- Subcategorization
- Movement

Agreement

This dog	*This dogs
Those dogs	*those dog
This dog eats	*This dog eat
Those dogs eat	*Those dogs eats

5. Subcategorization

Sneeze: John sneezed

Find: Please find [a flight to NY]$_{NP}$

Give: Give [me]$_{NP}$ [a cheaper fare]$_{NP}$

Help: Can you help [me]$_{NP}$ [with a flight]$_{PP}$

Prefer: I prefer [to leave earlier]$_{TO\text{-}VP}$

Told: I was told [United has a flight]$_{S}$

...

*John sneezed the book

*I prefer United has a flight

*Give with a flight

Subcat expresses the econstraints that a predicate (verb for now) places on the number and type of the argument it want to take

So the various rules for VPs overgenerate.

They permit the presence of strings containing verbs and arguments that don't go together

For example

VP -> V NP therefore

Sneezed the book is a VP since "sneeze" is a verb and "the book" is a valid NP

Subcategorization frames can fix this problem ("slow down" overgeneration)

6. Movement

- Core example
  - $[[\text{My travel agent}]_{NP}\ [\text{booked}\ [\text{the flight}]_{NP}]_{VP}]_S$
- I.e. "book" is a straightforward transitive verb. It expects a single NP arg within the VP as an argument, and a single NP arg as the subject.
- What about?
  - Which flight do you want me to have the travel agent book?
- The direct object argument to "book" isn't appearing in the right place. It is in fact a long way from where its supposed to appear.
- And note that its separated from its verb by 2 other verbs.

Parsing With Context-free Grammar

Parse trees are directly useful in applications such as grammar check- ing in word-processing systems; a sentence which cannot be parsed may have grammatical errors (or at least be hard to read). In addition, parsing is an important intermediate stage of representation for semantic analysis, and thus plays an important role in applications like machine translation, question answering, and information extrac- tion. For example, in

order to answer the question "What books were written by British women authors before 1800? we'll want to know that the subject of the sentence was what books and that the by-adjunct was British women authors to help us figure out that the user wants a list of books (and not just a list of authors). Syntactic parsers are also used in lexicography applications for building online versions of dictionaries. Finally, stochastic versions of parsing algorithms have recently begun to be incorporated into speech recognizers, both for language models and for non-finite-state acoustic and phonotactic modeling.

In syntactic parsing, the parser can be viewed as searching through the space of all possible parse trees to find the correct parse tree for the sentence. The search space of possible parse trees is defined by the grammar. For example, consider the following sentence:

Book that flight. ...(1)

Using the miniature grammar and lexicon in Figure 2, which consists of some of the CFG rules for English, the correct parse tree that would be assigned to this example is shown in Figure 1

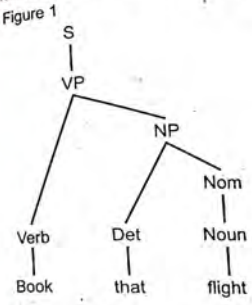


Figure 3. The correct parse tree for the sentence Book that flight according to the grammar in Figure 2.

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	$Nominal \rightarrow Nominal\ PP$

Figure 3: A miniature English grammar and lexicon.

The goal of a parsing search is to find all trees whose root is the start symbol S, which cover exactly the words in the input. Regardless of the search algorithm we choose, there are clearly two kinds of constraints that should help guide the search. One kind of constraint comes from the data, i.e. the input sentence itself. Whatever else is true of the final parse tree, we know that there must be three leaves, and they must be the words book, and flight. The second kind of constraint comes from the grammar. We know that whatever else is true of the final parse tree, it must have one root, which must be the start symbol S. These two constraints,, give rise to the two search strategies underlying most parsers: top-down or goal-directed search and bottom-up or data-directed search.

Top-Down Parsing

A top-down parser searches for a parse tree by trying to build from the root node S down to the leaves. Let's consider the search space that a top-down parser explores, assuming for the moment that it builds all possible trees in parallel. The algorithm starts by assuming the input can be derived by the designated start symbol S. The next step is to find the tops of all trees which can start with S, by looking for all the grammar rules with S on the left-hand side. In the grammar in Figure 2, there are three rules that expand S, so the second ply, or level, of the search space in Figure 3 has three partial trees. We next expand the constituents in these three new trees, just as we originally expanded S. The first tree tells us to expect an NP followed by a VP, the second expects

an Aux followed by an NP and a VP, and the third a VP by itself. To fit the search space on the page, we have shown in the third ply of Figure 3 only the trees resulting from the expansion of the left-most leaves of each tree. At each ply of the search space we use the right-hand- sides of the rules to provide new sets of expectations for the parser, which are then used to recursively generate the rest of the trees. Trees are grown downward until they eventually reach the part-of-speech categories at the bottom of the tree. At this point, trees whose leaves fail to match all the words in the input can be rejected, leaving behind those trees that represent successful parses. In Figure 3, only the 5th parse tree (the one which has expanded the rule VP ! Verb NP) will eventually match the input sentence Book that flight.

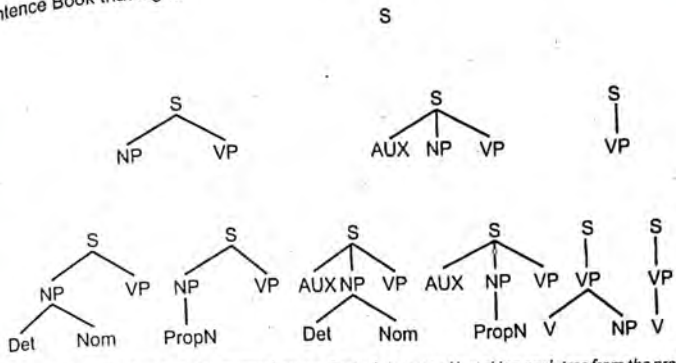


Figure 4: An expanding top-down search space. Each ply is created by taking each tree from the previous ply, replacing the leftmost non-terminal with each of its possible expansions, and collecting each of these trees into a new ply.

Bottom-Up Parsing

Bottom-up parsing is the earliest known parsing algorithm, and is used in the shift-reduce parsers common for computer languages. In bottom-up parsing, the parser starts with the words of the input, and tries to build trees from the words up, again by applying rules from the grammar one at a time. The parse is successful if the parser succeeds in building a tree rooted in the start symbol S that covers all of the input. Figure 4 show the bottom-up search space, beginning with the sentence Book that flight. The parser begins by looking up each word (book, that, and flight) in the lexicon and building three partial trees with the part of speech for each word. But the word book is ambiguous; it can be

a noun or a verb. Thus the parser must consider two possible sets of trees. The first two plies in Figure 4show this initial bifurcation of the search space.

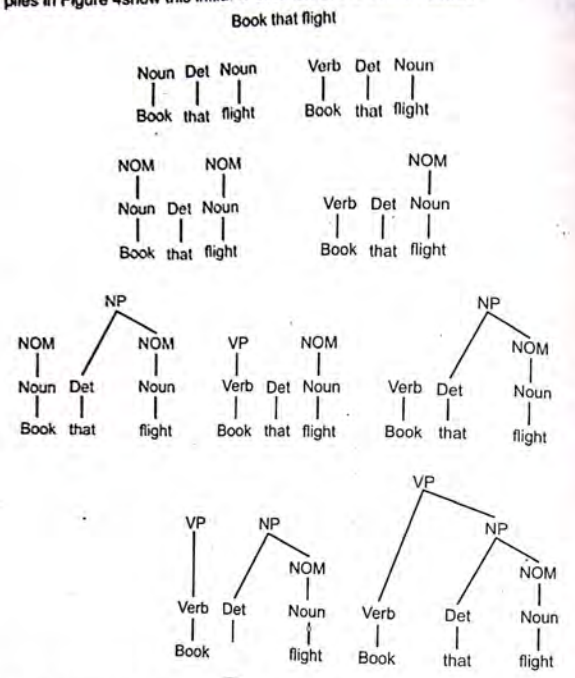


Figure 5: An expanding bottom-up search space for the sentence Book that flight. This figure does not show the final tier of the search with the correct parse tree Figure 1.

Each of the trees in the second ply are then expanded. In the parse on the left (the one in which book is incorrectly considered a noun), the Nominal! Noun rule is applied to both of the Nouns (book and flight). This same rule is also applied to the sole Noun (flight) on the right, producing the trees on the third ply. In general, the parser extends one ply to the next by looking for places in the parse-in-progress where the right-hand-side of some rule might fit. This contrasts with the earlier top-down parser, which expanded trees by applying rules when their left-hand side matched an unexpanded nonterminal. Thus in the fourth ply, in the first and third parse, the sequence Det Nominal is recognized as the right-hand side of the NP ! Det Nominal rule. In the fifth ply, the interpretation of book as a noun has been pruned from the search space. This is because this parse cannot be continued: there is no rule in the grammar with the right-hand side Nominal NP. The final ply of the search space is the correct parse tree.

Comparing Top-down and Bottom-up Parsing

Each of these two architectures has its own advantages and disadvantages. The top-down strategy never wastes time exploring trees that cannot result in an S, since it begins by generating just those trees. This means it also never explores subtrees that cannot find a place in some S-rooted tree. In the bottom-up strategy, by contrast, trees that have no hope of leading to an S, or fitting in with any of their neighbours, are generated with wild abandon. For example the left branch of the search space in Figure 4 is completely wasted effort; it is based on interpreting book as a Noun at the beginning of the sentence despite the fact no such tree can lead to an S given this grammar. The top-down approach has its own inefficiencies. While it does not waste time with trees that do not lead to an S, it does spend considerable effort on S trees that are not consistent with the input. Note that the first four of the six trees in the third ply in Figure 3 all have left branches that cannot match the word book. None of these trees could possibly be used in parsing this sentence. This weakness in top-down parsers arises from the fact that they can generate trees before ever examining the input. Bottom-up parsers, on the other hand, never suggest trees that are not at least locally grounded in the actual input. Neither of these approaches adequately exploits the constraints present by the grammar and the input words.

7. Sequence labeling

In natural language processing, it is a common task to extract words or phrases of particular types from a given sentence or paragraph. For example, when performing analysis of a corpus of news articles, we may want to know which countries are mentioned in the articles, and how many articles are related to each of these countries.

This is actually a special case of sequence labelling in NLP (others include POS tagging and Chunking), in which the goal is to assign a label to each member in the sequence. In the case of identifying country names, we would like to assign a 'country' label to words that form part of a country's name, and a 'irrelevant' label to all other words. For example, the following is a sentence broken down into tokens, and its desired output after the sequence labelling process:

input = ["Paris", "is", "the", "capital", "of", "France"]

output = ["I", "I", "I", "I", "I", "C"]

where I means that the token of that position is an irrelevant word, and C means that the token of that position is a word that form part of a country's name.

Methods of Sequence Labelling

A simple, though sometimes quite useful, approach is to prepare a dictionary of country names, and look for these names in each of the sentences in the corpus. However, this method relies heavily on the comprehensiveness of the dictionary. While there is a limited number of countries, in other cases such as city names the number of possible entries in the dictionary can be huge. Even for countries, many countries may be referred to using different sequence of characters in different contexts. For example, the United States of America may be referred to in an article as the USA, the States, or simply America.

In fact, a person reading a news article would usually recognise that a word or a phrase refers to a country, even when he or she has not seen the name of that country before. The reason is that there are many different cues in the sentence or the whole article that can be used to determine whether a word or a phrase is a country name. Take the following two sentences as examples:

Patrik travels to India capital, Delhi, on Monday for meetings of foreign ministers from the 10-member Association of South East Asia Nations (ASEAN).

The Governments of India and Srilanka will strengthen ties with a customs cooperation agreement to be in force on June 15th.

The first sentence implies that something called India has a capital, suggesting that India is a country. Similarly, in the second sentence we know that both India and Srilanka are countries as the news mentioned about their governments. In other words, the words around Delhi, India and Srilanka provide clues as to whether they are country names

Hidden Markov Model (HMM)

Markov Models

Let's talk about the weather Here in Mumbai we have three types of weather sunny rainy and foggy. Let's assume for the moment that the weather lasts all day ie it doesnt change from rainy to sunny in the middle of the day. Weather prediction is all about trying to guess what the weather will be like tomorrow based on a history of observations of weather. Let's assume a simplified model of weather prediction well collect statistics on what the weather was like today based on what the weather was like yesterday the day before and so forth. We want to collect the following probabilities:

$P(w_n | w_{n-1}, w_{n-2} \ldots . w_1)$ ..(1)

Using expression 1 we can give probabilities of types of weather for tomorrow and the next day using n days of history For example if we knew that the weather for the past three days was{ sunny, sunny, foggy} in chronological order the probability that tomorrow would be rainy is given by

$P(w_4 = Rainy | w_3 = Foggy, w_2 = Sunny, w_1 = sunny)$ ..(2)

Here's the problem the larger n is the more statistics we must collect. Suppose that n =5 then we must collect statistics for $3^5 = 243$ past histories Therefore we will make a simplifying assumption called the Markov Assumption

In a sequence $\{w_1, w_2, \ldots . w_n\}$:

$$P(w_n | w_{n-1}, wn-_2 \ldots . w_1) \approx P(w_n | w_{n-1})..(3)$$

This is called a first-order Markov assumption since we say that the probability of an observation at time n only depends on the observation at time n A second-order Markov assumption would have the observation at time n depend on n-1 and n-2. In general when people talk about Markov assumptions they usually mean first-order Markov assumptions. We will use the two terms interchangeably

We can express the joint probability using the Markov assumption

$P(w_1 \ldots . w_n) = \Pi_{i=1}^{n} P(w_i | w_{i-1})$

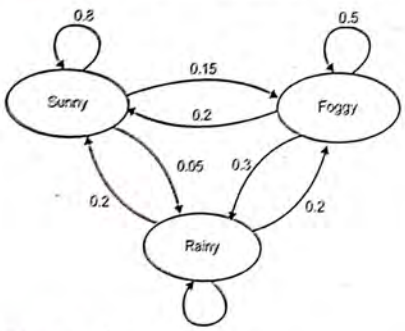
So this now has a profound aect on the number of histories that we have to nd statistics for we now only need $3^2 = 9$ numbers to characterize the probabilities of all of the sequences This assumption may or may not be a valid assumption depending on the situation(in the case of weather it's probably not valid) but we use these to simplify the situation

So let's arbitrarily pick some numbers for $P(w_{\text{tomorrow}} \mid w_{\text{today}})$ expressed in Table D1

Table D1 : Probabilities of Tomorrows weather based on Todays Weather

		Tomorrow's weather		
		Sunny	Rainy	Foggy
Today's weather	Sunny	0.8	0.05	0.15
	Rainy	0.2	0.6	0.2
	Foggy	0.2	0.3	0.5

For first-order Markov models we can use these probabilities to draw a probabilistic finite state automaton For the weather domain you would have three states Sunny Rainy and Foggy and every day you would transition to a possibly new state based on the probabilities in Table D1 Such an automaton would look like this



Hidden Markov Models

So what makes a Hidden Markov Model. Well suppose you were locked in a room for several days and you were asked about the weather outside. The only piece of evidence you have is whether the person who comes into the room carrying your daily meal is carrying an umbrella or not. Let's suppose the following probabilities

Table D2 : Probabilities of Seeing an Umbrella Based on the Weather

	Probability of Umbrella
Suuny	0.1
Rainy	0.8
Foggy	0.3

Remember the equation for the weather Markov process before you were locked in the room was

$$P(w_1,.....w_n)=\Pi_{i=1}^{n} P(w_i|w_{i-1}) \quad ..(5)$$

Now we have to factor in the fact that the actual weather is hidden from you. We do that by using Bayes Rule

$$P(w_1,.....w_n|u_1,.....u_n)=\frac{P(u_1,.....u_n|w_1,.....w_n)P(w_1,.....w_n)}{P(u_1,.....u_n)} \quad .(6)$$

where u_i is true if your caretaker brought an umbrella on day i and false if the caretaker didn't The probability $P(w_1,.....w_n)$ is the same as the Markov model from the last section and the probability $P(u_1,.....u_n)$ is the prior probability of seeing a particular sequence of umbrella events (eg {True, False, True}).

The probability $P(u_1,.....u_n|w_1,.....w_n)$ can be estimated as $\Pi_{i=1}^{n} P(u_i|w_i)$, if you assume that for all i given w_i, u_i is independent of all u_j and w_j for all $j \neq i$.

HMM for POS tagging

A Markov model is a stochastic (probabilistic) model used to represent a system where future states depend only on the current state. For the purposes of POS tagging, we make the simplifying assumption that we can represent the Markov model using a finite state transition network.

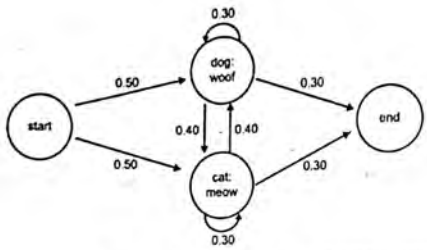


Figure 6: A finite state transition network representing a Markov model

Each of the nodes in the finite state transition network represents a state and each of the directed edges leaving the nodes represents a possible transition from that state to another state. Note that each edge is labeled with a number representing the probability that a given transition will happen at the current state. Note also that the probability of transitions out of any given state always sums to 1.

In the finite state transition network pictured above, each state was observable. We are able to see how often a cat meows after a dog woofs. What if our cat and dog were bilingual. That is, what if both the cat and the dog can meow and woof? Furthermore, let's assume that we are given the states of dog and cat and we want to predict the sequence of meows and woofs from the states. In this case, we can only observe the dog and the cat but we need to predict the unobserved meows and woofs that follow. The meows and woofs are the hidden states.

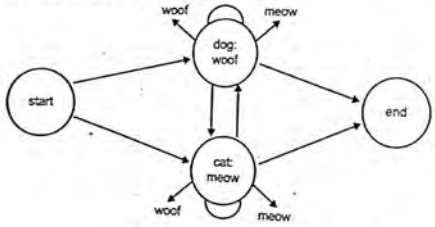


Figure 7: A finite state transition network representing an HMM

The figure above is a finite state transition network that represents our HMM. The black arrows represent emissions of the unobserved states woof and meow.

Let's now take a look at how we can calculate the transition and emission probabilities of our states. Going back to the cat and dog example, suppose we observed the following two state sequences:

dog, cat, cat

dog, dog, dog

Then the transition probabilities can be calculated using the maximum likelihood estimate:

$$P(s_i \mid s_{i-1}) = \frac{C(s_i, s_{i-1})}{C(s_{i-1})}$$

In English, this says that the transition probability from state i-1 to state i is given by the total number of times we observe state i-1 transitioning to state i divided by the total number of times we observe state i-1.

For example, from the state sequences we can see that the sequences always start with dog. Thus we are at the start state twice and both times we get to dog and never cat. Hence the transition probability from the start state to dog is 1 and from the start state to cat is 0.

Let's try one more. Let's calculate the transition probability of going from the state dog to the state end. From our example state sequences, we see that dog only transitions to the end state once. We also see that there are four observed instances of dog. Thus the transition probability of going from the dog state to the end state is 0.25. The other transition probabilities can be calculated in a similar fashion.

The emission probabilities can also be calculated using maximum likelihood estimates:

$$P(t_i \mid s_i) = \frac{C(t_i, s_i)}{C(s_i)}$$

In English, this says that the emission probability of tag i given state i is the total number of times we observe state i emitting tag i divided by the total number of times we observe state i.

Let's calculate the emission probability of dog emitting woof given the following emissions for our two state sequences above:

woof, woof, meow

meow, woof, woof

That is, for the first state sequence, dog woofs then cat woofs and finally cat meows. We see from the state sequences that dog is observed four times and we can see from the emissions that dog woofs three times. Thus the emission probability of woof given that we are in the dog state is 0.75. The other emission probabilities can be calculated in the same way. For completeness, the completed finite state transition network is given here:

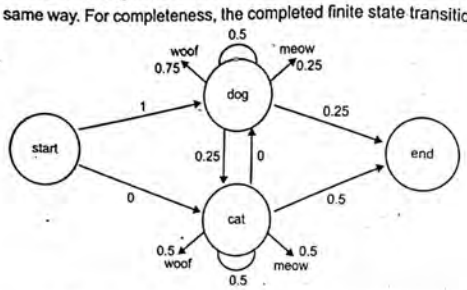


Figure 8: Finite state transition network of the hidden Markov model of our example.

So how do we use HMMs for POS tagging? When we are performing POS tagging, our goal is to find the sequence of tags T such that given a sequence of words W we get

$$T = \underset{\bar{T}}{\operatorname{argmax}} P(\bar{T} \mid W)$$

In English, we are saying that we want to find the sequence of POS tags with the highest probability given a sequence of words. As tag emissions are unobserved in our hidden Markov model, we apply Baye's rule to change this probability to an equation we can compute using maximum likelihood estimates:

$$T = \underset{\bar{T}}{\operatorname{argmax}} P(\bar{T} \mid W) = \underset{\bar{T}}{\operatorname{argmax}} \frac{P(W \mid \bar{T}) P(\bar{T})}{P(W)} \alpha \underset{\bar{T}}{\operatorname{argmax}} P(w \mid \bar{T}) P(\bar{T})$$

The second equals is where we apply Baye's rule. The symbol that looks like an infinity symbol with a piece chopped off means proportional to. This is because P(W) is a constant for our purposes since changing the sequence T does not change the probability P(W). Thus dropping it will not make a difference in the final sequence T that maximizes the probability.

In English, the probability P(W|T) is the probability that we get the sequence of words given the sequence of tags. To be able to calculate this we still need to make a simplifying assumption. We need to assume that the probability of a word appearing depends only on its own tag and not on context. That is, the word does not depend on neighboring tags and words. Then we have

$$P(W \mid \bar{T}) \approx \prod_{i=1}^{n} P(w_i \mid \bar{t}_i)$$

In English, the probability P(T) is the probability of getting the sequence of tags T. To calculate this probability we also need to make a simplifying assumption. This assumption gives our bigram HMM its name and so it is often called the bigram assumption. We must assume that the probability of getting a tag depends only on the previous tag and no other tags. Then we can calculate P(T) as

$$P(\bar{T}) = \prod_{i=1}^{n} P(\bar{t}_i \mid \bar{t}_{i-1})$$

Note that we could use the trigram assumption, that is that a given tag depends on the two tags that came before it. As it turns out, calculating trigram probabilities for the HMM requires a lot more work than calculating bigram probabilities due to the smoothing required. Trigram models do yield some performance benefits over bigram models but for simplicity's sake we use the bigram assumption.

Finally, we are now able to find the best tag sequence using

$$T = \underset{\bar{T}}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i \mid \bar{t}_i) P(\bar{t}_i \mid \bar{t}_{i-1})$$

The probabilities in this equation should look familiar since they are the emission probability and transition probability respectively.

$P(w_i \mid \bar{t}_i)$ Emission probability

$P(\bar{t}_i \mid \bar{t}_{i-1})$ Transition probability

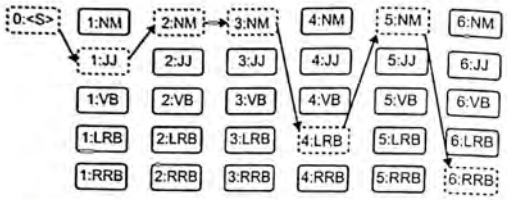
Hence if we were to draw a finite state transition network for this HMM, the observed states would be the tags and the words would be the emitted states similar to our woof and meow example. We have already seen that we can use the maximum likelihood estimates to calculate these probabilities.

Given a dataset consisting of sentences that are tagged with their corresponding POS tags, training the HMM is as easy as calculating the emission and transition probabilities as described above. For an example implementation, check out the bigram model as

implemented here. The basic idea of this implementation is that it primarily keeps track of the values required for maximum likelihood estimation during training. The model then calculates the probabilities on the fly during evaluation using the counts collected during training.

An astute reader would wonder what the model does in the face of words it did not see during training. We return to this topic of handling unknown words later as we will see that it is vital to the performance of the model to be able to handle unknown words properly.

Viterbi Decoding



The HMM gives us probabilities but what we want is the actual sequence of tags. We need an algorithm that can give us the tag sequence with highest probability of being correct given a sequence of words. An intuitive algorithm for doing this, known as greedy decoding, goes and chooses the tag with the highest probability for each word without considering context such as subsequent tags. As we know, greedy algorithms don't always return the optimal solution and indeed it returns a sub-optimal solution in the case of POS tagging. This is because after a tag is chosen for the current word, the possible tags for the next word may be limited and sub-optimal leading to an overall sub-optimal solution.

We instead use the dynamic programming algorithm called Viterbi. Viterbi starts by creating two tables. The first table is used to keep track of the maximum sequence probability that it takes to reach a given cell. If this doesn't make sense yet that is okay. We will take a look at an example. The second table is used to keep track of the actual path that led to the probability in a given cell in the first table.

Let's look at an example to help this settle in. Returning to our previous woof and meow example, given the sequence

meow woof

we will use Viterbi to find the most likely sequence of states that led to this sequence. First we need to create our first Viterbi table. We need a row for every state in our finite state transition network. Thus our table has 4 rows for the states start, dog, cat and end. We are trying to decode a sequence of length two so we need four columns. In general, the number of columns we need is the length of the sequence we are trying to decode. The reason we need four columns is because the full sequence we are trying to decode is actually

<start> meow woof <end>

The first table consists of the probabilities of getting to a given state from previous states. More precisely, the value in each cell of the table is given by

a_{ij} transition probability from state s_i to s_j

b_{jt} emission probability of word w_t at state s_j

$$v_t(j) = \max_{i=1}^{n} v_{t-1}(i) a_{ij} b_{jt}$$

Let's fill out the table for our example using the probabilities we calculated for the finite state transition network of the HMM model.

	<start>	meow	woof	<end>
Start	1			
dog	0			
Cat	0			
End	0			

Notice that the first column has 0 everywhere except for the start row. This is because the sequences for our example always start with <start>. From our finite state transition network, we see that the start state transitions to the dog state with probability 1 and never goes to the cat state. We also see that dog emits meow with a probability of 0.25. It is also important to note that we cannot get to the start state or end state from the start state.

Thus we get the next column of values

	<start>	meow	woof	<end>
Start	1	0		
dog	0	1*1*0.25=0.25		
Cat	0	0		
End	0	0		

Notice that the probabilities of all the states we can't get to from our start state are 0. Also, the probability of getting to the dog state for the meow column is 1 * 1 * 0.25 where the first 1 is the previous cell's probability, the second 1 is the transition probability from the previous state to the dog state and 0.25 is the emission probability of meow from the current state dog. Thus 0.25 is the maximum sequence probability so far. Continuing onto the next column:

	<start>	meow	woof	<end>
Start	1	0	0	
dog	0	1*1*0.25=0.25	0.25*0.5*0.75=0.09375	
Cat	0	0	0.25*0.25*0.5=0.03125	
End	0	0	0	

Observe that we cannot get to the start state from the dog state and the end state never emits woof so both of these rows get 0 probability. Meanwhile, the cells for the dog and cat state get the probabilities 0.09375 and 0.03125 calculated in the same way as we saw before with the previous cell's probability of 0.25 multiplied by the respective transition and emission probabilities. Finally, we get

	<start>	meow	woof	<end>
Start	1	0	0	
dog	0	1*1*0.25=0.25	0.25*0.5*0.75=0.09375	
Cat	0	0	0.25*0.25*0.5=0.03125	
End	0	0	0	0.09375*0.25=0.0234375 0.03125*0.5=0.015625

At this point, both cat and dog can get to <end>. Thus we must calculate the probabilities of getting to end from both cat and dog and then take the path with higher probability.

	<start>	meow	woof	<end>
Start	1	0	0	0
dog	0	1*1*0.25=0.25	0.25*0.5*0.75=0.09375	0
Cat	0	0	0.25*0.25*0.5=0.03125	0
End	0	0	0	0.09375*0.25=0.0234375 ~~0.03125*0.5=0.015625~~

Going from dog to end has a higher probability than going from cat to end so that is the path we take. Thus the answer we get should be

<start> dog dog <end>

In a Viterbi implementation, the whole time we are filling out the probability table another table known as the back pointer table should also be filled out. The value of each cell in the back pointer table is equal to the row index of the previous state that led to the maximum probability of the current state. Thus in our example, the end state cell in the back pointer table will have the value of 1 (0 starting index) since the state dog at row 1 is the previous state that gave the end state the highest probability. For completeness, the back pointer table for our example is given below.

	<start>	meow	woof	<end>
Start	-1			
dog		0	1	
Cat			1	
End				1

Note that the start state has a value of -1. This is the stopping condition we use for when we trace the backpointer table backwards to get the path that provides us the sequence with the highest probability of being correct given our HMM. To get the state sequence <start> dog dog <end>, we start at the end cell on the bottom right of the table. The 1 in this cell tells us that the previous state in the woof column is at row 1 hence the previous state must be dog.

From dog, we see that the cell is labelled 1 again so the previous state in the meow column before dog is also dog. Finally, in the meow column, we see that the dog cell is labelled 0 so the previous state must be row 0 which is the <start> state. We see -1 so we stop here. Our sequence is then <end> dog dog <start>. Reversing this gives us our most likely sequence.

Maximum Entropy

Maximum entropy modelling is a framework for integrating information from many heterogeneous information sources for classification. The term maximum entropy refers to an optimization framework in which the goal is to find the probability model that maximizes entropy over the set of models that are consistent with the observed evidence.

Maximum Entropy model is used to predict observations from training data. This does not uniquely identify the model but chooses the model which has the most uniform distribution i.e. the model with the maximum entropy. Entropy is a measure of uncertainty of a distribution, the higher the entropy the more uncertain a distribution is. Entropy measures uniformity of a distribution but applies to distributions in general.

The Principle of Maximum Entropy argues that the best probability model for the data is the one which maximizes entropy, over the set of probability distributions that are consistent with the evidence.

Maximum Entropy: A simple example

The following example illustrates the use of maximum entropy on a very simple problem.

- Model an expert translator's decisions concerning the proper French rendering of the English word ***on***
- A model(p) of the expert's decisions assigns to each French word or phrase(f) an estimate, p(f), of the probability that the expert would choose f as a translation of ***on***. Our goal is to extract a set of facts about the decision-making process from the sample and construct a model of this process

A clue from the sample is the list of allowed translations

on **---> {sur, dans, par, au bord de}**

With this information in hand, we can impose our first constraint on p:

p(sur) + p(dans) + p(par) + p(au bord de) =1

The most uniform model will divide the probability values equally. Suppose we notice that the expert chose either dans or sur 30% of the time, then a second constraint can be added.

p(dans) + p(sur) = 3/10

- Intuitive Principle: Model all that is known and assume nothing about that which is unknown

A random process which produces an output value y, a member of a finite set Y.

y ∈ {sur, dans, par, au bord de}

The process may be influenced by some contextual information x, a member of a finite set X. x could include the words in the English sentence surrounding ***on.***

maximum entropy framework can be used to solve various problems in the domain of natural language processing like sentence boundary detection, part-of-speech tagging, prepositional phrase attachment, natural language parsing, and text categorization .

Conditional Random Field (CRF)

In POS tagging, the goal is to label a sentence (a sequence of words or tokens) with tags like ADJECTIVE, NOUN, PREPOSITION, VERB, ADVERB, ARTICLE.

For example, given the sentence "Bob drank coffee at Starbucks", the labeling might be "Bob (NOUN) drank (VERB) coffee (NOUN) at (PREPOSITION) Starbucks (NOUN)".

So let's build a conditional random field to label sentences with their parts of speech. Just like any classifier, we'll first need to decide on a set of feature functions fi.

Feature Functions in a CRF

In a CRF, each feature function is a function that takes in as input:

a sentence s

the position i of a word in the sentence

the label l_i of the current word

the label l_{i-1} of the previous word

and outputs a real-valued number (though the numbers are often just either 0 or 1).

For example, one possible feature function could measure how much we suspect that the current word should be labeled as an adjective given that the previous word is "very".

Features to Probabilities

Next, assign each feature function f_j a weight λ_j. Given a sentence s, we can now score a labelling l of s by adding up the weighted features over all the words in the sentence:

$$score(l|s) = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_i, l_{i-1})$$

The first sum runs over each feature function j, and the inner sum runs over each position i of the sentence.

Finally, we can transform these scores into probabilities p(l|s) between 0 and 1 by exponentiating and normalizing

$$p(l|s) = \frac{\exp[score(l|s)]}{\sum_{l'} \exp[score(l'|s)]} = \frac{\exp\left[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_i, l_{i-1})\right]}{\sum_{l'} \exp\left[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l'_i, l'_{i-1})\right]}$$

Example Feature Functions

So what do these feature functions look like? Examples of POS tagging features could include:

f1(s,i,li,li−1)=1 if li= ADVERB and the ith word ends in "-ly"; 0 otherwise. If the weight λ1 associated with this feature is large and positive, then this feature is essentially saying that we prefer labellings where words ending in -ly get labeled as ADVERB.

f2(s,i,li,li−1)=1 if i=1, li= VERB, and the sentence ends in a question mark; 0 otherwise. Again, if the weight λ2 associated with this feature is large and positive, then labelings that assign VERB to the first word in a question (e.g., "Is this a sentence beginning with a verb?") are preferred.

f3(s,i,li,li−1)=1 if li−1= ADJECTIVE and li= NOUN; 0 otherwise. Again, a positive weight for this feature means that adjectives tend to be followed by nouns.

f4(s,i,li,li−1)=1 if li−1= PREPOSITION and li= PREPOSITION. A negative weight λ4 for this function would mean that prepositions don't tend to follow prepositions, so we should avoid labelings where this happens.

to build a conditional random field, you just define a bunch of feature functions (which can depend on the entire sentence, a current position, and nearby labels), assign them weights, and add them all together, transforming at the end to a probability if necessary.

CRF vs HMM

CRFs can define a much larger set of features. Whereas HMMs are necessarily local in nature (because they're constrained to binary transition and emission feature functions, which force each word to depend only on the current label and each label to depend only on the previous label), CRFs can use more global features. For example, one of the features in our POS tagger above increased the probability of labelings that tagged the first word of a sentence as a VERB if the end of the sentence contained a question mark.

CRFs can have arbitrary weights. Whereas the probabilities of an HMM must satisfy certain constraints

Expected Questions

1. What is POS tagging? Explain types of word classes in English NL. Also comment on possible tag sets available in ENGLISH NL.
2. Explain open and closed word classes in English Language. Comment on possible tag sets available in ENGLISH NL. Show how the tags are assigned to the words of the following sentence:
   "Time flies like an arrow."
3. Why POS tagging is hard? Discuss possible challenges to be faced while performing POS tagging.
4. Discuss various approaches/algorithms to perform POS tagging.
5. Explain in detail Rule based POS tagging/ Stochastic (HMM) POS tagging/ Hybrid POS tagging.
6. Explain transformation-based POS tagging with suitable example.
7. What do you mean by constituency? Explain following key constituents' with suitable example w.r.t English language: 1: Noun phrases 2) Verb phrases 3) Prepositional phrases .

8. Explain CFG with suitable example . Discuss the following potential problems in CFG such as 1) Agreement 2) Sub categorization 3) Movement .
9. What is parsing? Explain Top-down & Bottom-up approach of parsing with suitable example.
10. W.r.t following Grammar show Shift reduce parsing of following sentences
    Book that flight
    Does that flight include meal
    S → NP VP S → Aux NP VP S → VP NP → Det NOM
    NOM → Noun NOM → Noun NOM VP → Verb
    VP → Verb NP
    Det → that | this | a | the Noun → book | flight | meal | man
    Verb → book | include | read Aux → does
11. Compare between Top-down & Bottom-up parsing approach.

4 Semantic Analysis

OBJECTIVES

After reading this chapter, the student will be able to understand :

- Lexical semantics
- Attachment for fragment of English
- Relations among lexemes and their senses
- WordNet
- Robust Word Sense Disambiguation (WSD), Dictionary based approach

SEMANTIC ANALYSIS

The process whereby meaning representations are composed and assigned to linguistic inputs

Semantics involves figure out the meaning of linguistic input (construct meaning representations) and process language to produce common-sense knowledge about the world (extract data and construct models of the world)

Semantics is associated with the meaning of language. General idea of semantic interpretation is to take natural language sentences or utterances and map the monitor some representation off meaning semantic analysis is concerned with creating representations for the meaning of linguistic inputs this chapter deals with the meaning of written text we can divide semantics into two parts as follows:

- The study of meaning of individual words (lexical semantics) and
- the study of how individual words combine to give meaning to a sentence (or larger units)

Once we have the meaning of the words we need to combine them into meaning of the whole sentence the principle of semantic compositionality sometimes called freg's principle states that the meaning of the whole sentence is comprised of the meaning of its parts that is the meaning of the sentence can be composed from the meaning of its constituent words. Natural languages do not obey this principle of compositionality. Often, the meaning of the whole is only partially dependent on the meaning of its constituents (as in collocates) and sometimes entirely different from the meanings of individuals (e.g., in idioms).

Hence, this decomposition of semantics may not be realistic. Word meaning is just one component of semantics. The relationships that exist between words, the domain, the word order, the semantic structure, the underlying context and the real-world knowledge, all contribute to the meaning of a sentence. Still, compositional and lexical word semantics remain the dominant approach to semantics in compositional linguistics. There are theories supporting the idea that linguistic knowledge is knowledge about words. This means that the lexicon contains all the knowledge about language.

These theories completely dispense with grammar as an independent entity which has been accepted for so many years. The idea that Syntax and lexicon cannot be described adequately without reference to each other is gaining increasing support from the results of Corpus analysis which attempts to investigate the role of contexts in syntax and semantics.

Lexical semantics has been the starting point for all the early theories of semantics. The most common paradigm involves decomposing lexical meanings in terms of semantic primitives or atomic units of meaning. However, this theory turns out to be inadequate in handling compositional semantics. This has led to the development of model theoretic semantics which, along with the structural semantics, is the dominant approach to semantics within linguistics. Model theoretic semantics is inspired by the semantics that logicians used for formal logical languages. Logicians focus on logical words (and, or, not, if, all, some, only, etc.) and do not pay attention to non-logical words (most nouns, verbs and adjectives). The primary advantage of this theory is its ability to explain compositional semantics - how the meaning of a sentence is determined from the meaning of its parts. One important aspect off meaning is that it related sentences to the outside world. However, we do not know the world is. A model theoretic approach to semantics attempt to create a model of the world and determine the truth of a sentence using this model. In this theory, the truth of a sentence does not mean that sentence is actually true; it simply means that the sentence is true in the world being modelled. Model theoretic semantics is effective in studying pragmatic as well as semantics.

Among the source of knowledge typically used are

- the meanings of words,
- the meanings associated with grammatical structures,
- knowledge about the structure of the discourse,
- Knowledge about the context in which the discourse is occurring, and
- Common-sense knowledge about the topic at hand

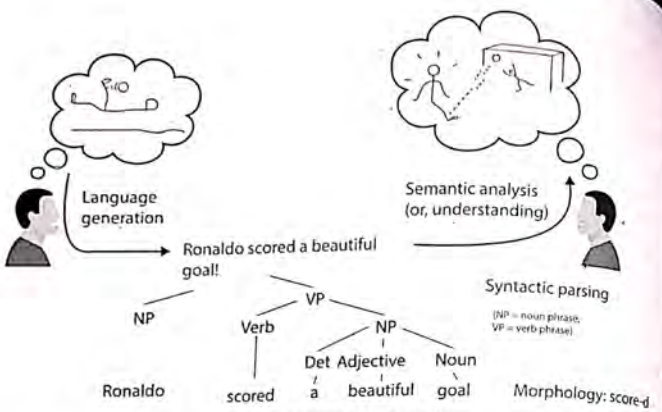


Figure 1: Understanding Semantics

Syntax-driven semantic analysis

Assigning meaning representations to input based solely on static knowledge from the lexicon and the grammar.

1. Lexical semantics

So far, we have focused on the creation of meaning representation of whole sentence. The approach we took supports the view that words contribute to the meaning of a sentence but do not themselves have meaning. It is perhaps this view that has led some to consider the lexicon as a simple list of words. However, this is narrow concept, too far from reality. Words have meanings; they have internal structure, and are involved in different relationship with other words. All this information can be captured in a systematic structure by lexicon. In this section, we focus on this and other issues related to words. More precisely, we focus on lexical semantics, which is concerned with the linguistic study of systematic, meaning related structure of words or lexemes (the minimal unit and lexicon). It involves meanings of component words

Compositional semantics

It involves how words combine to form larger meanings. So, if knowledge of each word's meaning isn't enough for comprehension of a sentence or phrase as it is meant to be understood, how is the overall compositional meaning derived? Obviously the overall meaning must have something to do with the meanings of those words contained within the phrase. "I don't eat" and "I don't drink", for example, express different ideas because of the difference in lexical meaning between "eat" and "drink." However, syntax, or the way in which the sentence is constructed , plays a role as well, as demonstrated below: 1. I like you. 2. You like me. Both sentences express totally different propositions, however they have the same words and each word has a clearly understood meaning. How is it that the meaning has changed? In sentence 1, "I" = subject, "you" = object, while in the second sentence these are reversed. This means that overall meaning relies not only on the meaning of each part, but additionally on syntactic composition. This premise is known as the principle of compositionality. All languages contain an infinite number of word combinations, so memorization of each separate phrasal meaning is impossible. This means that in order to understand the meanings of new phrases, one must rely on individual word meanings combined with the specific syntactic structure. The roles played by lexical and compositional semantics are equally necessary with regard to total understanding of a phrase or sentence. Knowledge of one without the other will invariably lead to miscommunication, and an understanding of denotation, connotation, and syntactical structure is necessary for compositional understanding of the whole. Understanding the difference between these terms and how the ideas they represent interact is paramount to understanding the meaning of a phrase or sentence.

What is language understanding

What exactly is meant by "understanding" an utterance?

- Extracting knowledge about the world from the utterance–in the case of concrete, physical things: ultimately translating linguistic input to physical and geometrical terms... uh huh
- "Understanding language means knowing how to use it.
- To master a technique. At matriculation examination: the student is presented with a portion of text or speech and then asked questions about it

- It is difficult to give one generic definition
- Thinking of a concrete application of semantic analysis is probably the best way t defining the problem

Semantic analysis vs. other areas of natural language processing

Phonetics: the study of linguistic sounds

Morphology: the study of the meaning components of words

scored = score –d = Verb score + past tense

employ, employee, employment, ...

Syntax: a study of the structural relationship between words

Semantics: the study of meaning

Pragmatics: the study of how language is used to accomplish goals; discourse conventions (turn taking, politeness, etc.); relation between language and context-d use

Semantic analysis often requires syntactic parsing, pragmatics etc. too (in some form not necessarily formal linguistics)

Approaches to semantic analysis

Predicate logic

The sentence "a restaurant that serves Chinese food near TUT" corresponds to the meaning representation $\exists x$ Restaurant(x)$\wedge$Serves(x, ChineseFood)$\wedge$Near(LocationOf(x), LocationOf(TUT) semantic analysis is equivalent to creating meaning representations from language.

scalability problem (large vocabulary or unrestricted domain)

Statistical approach

statistical machine translation (as an example)

find a bilingual database (e.g. parliamentary proceedings in two languages)

learn an alignment: words and phrases that correspond to each other

learn word order in the target language (probabilities of target word strings)

translated by matching source fragments against a database of real examples, identifying the corresponding translation fragments, and then recombining these to give the target text

Information retrieval

Google solves a certain part of the problem in a statistical way: answers to "trivial" kind of questions can be located using a web search engine assumes a database (Internet) and a clever page ranking system

Domain knowledge driven analysis

expect certain "slots" of information to be filled in football example in the beginning: hearer is aware of missing details and may expect to hear them another example: booking a flight restricting to a certain domain allows the use of specific patterns, rules, expectations, etc. customer at a restaurant buying train tickets

Applications of semantic analysis

Information extraction : extract small amounts of pertinent information from large bodies of text to find an answer to a question for example

Text summarization : Information retrieval (cf. Google) and document classification

Machine translation

Human-computer interaction : conversational agents: book plane tickets, query for a restaurant

Expert systems : free help: "please show me how to widen the margins of my document"

Surveillance

Several well-defined problems and applications yet unsolved

List on the right is from Jim Gray s Turing talk: "What next? A few remaining problems in IT"

Why is semantic analysis difficult?

Ambiguity of language

"I made her duck", for example, could mean

I cooked waterfowl for her.

I created the (plaster?) duck she owns.

I caused her to quickly lower her head or body.

I waved my magic wand and turned her into undifferentiated waterfowl.

Commonsense knowledge is typically omitted from social communications examples "Laura hid George's car keys. He was drunk."

Language understanding often requires unsound inference abduction ((A ⊃ B) and B) infer A (which is not sound logic)

Language is dynamic: allows defining new terms, allegory, etc.

Why is semantic analysis important?

Power of language: transfer thoughts from a head to another transfer between brains and a computer as well?

Language is a very generic representation (the most generic?)

- words can describe almost anything
- ability to reason with language ability to reason about almost anything (assuming the ability to construct a model of the world, too)

2. Attachment for Fragment of English

Words in a sentence are not tied together as a sequence of part of speech. Language puts constraints on word order. For example, certain words both together with each other more than with others, and to behave as a unit. The fundamental idea of Syntax is that words grouped together to form constituents (often termed phrases), each of which acts as a single unit. They combine with other constituents to form larger constituents, and eventually, a sentence. *The bird, The rain, The Wimbledon Court, The beautiful garden* are all noun phrases that can occur in the same syntactic context. For example, they can all function as the subject or the object of a verb.

These constituents combine with others to form a sentence constituent. For example, the noun phrase *the bird* can combine with the verb phrase *flies* to form the sentence *the bird flies*. Different types of phrases have different internal structures. In this section, we discuss some of the major phrase types and try to build phrase structure rules to identify them.

Phrase level Constructions

As discussed earlier, a fundamental notion in natural language is that certain groups of words behave as constituents. These constituents are identified by their ability to occur in similar context. One of the simplest ways to decide whether a group of words is a phrase, is to see if it can be substituted with some other group of words without changing the meaning. If such a substitution is possible in the set of words forms a face. This is called the substitution test. Consider sentence below where we can substitute a number of phrases:

Henna reads a book.

Henna reads storybook.

Those girls read a book.

She read the comic book.

We can easily identify the constituents that can be replaced for each other in the sentences. These are *Henna*, *she*, and *Those girls* and *a book*, *a story book*, and *a comic book*. These are the words that form a phrase. In linguistics, such constituents represent a paradigmatic relationship. Elements that can substitute each other in certain syntactic positions are said to be members of one paradigm.

Phrase types are named after their head, which is a lexical category that determines the properties of the phrase. Thus, if the head is a noun, the phrase is called a noun phrase, if head is a verb, then the phrase is a verb phrase and so on for other lexical categories such as adjective and prepositions. Figure below shows the sentences with the noun phrase verb phrase and preposition phrase.

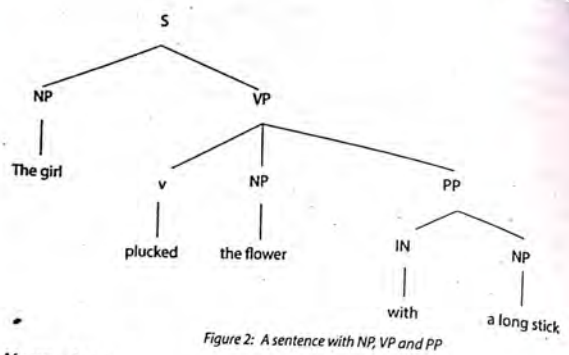


Figure 2: A sentence with NP, VP and PP

Noun phrase

A noun phrase is a phrase whose head is a noun or a pronoun, optionally accompanied by a set of modifiers. It can function as a subject, object or compliment. The modifiers of a noun phrase can we determine hours or adjective phrase. The obligatory constituent of a noun phrase is the noun head-all other constituents are optional. These structures can be represented using the phrase structure rule. As discussed earlier, phrase structure rules are of the form A →BC, which states that constituent A can be rewritten as two constitutes B and C. These rules specify the element can occur in a phrase and in what order. Using this notation, we can represent the phrase structure rules for a noun phrase as follows.

NP → Pronoun

NP → Det Noun

NP → Noun

NP → Adj Noun

NP → Det Adj Noun

We can combine all these rule in a single phrase structure rule as follows:

NP → (Det) (Adj) Noun|Pronoun

The constituents in parenthesis are optional. This rule states that a noun phrases consists of a noun, possibly preceded by determiner and adjective (in that order). This rule does not cover all possible NPs. A noun phrase may include post modifiers and more than one at adjective. For example, it may include a prepositional phrase (PP). More than one adjective is handled by allowing an adjective phrase (AP) for the adjective in the rule. After incorporating PP and AP in the phrase structure rule, we get the following

NP → (Det) (AP) Noun (PP)

The following are a few examples of noun phrases

They .... (1)

The foggy morning .... (2)

Chilled water .... (3)

A beautiful lake in Kashmir .... (4)

Cold banana shake .... (5)

Let's see how the above phrases can be generated using phrase structure rules. The (1) phrase consists only of a pronoun. The (2) phrase consists of a determiner, an adjective (foggy) that stands for an entire adjective phrase, and a noun. The (3) phrase comprises and adjective phrase and noun, (4) phrase consists of a determiner (the), an adjective phrase (beautiful), a noun (lake) and a prepositional phrase (in Kashmir) and (5) phrase consists of an adjective followed by a sequence of nouns. A noun sentence is termed as nominal. None of the phrase structure rules discussed so far are able to handle nominals. So, we modify our rules to cover this situation.

NP → (Det) (AP) Nom (PP)

Nom → Noun | Noun Nom

A noun phrase can act as a subject, an object or a predicate. The following sentences demonstrate each of these uses.

The foggy damped weather disturbed the match. .... (1)

I would like a nice cold banana shake. .... (2)

Kula botanical garden is a beautiful location. .... (3)

In (1), the noun phrase acts as subject. In (2), its acts as an object, and in (3), it is a predicate.

Verb Phrase

Analogous to the noun phrase is the verb phrase, which is headed by of verb. There is a fairly wide range of reasons that can modify of verb. This makes verb phrase a bit more Complex. The verb phrase organises various elements of the sentence that depend syntactically on the verb find the following are some examples of phrases:

Khushbu slept. .... (1)

The boy kicked the ball. .... (2)

Khushbu slept in the garden. .... (3)

The boy gave the girl a book. .... (4)

The Boy gave the girl a book with blue cover. .... (5)

As you can see from these examples a verb phrase can have a verb {VP → Verb in (1)}, a verb followed by an NP {VP → Verb NP in (2)}, a verb followed by a PP {VP → Verb PP in (3)}, a verb followed by two NPs {VP → Verb NP NP in (4)}, or a verb followed by two NPs and a PP {VP → Verb NP NP PP in (5)}. In general, the number of NPs in BP is limited to two, whereas it is possible to add more than two PPs.

VP → Verb (NP) (NP) (PP)*

Things are further complicated by the fact that objects may also be entire clauses as in the sentence, *I know that Taj is one of the seven wonders*. Hence, we must also allow for an alternative phrase statement rule, in with NP is replaced by S.

VP → Verb S

Prepositional Phrase

Prepositional phrases are headed by a preposition. They consist of a preposition, possibly followed by some other constituent, usually a noun phrase.

We played volleyball on the beach.

We can have a preposition phrase that consists of just a preposition.

John went outside.

The phrase structure rule that captures the above eventualities is as follows.

PP → Prep (NP)

Adjective Phrase

The head of an adjective phrase (AP) is an adjective. AP consists of an adjective, which may be preceded by an adverb and followed by a PP. Here are some examples.

Ashish is love.

The train is very late.

My sister is fond of animals.

The phrase structure rule for adjective phrase is

AP → (adv) Adj (PP)

Adverb Phrase

An adverb phrase consists of an adverb, possibly preceded by a degree adverb. Here is an example.

Time passes very quickly.

AdvP → (Intens) Adv

Sentences

Having discussed phrase structures, sentences. A sentence can have bearing structure commonly known structures are declarative structure, imperative structure, yes-no question structure and WH question structure.

Sentences with a declarative structure have a subject followed by a predicate. The predicate of a declarative sentence is a noun phrase and predicate is a verb phrase, e.g., *I like horse riding.* The phrase structure rule for declarative sentences are sentences is

S → NP VP

Sentences with an imperative structure usually begin with the verb phrase and lack subject. The subject of these types of sentences is implicit and is understood to be "you". These types of sentences are used for commands and suggestions, and hence are called imperative. The grammar rule for this kind of sentence structure is

S → VP

Examples of this kind of sentences are as follows:

Look at the door.

Give me the book.

Stop talking.

Show me the latest design.

Sentences with the yes-no question structure ask questions which can be answered using yes or no. These sentences begin with an auxiliary verb, followed by a subject NP, followed by a VP. Here are some examples

Do you have a Red pen?

Is awake and cotter?

Is the game over?

Can you show me your album?

We expand our grammar by adding another rule for the expansion of as, as follows:

S → Aux NP VP

Sentences with WH question structure are more complex. These sentences begin with WH words (who, which, where, what why and how). A WH question may have a WH phrase as a subject or may include another subject. Consider the following WH questions:

Which team won the match?

This sentence is similar to a declarative sentence except it contains a WH word. How simple rule to handle this type of sentence structure is

S → Wh-NP VP

Another type of WH questions structure is one that involves more than one NP. In this type of questions, the auxiliary verb comes before the subject NP, just as in yes-no questions structure.

Which cameras can you show me in your shop?

The rule for this type of WH questions is

- S → Wh-NP Aux NP VP

3. Relations among lexemes & their senses – Homonymy, Polysemy, Synonymy, Hyponymy

One way to approach lexical semantics is to study the relationship among lexemes (an abstract representation of a "word", the lexical entry in a dictionary). Semantics of a lexeme can be understood by analysing the relationship of lexemes with other lexemes. Lexical semantics information is useful for wide variety of NLP applications. This section discusses a variety of relationship that holds among lexemes and their senses.

Homonym

The first relationship that we discuss is homonymy which is perhaps the simplest relationship that exists among lexemes. Homonyms are words that have the same form but have different, unrelated meanings. A classic example of homonymy is Bank (river bank or financial institution). A related idea is that of homophones that refers to words that are pronounced in the same way but different meaning or spelling of both (e.g., be and bee, bear and bare).

Polysemy

Many words have more than one meaning of sense. Unlike homonyms, polysemes are words with related meanings. This linguistic phenomenon is called polysemy or lexical ambiguity. Words that have several senses are ambiguous and called polysemous. For example, the word "chair" can refer to a piece of furniture, a person, the act of presiding over a discussion etc. The word "employ" is a polysemy as its two meaning- to hire (employ a person) and to accept (employ an idea) are related. In a particular use, only one of these meanings is correct.

Hyponymy

The hypernym is a word with the more general sense. The word automobile is a hypernym for a car and a truck. The hyponym is a word with the most specific meaning. In the relationship between car and automobile, car is a hyponym of automobile. Antonym is a semantic relationship that holds between words that express opposite meanings. The word Good is an antonym of Bad, and White is an antonym of Black.

Synonym

The word synonym defines the relationship between different words that have a similar meaning. A simple way to decide whether two words are synonymous is to check for substitutability. Two words are synonyms in a context if they can be substituted for each other without changing the meaning of the sentence.

These relationships are useful in organising words in lexical databases one widely known lexical database is WordNet discussed in next topic.

4. WordNet

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing.

WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

Structure

The main relation among words in WordNet is synonymy, as between the words shut and close or car and automobile. Synonyms--words that denote the same concept and are interchangeable in many contexts--are grouped into unordered sets (synsets). Each of WordNet's 117 000 synsets is linked to other synsets by means of a small number of "conceptual relations." Additionally, a synset contains a brief definition ("gloss") and, in most cases, one or more short sentences illustrating the use of the synset members. Word forms with several distinct meanings are represented in as many distinct synsets. Thus, each form-meaning pair in WordNet is unique.

Relations

The most frequently encoded relation among synsets is the super-subordinate relation (also called hyperonymy, hyponymy or ISA relation). It links more general synsets like {furniture, piece_of_furniture} to increasingly specific ones like {bed} and {bunkbed}. Thus, WordNet states that the category furniture includes bed, which in turn includes bunkbed; conversely, concepts like bed and bunkbed make up the category furniture. All noun hierarchies ultimately go up the root node {entity}. Hyponymy relation is transitive: if an armchair is a kind of chair, and if a chair is a kind of furniture, then an armchair is a kind of furniture. WordNet distinguishes among Types (common nouns) and Instances (specific persons, countries and geographic entities). Thus, armchair is a type of chair; Barack Obama is an instance of a president. Instances are always leaf (terminal) nodes in their hierarchies.

Meronymy, the part-whole relation holds between synsets like {chair} and {back, backrest}, {seat} and {leg}. Parts are inherited from their super ordinates: if a chair has legs, then an armchair has legs as well. Parts are not inherited "upward" as they may be characteristic only of specific kinds of things rather than the class as a whole: chairs and kinds of chairs have legs, but not all kinds of furniture have legs.

Verb synsets are arranged into hierarchies as well; verbs towards the bottom of the trees (troponyms) express increasingly specific manners characterizing an event, as in {communicate}-{talk}-{whisper}. The specific manner expressed depends on the semantic field; volume (as in the example above) is just one dimension along which verbs can be elaborated. Others are speed (move-jog-run) or intensity of emotion (like-love-idolize). Verbs describing events that necessarily and unidirectionally entail one another are linked: {buy}-{pay}, {succeed}-{try}, {show}-{see}, etc.

Adjectives are organized in terms of antonymy. Pairs of "direct" antonyms like wet-dry and young-old reflect the strong semantic contract of their members. Each of these polar adjectives in turn is linked to a number of "semantically similar" ones: dry is linked to parched, arid, dessicated and bone-dry and wet to soggy, waterlogged, etc. Semantically similar adjectives are "indirect antonyms" of the contral member of the opposite pole. Relational adjectives ("pertainyms") point to the nouns they are derived from (criminal-crime).

There are only few adverbs in WordNet (hardly, mostly, really, etc.) as the majority of English adverbs are straightforwardly derived from adjectives via morphological affixation (surprisingly, strangely, etc.)

Applications of WordNet

WordNet has found numerous applications in problems related with IR and NLP. Some of these are given below:

Concept Identification in Natural Language: WordNet can be used to identify concepts pertaining to a term, to suit them to the full semantic richness and complexity of a given information need.

Word Sense Disambiguation: WordNet combines features of a number of the other resources commonly used in disambiguation work. It offers sense definitions of words, identifies synsets of synonyms, defines a number of semantics relations and is freely available. This makes it the (currently) best known and most utilized resource for word sense disambiguation.

Automatic Query Expansion: WordNet semantic relations can be used to expand queries so that the search for a document is not confined to the pattern-matching of query terms, but also covers synonyms.

Document Summarization: WordNet has found useful application in text summarization. Few approaches utilize information from WordNet to compute lexical chains.

5. Robust Word Sense Disambiguation (WSD) - Dictionary based approach

Most of us find it difficult to understand by computers are not able to understand language the way people are. This is because we do not realise how weight and ambiguous natural languages are. Ambiguity that is having more than one meaning, can be result of syntax or semantics. Vagueness is not the same as ambiguity. In vagueness, words or phrases have only one meaning but they lack clarity, which makes it difficult to arrive at a precise meaning.

In many cases, a single word in a language corresponds to more than one thought, for example the noun Bank (financial institution of bank of a river) and the verb run (to move fast or to direct and manage). But this does not create a problem for us. We hardly give any thought to understanding that what constitutes the correct meaning of a word or phrase, but we generally arrive at the correct one. The process is almost effortless. People are good at resolving ambiguity by considering the context of the written text or coconut princess. Except for jokes and puns, where it is intended, ambiguity is not perceived as such. However, ambiguities existing at different levels are one of the major challenges in computational linguistics.

Let us try to understand why ambiguity is difficult. This is because it increases the range of possible interpretations of natural language. Suppose each word in an 8-word sentence is ambiguous and has three possible interpretations. The total number of interpretations of the whole sentence is $3^8 = 3561$. Further, synthetic and pragmatic ambiguities make the actual number of interpretations even larger. Resolving all these interpretations in a reasonable amount of time is difficult. There are words with a much larger number of sensors that then considered in this example. This gives a clear picture of the difficulty involved in the automatic interpretation of natural languages.

Ambiguity is the property of linguistic expressions. Ambiguity means capable of being understood in more than one way of having more than one meaning. It refers to a situation where an expression can have more than one partition. Ambiguity can occur at four different levels:

Lexical

Lexical ambiguity is ambiguity of a single word. A word can be ambiguous with respect to its internal structure or to its syntactic class. For example in the sentence, *look at the screen*, look is verb where as in *she gave me a stern look*, it is a noun. However this type of ambiguity is viewed as part of speech tagging in NLP and is considered to have been solved with reasonable accuracy.

Syntactic

There are different ways in which sequence of words can be grammatically structured. Each structuring leads to a different interpretation. For example, *the man saw the girl with the telescope*. It is unclear & ambiguous whether the Man saw a girl carrying a telescope, or he saw her through the telescope. It is the syntax, not the meaning of the word which is unclear. The meaning is dependent on whether the preposition "with" is attached to the girl or the man.

Semantic

Semantic ambiguity occurs when the meaning of the word themselves can be misinterpreted. For example, the meaning of words in the phrase can be combined in different ways, leading to different interpretations.

Iraqi head seeks arms

The homograph "head" can be interpreted as a noun meaning either chief or anatomical head of a body. Likewise the homograph arms can be interpreted as a plural noun meaning either weapons or body parts.

Pragmatic

Pragmatic ambiguity refers to a situation where the context of a phrase gives it multiple interpretations.

For example, *Give it to the kids.* Here "it" may refer to many things depending on the context. Consider a larger context.

Cake is on the table. I have prepared some snacks. Give it to kids.

It is not clear whether it refers to cake or snacks for both. Perhaps a larger context may help us.

Cake is on the table. I have prepared some stacks. Give it to kids. Kids enjoyed cake and snacks

Now it is clear that it refers to both snacks and the cake. Resolving these type of ambiguous required Discourse Processing

We understand that words have different meanings based on the context of its usage in the sentence. If we talk about human languages, then they are ambiguous too because many words can be interpreted in multiple ways depending upon the context of their occurrence.

Word sense disambiguation in natural language processing (NLP), may be defined as the ability to determine which meaning of word is activated by the use of words in a particular context. Lexical ambiguity, syntactic or semantic, is one of the very first problems that any NLP system faces. Part-of-speech (POS) taggers with high level of accuracy can solve Word's syntactic ambiguity. On the other hand, the problem of resolving semantic ambiguity is called WSD (word sense disambiguation). Resolving semantic ambiguity is harder than resolving syntactic ambiguity.

For example, consider the two examples of the distinct sense that exist for the word *"bank"* –

*The **bank** will not be accepting cash on Saturdays.*

*The river overflowed the **bank**.*

The occurrence of the word **bank** clearly denotes the distinct meaning. In the first sentence, it means commercial (finance) banks, while in the second sentence; it refers to the river bank. Hence, if it would be disambiguated by WSD then the correct meaning to the above sentences can be assigned as follows –

*The **bank/financial institution** will not be accepting cash on Saturdays.*

*The river overflowed the **bank/riverfront.***

The evaluation of WSD requires the following two inputs:

A Dictionary: The very first input for evaluation of WSD is dictionary, which is used to specify the senses to be disambiguated.

Test Corpus: Another input required by WSD is the high-annotated test corpus that has the target or correct-senses. The test corpora can be of two types:

- **Lexical sample** - This kind of corpora is used in the system, where it is required to disambiguate a small sample of words.
- **All-words** - This kind of corpora is used in the system, where it is expected to disambiguate all the words in a piece of running text.

Dictionary-based or Knowledge-based Approach (WSD)

As the name suggests, for disambiguation, these methods primarily rely on dictionaries, treasures and lexical knowledge base. They do not use corpora evidence for disambiguation. The Lesk method is the seminal dictionary-based method introduced by Michael Lesk in 1986. The Lesk definition, on which the Lesk algorithm is based, is **"measure overlap between sense definitions for all words in context"**. However, in 2000, Kilgarriff and Rosensweig gave the simplified Lesk definition as **"measure overlap between sense definitions of word and current context"**, which further means identify the correct sense for one word at a time. Here the current context is the set of words in the surrounding sentence or paragraph.

The Lesk algorithm is based on the assumption that words in a given "neighbourhood" (section of text) will tend to share a common topic. A simplified version of the Lesk algorithm is to compare the dictionary definition of an ambiguous word with the terms contained in its neighbourhood.

Versions have been adapted to use WordNet. An implementation might look like this:

1. for every sense of the word being disambiguated one should count the amount of words that are in both neighbourhood of that word and in the dictionary definition of that sense
2. the sense that is to be chosen is the sense which has the biggest number of this count

A frequently used example illustrating this algorithm is for the context "pine cone". The following dictionary definitions are used:

Pine:

- *Kinds of evergreen tree with needle-shaped leaves*
- *Waste away through sorrow or illness*

Cone:

- *Solid body which narrows to a point*
- *Something of this shape whether solid or hollow*
- *Fruit of certain evergreen trees*

As can be seen,

Pine#1 Ç Cone#1 = 0
Pine#2 Ç Cone#1 = 0
Pine#1 Ç Cone#2 = 1
Pine#2 Ç Cone#2 = 0
Pine#1 Ç Cone#3 = 2
Pine#2 Ç Cone#3 = 0

the best intersection is Pine #1 Ç Cone #3 = 2.

Expected Questions

1. What is semantic analysis? why semantic analysis difficult? Explain various approaches to semantic analysis.
2. Explain with suitable examples following relationships between word meanings Homonymy, Polysemy, Synonymy, Antonymy, Hypernomy, Hyponomy, Meronomy
3. What is semantic analysis? Discuss different semantic relationships between the words.
4. What is WordNet? How is "sense" defined in WordNet? Explain with example.
5. What do you mean by word sense disambiguation (WSD)? discuss dictionary based approach for WSD.
6. What do you mean by word sense disambiguation (WSD)? Discuss knowledge based WSD
7. What do you mean by word sense disambiguation (WSD)? discuss machine learning based(Navie based) approach for WSD.
8. Explain how a supervised learning algorithm can be applied for word sense disambiguation.

Pragmatics

5

OBJECTIVES

After reading this chapter, the student will be able to understand:

- Pragmatic Analysis
- Discourse – reference resolution.
- Reference phenomenon.
- Syntactic & semantic constraints on coreference

1. Pragmatic analysis

Pragmatics is a subfield of linguistics that studies the ways in which context contributes to meaning. Pragmatics encompasses speech act theory, conversational implicature, talk in interaction and other approaches to language behaviour in philosophy, sociology, linguistics and anthropology. Unlike semantics, which examines meaning that is conventional or "coded" in a given language, pragmatics studies how the transmission of meaning depends not only on structural and linguistic knowledge (e.g., grammar, lexicon, etc.) of the speaker and listener, but also on the context of the utterance, any pre-existing knowledge about those involved, the inferred intent of the speaker, and other factors. In this respect, pragmatics explains how language users are able to overcome apparent ambiguity, since meaning relies on the manner, place, time, etc. of an utterance.

Pragmatics deals with using and understanding sentences in different situations and how the interpretation of the sentence is affected. The ability to understand another speaker's intended meaning is called pragmatic competence.

Pragmatics is different than semantics, which concerns the relations between signs and the objects they signify. Semantics refers to the specific meaning of language; pragmatics, by contrast, involves all of the other social cues that accompany language

Pragmatics focuses not on what people say but how they say it and how others interpret their utterances in social contexts. Utterances are literally the units of sound you make when you talk, but the signs that accompany those utterances are what give the sounds their true meaning.Ref

Example : You invited your friend over for dinner. Your child sees your friend reach for some cookies and says, 'Better not take those, or you'll get even bigger.' You can't believe your child could be so rude."

In a literal sense, the daughter is simply saying that eating cookies can make you gain weight. But due to the social context, the mother interprets that same sentence to mean that her daughter is calling her friend fat. The first sentence in this explanation refers to the semantics—the literal meaning of the sentence. The second and third refer to the pragmatics, the actual meaning of the words as interpreted by a listener based on social context.

Some Example :

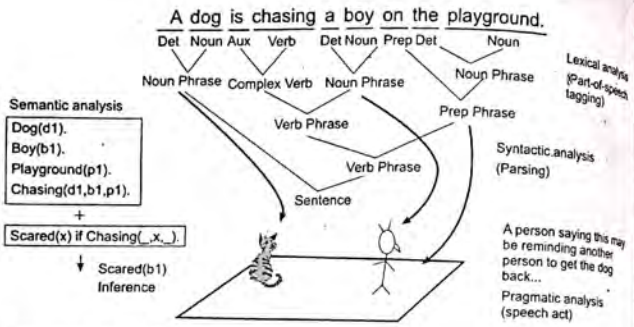
"The question 'can I cut you?' Means very different things if I'm standing next to you in line or if I am holding a knife".

"Is that water?" The action to be performed is different in a chemistry lab and on a dining table

If you go to your editor and ask her to suggest a better sentence structure for a line, her immediate question to you will be, "What's the context?" Most of the time, due to flexibility of the natural language, complexities arise in interpreting the meaning of an isolated statement. Ref.

Pragmatic analysis uses the context of utterance—when, why, by who, where, to whom something was said. It deals with intentions like criticize, inform, promise, request, and so on. For example, if I say "You are late," is it information or criticism? In discourse integration, the aim is to analyze the statement in relation to the preceding or succeeding statements or even the overall paragraph in order to understand its meaning. Take this one: Chloe wanted it. ("It" depends on Chloe). Pragmatic analysis interprets the meaning in terms of context of use unlike semantics. Ref.

An Example of NLP



Pragmatics can be defined as :

"It is the study of speaker meaning"

It is concerned with the study of meaning as communicated by speaker and interpreted by the listener.

"It is the study of contextual meaning"

It involves interpretation of what people mean in a particular context and how the context influences what is said.

"It is the study of how more gets communicated than is said"

This type of study explores how great deal of what is unsaid is recognized as part of what is communicated.

"It is the study of the expression of relative distance"

On the assumption of how close and distant the listener is speakers determine how much needs to be said.

Five aspects of pragmatics

Deixis:

Deixis concerns the ways in which languages encode or grammaticalized features of the context of utterance or speech event, and thus also concerns ways in which the interpretation of utterances depends on the analysis of that context of utterance. Thus the pronoun this does not name or refer to any particular entity on all occasions of use; rather it is a variable or place-holder for some particular entity given by the context (e.g. by a gesture). In short Deixis is methods of directly encoding context into language

Consider, for example, finding the following notice on someone's office door:

"I'll be back in an hour"

Because we don't know when it was written, we cannot know when the writer will return.

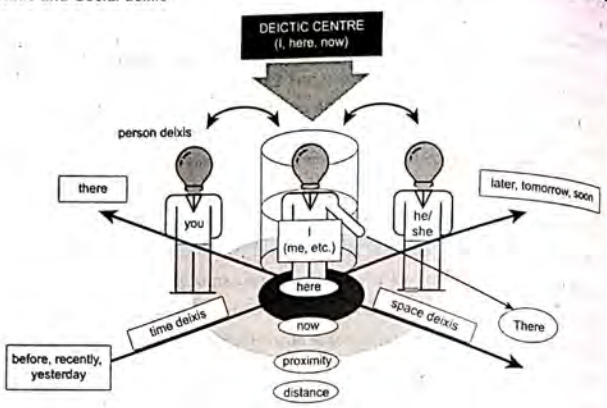
Or, suppose we find a bottle in the sea, and inside it a message which reads

"Meet me here a week from now with a stick about this big"

We do not know who to meet where or when to meet him or her, or how big a stick to bring

Deixis refers to words and phrases, such as "me" or "here", that cannot be fully understood without additional contextual information—in this case, the identity of the speaker ("me") and the speaker's location ("here"). Words are deictic if their semantic meaning is fixed but their denotational meaning varies depending on time and/or place. Words or phrases that require contextual information to convey any meaning—for example, English pronouns—are deictic. Deictic pronouns are pronouns that reference speaker- and hearer-known context.

Types of deixis are : Person dexis, *Time deixis, Place deixis or Spatial deictic.Discourse deixis and Social deixis*



Implicature

Implicature: It means more being communicated than is said.

Conversational implicature: a meaning or message that is implicated in a conversation

When people oversay (or say more of) or undersay (say less of) something, they produce certain extra meaning or meanings beyond the literal meanings of words and sentences. This extra meaning is conversationally dependent, hence conversation implicature.

An implicature may also be seen as an indirect way of expressing oneself.

Example 1:

A: Where is the fish?

B: The cat looks very happy.

Example 2:

A: Did you invite John and Mary?

B: I invited John

Example 3:

A (to passer by): I am out of gas.

B: There is a gas station around the corner.

Here, B does not say, but conversationally implicates, that the gas station is open, because otherwise his utterance would not be relevant in the context

Conversational implicatures are classically seen as contrasting with entailments: They are not necessary consequences of what is said, but are defeasible (cancellable). So, B could continue without contradiction:

B: But unfortunately, it's closed today.

An example of a conventional implicature is "Donovan is poor but happy", where the word "but" implies a sense of contrast between being poor and being happy

An important contribution made by the notion of implicature is that it provides some explicit account of how it is possible to mean (in some general sense) more than what is actually 'said' (i.e. more than what is literally expressed by the conventional sense of the linguistic expressions uttered).

Consider, for example:

A: Can you tell me the time?

B: Well, the milkman has come

Grice's theory says that a speaker should: be cooperative, be truthful (quality), be informative (quantity), be relevant, and be perspicuous (manner: avoid obscurity and ambiguity, be brief and orderly). This theory is insufficient, but useful. Implicature also includes notions of quantification (perhaps, some, many, etc.) and metaphors.

Presupposition

Presupposition is something the speaker assumes to be the case prior to making an utterance. It roughly describes that which is immediately inferrable but not the new information in an utterance.

Examples of presuppositions include:

Jane no longer writes fiction. Presupposition: Jane once wrote fiction.

Have you stopped eating meat? Presupposition: you had once eaten meat.

Have you talked to Hans? Presupposition: Hans exists.

Example:

"Mary's brother bought three horses"

Presuppositions:

1: Mary exists

2: She has a brother

3: She has only one brother

4: He has a lot of money

Eg., "Sue cried before she finished her thesis" presupposes that Sue finished her thesis, but this is not the new information (which is that she cried). There are both semantic and pragmatic presuppositions; the latter involve shared knowledge between speaker and hearer. A presupposition must be mutually known or assumed by the speaker and addressee for the utterance to be considered appropriate in context. It will generally remain a necessary assumption whether the utterance is placed in the form of an assertion, denial, or question, and can be associated with a specific lexical item or grammatical feature (presupposition trigger) in the utterance. Crucially, negation of an expression does not change its presuppositions: I want to do it again and I don't want to do it again both presuppose that the subject has done it already one or more times

Speech Acts

In linguistics, a speech act is an utterance defined in terms of a speaker's intention and the effect it has on a listener. Essentially, it is the action that the speaker hopes to provoke in his or her audience. Speech acts might be requests, warnings, promises, apologies, greetings, or any number of declarations. As you might imagine, speech acts are an important part of communication. Ref

A speech act is a statement that does something rather than just one that says something (eg., "I declare war on Zanzibar." The basic question of speech acts seems to be that of identifying them and understanding how they differ from normal statements. A speech act in linguistics and the philosophy of language is something expressed by an individual that not only presents information, but perform an action as well. For example, the phrase "I would like the mashed potatoes, could you please pass them to me?" is considered a speech act as it expresses the speaker's desire to acquire the mashed potatoes, as well as presenting a request that someone pass the potatoes to them.

Another example The boss utters "You are fired" Performs an act of ending the employment Speech acts can be Command, apology, complaint, compliment, invitation, promise, request etc. "The tea is really cold!" Complaint during winter Compliment during summer.

"You're fired!" expresses both the employment status of the individual in question, as well as the action by which said person's employment is ended.

"I hereby appoint you as chairman" expresses both the status of the individual as chairman, and is the action which promotes the individual to this position.

"We ask that you extinguish your cigarettes at this time, and bring your tray tables and setbacks to an upright position." This statement describes the requirements of the current location, such as an airplane, while also issuing the command to stop smoking and to sit up straight.

"Would it be too much trouble for me to ask you to hand me that wrench?" functions to simultaneously ask two questions. The first is to ask the listener if they are capable of passing the wrench, while the second is an actual request

"Well, would you listen to that?" acts as a question, requesting that a listener heed what is being said by the speaker, but also as an exclamation of disbelief or shock

Conversational Structure

analysis of the sequential (and anti-sequential) nature of conversations, interruptions, etc.

Application that demands pragmatic understanding :

- IR and IE also seem impervious to the pragmatic hammer. QA might benefit, "Did Sue finish her thesis?"
- Even summarization seems fairly robust to a lack of pragmatic understanding, although the "new/old" issue is important here. Perhaps what is old in the real discourse is not new for the reader of the summary.
- Sentiment Analysis

2. Discourse - reference resolution

Language does not normally consist of isolated, unrelated sentences, but instead of collocated, related groups of sentences. Such a group of sentences as a discourse. Monologues are characterized by a speaker (a term which will be used to include writers as it is here), and a hearer (which, analogously, includes readers). The communication flows in only one direction in a monologue, that is, from the speaker to the hearer. A conversation with a friend about it, which would consist of a much freer interchange. Such a discourse is called a dialogue. In this case, each participant periodically takes turns being a speaker and hearer. Unlike a typical monologue, dialogues generally consist of many different types of communicative acts: asking questions, giving answers, making corrections, and so forth. Finally, computer systems exist and continue to be developed that allow for human-computer interaction, or HCI. HCI has properties that distinguish it from normal human-human dialogue, in part due to the present-day limitations on the ability of computer systems to participate in free, unconstrained conversation. A system capable of HCI will often employ a strategy to constrain the conversation in ways that allow it to understand the user's utterances within a limited context of interpretation. While many discourse processing problems are common to these three forms of discourse, they differ in enough respects that different techniques have often been used to process them.

Language is rife with phenomena that operate at the discourse level. John went to Bill's car dealership to check out an Acura Integra. He looked at it for about an hour. What do pronouns such as he and it denote? Can we build a computational model for the resolution of referring expressions?

Algorithms for resolving discourse-level phenomena are essential for a wide range of language applications. For instance, interactions with query interfaces and dialogue interpretation systems like ATIS frequently contain pronouns and similar types of expressions.

I'd like to get from Boston to San Francisco, on either December 5th or December 6th. It's okay if it stops in another city along the way.

It denotes the flight that the user wants to book in order to perform the appropriate action.

IE systems must frequently extract information from utterances that contain pronouns

First Union Corp is continuing to wrestle with severe problems unleashed by a botched merger and a troubled business strategy. According to industry insiders at Paine Webber, their president, John R. Georgius, is planning to retire by the end of the year.

Text summarization systems employ a procedure for selecting the important sentences from a source document and using them to form a summary.

Reference Resolution

In this section we study the problem of reference, the process by which REFERENCE speakers use expressions like John and he in passage (John went to Bill's car dealership to check out an Acura Integra. He looked at it for about an hour) to denote a person named John. Our discussion requires that we first define some terminology. A natural language expression used to perform reference is called a referring expression, and the entity that is referred to is called the referent. Thus, John and he in passage (John went to Bill's car dealership to check out an Acura Integra. He looked at it for about an hour) are referring expressions, and John is their referent. (To distinguish between referring expressions and their referents, we italicize the former.) As a convenient shorthand, we will sometimes speak of a referring expression referring to a referent, e.g., we might say that he refers to John. However, the reader should keep in mind that what we really mean is that the speaker is performing the act of referring to John by uttering he. Two referring expressions that are used to refer to the same entity are said to corefer, thus John and he corefer in passage. There is also a term for a referring expression that licenses the use of another, in the way that the mention of John allows John to be subsequently referred to using he. We call John the antecedent of he. Reference to an entity that has been previously introduced into the discourse is called anaphora, and the referring expression used is said to be anaphoric. In passage .the pronouns he and it are therefore anaphoric. Natural languages provide speakers with a variety of ways to refer to entities. Say that your friend has an Acura Integra automobile and you want to refer to it. Depending on the operative discourse context, you might say it, this, that, this car, that car, the car, the Acura, the Integra, or my friend's car, among many other possibilities. However, you are not free to choose between any of these alternatives in any context. For instance, you cannot simply say it or the Acura if the hearer has no prior knowledge of your friend's car, it has not been mentioned before, and it is not in the immediate surroundings of the discourse participants (i.e., the situational context of the discourse).

The reason for this is that each type of referring expression encodes different signals about the place that the speaker believes the referent occupies within the hearer's set of beliefs. A subset of these beliefs that has a special status form the hearer's mental model of the ongoing discourse, which we call a discourse model (Webber, 1978). The discourse model contains representations of the entities that have been referred to in the discourse and the relationships in which they participate. Thus, there are two components required by a system to successfully produce and interpret referring expressions: a method for constructing a discourse model that evolves with the dynamically-changing discourse it represents, and a method for mapping between the signals that various referring expressions encode and the hearer's set of beliefs, the latter of which includes the discourse model. We will speak in terms of two fundamental operations to the discourse model. When a referent is first mentioned in a discourse, we say that a representation for it is evoked into the model. Upon subsequent mention, this representation is accessed from the model. The operations and relationships are illustrated in Figure below

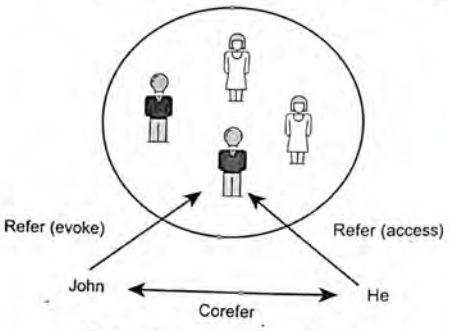


Figure 1: reference operation and relationships

We will restrict our discussion to reference to entities, although discourses include references to many other types of referents. Consider the possibilities in example:

According to John, Bob bought Sue an Integra, and Sue bought Fred a Legend.

a. But that turned out to be a lie.
b. But that was false.
c. That struck me as a funny way to describe the situation.
d. caused Sue to become rather poor.
e. That caused them both to become rather poor. The referent of that is a speech act in (a), a proposition in (b), a manner of description in (c), an event in (d), and a combination of several events in (e). The field awaits the development of robust methods for interpreting these types of reference.

3. Reference Phenomenon

The set of referential phenomena that natural languages provide is quite rich indeed. Five types of referring expressions: indefinite noun phrases, definite noun phrases, pronouns, demonstratives, and one-anaphora. Three types of referents that complicate the reference resolution problem: inferrables, discontinuous sets, and generics.

Indefinite Noun Phrases: Indefinite reference introduces entities that are new to the hearer into the discourse context. The most common form of indefinite reference is marked with the determiner a (or an), as in A, but it can also be marked by a quantifier such as B or even the determiner this C.

A. I saw an Acura Integra today.
B. Some Acura Integras were being unloaded at the local dealership today.
C. I saw this awesome Acura Integra today.

Such noun phrases evoke a representation for a new entity that satisfies the given description into the discourse model. The indefinite determiner a does not indicate whether the entity is identifiable to the speaker, which in some cases leads to a specific/non-specific ambiguity. Example A only has the specific reading, since the speaker has a particular Integra in mind, particularly the one she saw. In sentence D, on the other hand, both readings are possible.

D. I am going to the dealership to buy an Acura Integra today.

That is, the speaker may already have the Integra picked out (specific), or may just be planning to pick one out that is to her liking (nonspecific). The readings may be disambiguated by a subsequent referring expression in some contexts; if this expression is definite then the reading is specific (I hope they still have it), and if it is indefinite then the reading is nonspecific (I hope they have a car I like). This rule has exceptions, however; for instance definite expressions in certain modal contexts (I will park it in my garage) are compatible with the nonspecific reading.

Definite Noun Phrases Definite reference is used to refer to an entity that is identifiable to the listener, either because it has already been mentioned in the discourse context (and thus is represented in the discourse model), it is contained in the hearer's set of beliefs about the world, or the uniqueness of the object is implied by the description itself. The case in which the referent is identifiable from discourse context is shown in , S1: I saw an Acura Integra today. The Integra was white and needed to be washed.

Pronouns Another form of definite reference is pronominalization, illustrated in example I saw an Acura Integra today. It was white and needed to be washed. The constraints on using pronominal reference are stronger than for full definite noun phrases, requiring that the referent have a high degree of activation or salience in the discourse model. Pronouns usually (but not always) refer to entities that were introduced no further back than one or two sentences back in the ongoing discourse, whereas definite noun phrases can often refer further back.

A . John went to Bob's party, and parked next to a beautiful Acura Integra. b. He went inside and talked to Bob for more than an hour. c. Bob told him that he recently got engaged. d. ?? He also said that he bought it yesterday. d.' He also said that he bought the Acura yesterday. By the time the last sentence is reached, the Integra no longer has the degree of salience required to allow for pronominal reference to it. Pronouns can also participate in cataphora, in which they are mentioned before their referents are, as in example (S2). S2 Before he bought it, John checked over the Integra very carefully. Here, the pronouns he and it both occur before their referents are introduced. Pronouns also appear in quantified contexts in which they are considered to be bound, as in example (S3). BOUND (S3) Every woman bought her Acura at the local dealership. Under the relevant reading, her does not refer to some woman in context, but instead behaves like a variable bound to the quantified expression every woman. We will not be concerned with the bound interpretation of pronouns in this chapter.

Demonstratives: Demonstrative pronouns, like this and that, behave somewhat differently than simple definite pronouns like it. They can appear either alone or as determiners, for instance, this Acura, that Acura. The choice between two demonstratives is generally associated with some notion of spatial proximity: this indicating closeness and that signalling distance.

Spatial distance might be measured with respect to the discourse participants' situational context, as in (S4). (S4) [John shows Bob an Acura Integra and a Mazda Miata.] Bob (pointing): I like this better than that. Alternatively, distance can be metaphorically

interpreted in terms of conceptual relations in the discourse model. For instance, consider example (S5). (S5) I bought an Integra yesterday. It's similar to the one I bought five years ago. That one was really nice, but I like this one even better. Here, that one refers to the Acura bought five years ago (greater temporal distance), whereas this one refers to the one bought yesterday (closer temporal distance).

One Anaphora: One-anaphora, exemplified in (S6), blends properties of definite and indefinite reference. (S6) I saw no less than 6 Acura Integras today. Now I want one. This use of one can be roughly paraphrased by one of them, in which them refers to a plural referent (or generic one, as in the case of (S6), see below), and one selects a member from this set (Webber, 1983). Thus, one may evoke a new entity into the discourse model, but it is necessarily dependent on an existing referent for the description of this new entity. This use of one should be distinguished from the formal, non-specific pronoun usage in (S6), and its meaning as the number one in (S7). (S6) One shouldn't pay more than twenty thousand dollars for an Acura. (S7) John has two Acuras, but I only have one.

Inferrables: Now that we have described several types of referring expressions, we now turn our attention to a few interesting types of referents that complicate the reference resolution problem. First, we consider cases in which a referring expression does not refer to an entity that has been explicitly evoked in the text, but instead one that is inferentially related to an evoked entity. Such referents are called inferrables (Haviland and Clark, 1974; Prince, 1981). Consider the expressions a door and the engine in sentence (S8). (S8) I almost bought an Acura Integra today, but a door had a dent and the engine seemed noisy. The indefinite noun phrase a door would normally introduce a new door into the discourse context, but in this case the hearer is to infer something more: that it is not just any door, but one of the doors of the Integra. Similarly, the use of the definite noun phrase the engine normally presumes that an engine has been previously evoked or is otherwise uniquely identifiable. Here, no engine has been explicitly mentioned, but the hearer infers that the referent is the engine of the previously mentioned Integra. Inferrables can also specify the results of processes described by utterances in a discourse. Consider the possible follow-ons (a-c) to sentence (S9) in the following recipe (from Webber and Baldwin (1992)): (S9) Mix the flour, butter, and water. a. Kneed the dough until smooth and shiny. b. Spread the paste over the blueberries. c. Stir the batter until all lumps are gone. Any of the expressions the dough (a solid), the batter (a liquid), and the paste (somewhere in between) can be used to refer to the

result of the actions described in the first sentence, but all imply different properties of this result.

Discontinuous Sets: In some cases, references using plural referring expressions like they and them (see page 10) refer to sets of entities that are evoked together, for instance, using another plural expression (their Acuras) or a conjoined noun phrase (John and Mary): (S10) John and Mary love their Acuras. They drive them all the time. However, plural references may also refer to sets of entities that have been evoked by discontinuous phrases in the text: (S11) John has an Acura, and Mary has a Mazda. They drive them all the time. Here, they refers to John and Mary, and likewise them refers to the Acura and the Mazda. Note also that the second sentence in this case will generally receive what is called a pairwise or respectively reading, in which John drives the Acura and Mary drives the Mazda, as opposed to the reading in which they both drive both cars.

Generics: Making the reference problem even more complicated is the existence of generic reference. Consider example (S12). (S12) I saw no less than 6 Acura Integras today. They are the coolest cars. Here, the most natural reading is not the one in which they refers to the particular 6 Integras mentioned in the first sentence, but instead to the class of Integras in general

4. Syntactic and Semantic Constraints on Coreference

Having described a variety of reference phenomena that are found in natural language, we can now consider how one might develop algorithms for identifying the referents of referential expressions. One step that needs to be taken in any successful reference resolution algorithm is to filter the set of possible referents on the basis of certain relatively hard-and-fast constraints. We describe some of these constraints here.

Number Agreement Referring expressions and their referents must agree in number; for English, this means distinguishing between singular and plural references. A categorization of pronouns with respect to number is shown in Figure A

Singular	Plural	Unspecified
she, her, he, him, his, it	we, us, they, them	you

Figure 2: Number agreement in the English pronominal system.

The following examples illustrate constraints on number agreement.

1. John has a new Acura. It is red.
2. John has three new Acuras. They are red.
3. John has a new Acura. They are red.
4. John has three new Acuras. It is red.

Person and Case Agreement

English distinguishes between three forms of person: first, second, and third. A categorization of pronouns with respect to person is shown in Figure B The following examples illustrate constraints on person agreement. S13 You and I have Acuras. We love them.

	First	Second	Third
Nominative	I, we	you	he, she, they
Accusative	me, us	you	him, her, them
Genitive	my, our	your	his, her, their

Figure 3: Person and case agreement in the English pronominal system

Coreference

In linguistics, coreference, sometimes written co-reference, occurs when two or more expressions in a text refer to the same person or thing; they have the same referent, e.g. Bill said he would come; the proper noun Bill and the pronoun he refer to the same person, namely to Bill. Coreference occurs when one or more expressions in a document refer back to the entity that came before it/them.

Coreference is the main concept underlying binding phenomena in the field of syntax. The theory of binding explores the syntactic relationship that exists between coreferential expressions in sentences and texts. When two expressions are coreferential, the one is usually a full form (the antecedent) and the other is an abbreviated form (a proform or anaphor). Linguists use indices to show coreference, as with the i index in the example Billi said hei would come. The two expressions with the same reference are coindexed, hence in this example Bill and he are coindexed, indicating that they should be interpreted as coreferential. Ref

Coreference distinctions

When exploring coreference, there are numerous distinctions that can be made, e.g. anaphora, cataphora, split antecedents, coreferring noun phrases, etc. When dealing with proforms (pronouns, pro-verbs, pro-adjectives, etc.), one distinguishes between anaphora and cataphora. When the proform follows the expression to which it refers, anaphora is present (the proform is an anaphor), and when it precedes the expression to which it refers, cataphora is present (the proform is a cataphor).

These notions are illustrated as follows:

Anaphora

a. **The music**$_i$ was so loud that **it**$_i$ couldn't be enjoyed. –The anaphor *it* follows the expression to which it refers (its antecedent).

b. **Our neighbors**$_i$ dislike the music. If **they**$_i$ are angry, the cops will show up soon. – The anaphor *they* follows the expression to which it refers (its antecedent).

Cataphora

a. If **they**$_i$ are angry about the music, **the neighbors**$_i$ will call the cops. – The cataphor *they* precedes the expression to which it refers (its postcedent).

b. Despite **her**$_i$ difficulty, **Wilma**$_i$ came to understand the point. – The cataphor *her* precedes the expression to which it refers (its postcedent)

Split antecedents

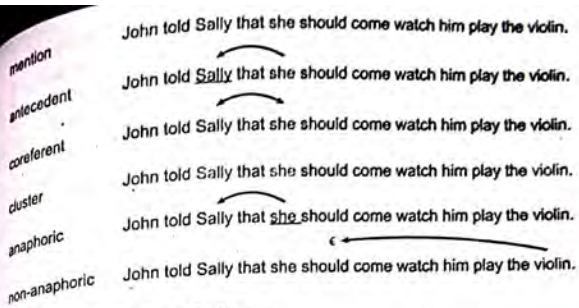
a. **Carol**$_i$ told **Bob**$_j$ to attend the party. **They**$_{i,j}$ arrived together. – The anaphor *they* has a split antecedent, referring to both *Carol* and *Bob*.

b. When **Carol**$_i$ helps **Bob**$_j$ and **Bob**$_j$ helps **Carol**$_i$, **they**$_{i,j}$ can accomplish any task. - The anaphor *they* has a split antecedent, referring to both *Carol* and *Bob*.

Coreferring noun phrases

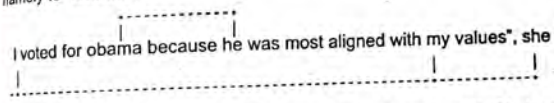
a. **The project leader**$_i$ is refusing to help. **The jerk**$_i$ thinks only of himself. – Coreferring noun phrases, whereby the second noun phrase is a predication over the first.

b. **Some of our colleagues**$_i$ are going to be supportive. **These kinds of people**$_i$ will earn our gratitude. – Coreferring noun phrases, whereby the second noun phrase is a predication over the first.



Coreference resolution

Coreference resolution is the task of clustering mentions in text that refer to the same underlying real-world entities. Coreference resolution, is the task of finding all expressions that are coreferent with any of the entities found in a given text. Coreference resolution is the task of resolving noun phrases to the entities that they refer to. Coreference resolution finds the mentions in a text that refer to the same real-world entity. For example, in the sentence, "Andrew said he would buy a car" the pronoun "he" refers to the same person, namely to "Andrew".



"I", "my", and "she" belong to the same cluster and "Obama" and "he" belong to the same cluster.

In computational linguistics, coreference resolution is a well-studied problem in discourse. To derive the correct interpretation of a text, or even to estimate the relative importance of various mentioned subjects, pronouns and other referring expressions must be connected to the right individuals. Algorithms intended to resolve coreferences commonly look first for the nearest preceding individual that is compatible with the referring expression. For example, she might attach to a preceding expression such as the woman or Anne, but not to Bill. Pronouns such as himself have much stricter constraints. Algorithms for resolving coreference tend to have accuracy in the 75% range. As with many linguistic tasks, there is a trade-off between precision and recall.

A classic problem for coreference resolution in English is the pronoun it, which has many uses. It can refer much like he and she, except that it generally refers to inanimate objects (the rules are actually more complex: animals may be any of it, he, or she; ships are traditionally she; hurricanes are usually it despite having gendered names). It can also refer to abstractions rather than beings: "He was paid minimum wage, but didn't seem to mind it." Finally, it also has pleonastic uses, which do not refer to anything specific:

a. It's raining.
b. It's really a shame.
c. It takes a lot of work to succeed.
d. Sometimes it's the loudest who have the most influence.

Approach to coreference resolution

Coreference Resolution in Two Steps

1. *Detect the mentions (easy)*
   "[I] voted for [Nader] because [he] was most aligned with [[my] values]," [she] said
   . mentions can be nested!
2. *Cluster the mentions (hard)*
   "[I] voted for [Nader] because [he] was most aligned with [my] values," [she] said

Mentions Detection

Mentions : span of text referring to some entity

Three kinds of mentions:

- Pronouns : I, your, it, she, him, etc. Use a part-of-speech tagger
- Named entities : People, places, etc.. Use a NER system
- Noun phrases : "a dog," "the big fluffy cat stuck in the tree" Use a constituency parser

Filter things that look referential but, in fact, are not – e.g.

- geographic names, the United State
- pleonastic "it", e.g. it's 3:45 p.m., it was cold –
- non-referential "it" , "they" , "there" e.g. it was essential, important, is understood, they say, there seems to be a mistake

All noun phrases (both indef. and def.) are considered potential referent candidates. – A referring phrase can also be a referent for a subsequent referring phrases, Example: He had 300 grams of plutonium 239 in his baggage. The suspected smuggler denied that the materials were his. (chain of 4 referring phrases) – All potential candidates are collected in a table collecting feature info on each candidate.

Features

Define features between a referring phrase and each candidate

– Number agreement: plural, singular or neutral

- He, she, it, etc. are singular, while we, us, they, them, etc. are plural and should match with singular or plural nouns, respectively
- Exceptions: some plural or group nouns can be referred to by either it or they IBM announced a new product. They have been working on it ...

– Gender agreement:

- Generally animate objects are referred to by either male pronouns he, his) or female pronouns (she, hers)
- Inanimate objects take neutral (it) gender

– Person agreement:

- First and second person pronouns are "I" and "you"
- Third person pronouns must be used with nouns

– Binding constraints

Reflexive pronouns (himself, themselves) have constraints on which nouns in the same sentence can be referred to:

John bought himself a new Ford. (John = himself)

John bought him a new Ford. (John cannot = him)

- Recency

Entities situated closer to the referring phrase tend to be more salient than those further away And pronouns can't go more than a few sentences away

- Grammatical role / Hobbs distance

Entities in a subject position are more likely than in the object position

- Repeated mention

Entities that have been the focus of the discourse are more likely to be salient for a referring phrase

-Parallelism

There are strong preferences introduced by parallel constructs Long John Silver went with Jim. Billy Bones went with him. (him = Jim)

-Verb Semantics and selectional restrictions

Certain verbs take certain types of arguments and may prejudice the resolution of pronouns John parked his car in the garage after driving it around for hours.

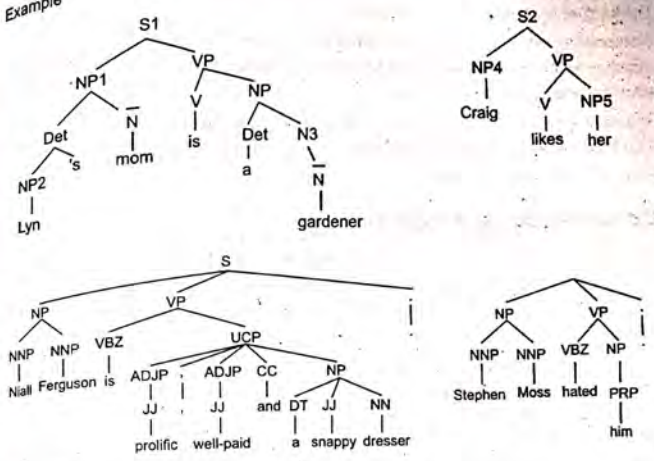
Hobb's Algorithm

Hobbs' algorithm was one of the earliest approaches to pronoun resolution. The algorithm is mainly based on the syntactic parse tree of the sentences. It makes use of syntactic constraints when resolving pronouns. First, intra-sentential antecedents are proposed - the syntactic tree of the current sentence is searched in a breadth-first left to-right fashion to find antecedents. The contra-indexing constraint is taken care inside the algorithm, by making sure that the path from the NP to the S node of the syntactic tree has at least one another NP on the way. If there are higher-level nodes in the current sentence, then antecedents resulting from a breadth-first left-to-right search of each subtree, are proposed. Then, parse trees of previous sentences in reverse chronological order are searched in the same fashion to propose antecedents. In essence, Hobbs' algorithm prefers entities that are within the same sentence, and entities that are closer pronoun in the same sentence. Depending on the position of the pronoun in the sentence, different entities in a sentence may become more relevant. When looking for antecedents in previous sentences, the antecedents that occur (or are realized) in the subject position are more salient, since a breadth-first left-to-tree search is performed starting at the root S node of the sentence. Depth of a node in the syntactic tree is thus a very important factor to determine discourse prominence. Ref

1. Start with target pronoun
2. Climb parse tree to S root
3. For each NP or S
   a. Do breadth-first, left-to-right search of children
   b. Restricted to left of target
   c. For each NP, check agreement with target

4. Repeat on earlier sentences until matching NP found

Example



Why Coreference Resolution is Hard

Entities can be very long and coreferent entities can occur extremely far away from one another. A greedy system would compute every possible span (sequence) of tokens and then compare it to every possible span that came before it. This makes the complexity of the problem O(T4), where T is the document length. For a 100 word document this would be 100 million possible options . If this does not make it concrete, imagine that we had the sentence

* Arya Stark walks her direwolf, Nymeria. *

Here we have three entities: Arya Stark, her, and Nymeria. As a native speaker of English it should be trivial to tell that her refers to Arya Stark. But to a machine with no knowledge, how should it know that Arya and Stark should be a single entity rather than two separate ones, that Nymeria does not refer back to her even though they are arguably related, or

even that that Arya Stark walks her direwolf, Nymeria is not just one big entity in and of itself?

For another example, consider the sentence

* Napoleon and all of his marvelously dressed, incredibly well-trained, loyal troops marched all the way across Europe to enter into Russia in an ultimately unsuccessful effort to conquer it for their country. *

The word their is referent to Napoleon and all of his marvelously dressed, incredibly well trained, loyal troops; entities can span many, many tokens. Coreferent entities can also occur far away from one another.

Coreference vs. Anaphora

Coreference resolution is often confused with the task of anaphora resolution. Two noun phrases are said to be co-referring to each other if both of them resolve to a unique referent (unambiguously). However, a noun phrase A is said to be the anaphoric antecedent of a noun phrase B, if and only if A is required for the interpretation of B. Thus, coreference is an equivalence relation, whereas anaphora is neither reflexive nor symmetric (nor transitive). Every speaker had to present his paper. Here, if "his" and "every speaker" are said to co-refer (i.e., considered the same entity), the sentence is interpreted as "Every speaker had to present Every speaker's paper" which is obviously not correct. Thus, "his" here is an anaphoric referent and not coreferential

A number of coreferential links could be anaphoric relations but some anaphora relations such as bound anaphora are not coreference links. For example, sentences like the following contain bound anaphora: a. Every dog has its day. b. The man who gave his paycheck to his wife was wiser than the man who gave it to his mistress. where the anaphor and the antecedent are not coreferent. Further, there could be other anaphoric relations like the following: The boy entered the room. The door closed automatically. where the relation between the room and the door is that of meronomy/holonomy. In these cases, the two noun phrases do not refer to a single entity. Other contentious cases exist such as concepts that vary with time such as the President of a nation, the price of a stock, etc. The problem with defining coreference resolution this way is the inappropriate use of the term "coreference" to cover semantic relations such as those involving temperature and price.

Application of Coreference Resolution

The Coreference Resolution can be applied to multiple Natural Language Processing (NLP) tasks, such as name entity recognition (NER), translation, question answering and text summarization, in a meaningful way. For example, for name entity recognition (NER) task, the Coreference Resolution helps to detect the references between entities and pronouns. On the one hand, if pronouns are replaced by coreferred entities, the incremented frequency of entities in the document will enhance the accuracy of the NER model. On the other hand, it is not necessary to detect name entity word by word if we have already known all words that refer to the same real-world entity. Ref

Another example, in a French-English translation system, we have a sentence "Barack Obama a un chien. Son chien est grand." which in English is "Barack Obama has a dog. His dog is big." However, the translator may mis-recognize "his dog" as "her dog" because of the lack of gender information. If we introduce the Coreference Resolution into this translation system, then the translator can understand "son chien" is the dog of Barack Obama, and he is a man. Applying the Coreference Resolution to these NLP tasks, especially NLU tasks, can eventually increases their performances.

Expected Questions

1. Define discourse and pragmatic analysis . Discuss reference resolution problem in detail.
2. Discuss following referring expressions with suitable examples w.r.t reference phenomena
   Indefinite NPs Definite NPs Pronouns Demonstratives and Oneanaphora.
3. Explain three types of referents that complicate the reference resolution problem.
4. Write a note on Syntactic and Semantic Constraints on Coreference.
5. Discuss various Preferences in Pronoun Interpretation with suitable example.
6. Explain in detail Lappin and Leass's Algorithm for Pronoun Resolution with suitable Example.
7. Describe in detail centering algorithm for reference resolution.

6 Applications (Preferably for Indian Regional Languages)

OBJECTIVES

After reading this chapter, the student will be able to understand:

- Machine translation
- Information retrieval
- Question answers system
- Categorization
- Summarization
- Sentiment analysis
- Named Entity Recognition

1. Machine Translation

Machine Translation (MT) is a standard name for computerized systems responsible for the production of translations from one natural language into another with or without human assistance. It is a subfield of computational linguistics that investigates the use of computer software to translate text or speech from one language to another. Development of a fully-fledged bilingual machine translation (MT) system for any two natural languages with limited electronic resources and tools is challenging and demanding task. In order to achieve reasonable translation quality in open source tasks, corpus-based machine translation approaches require large number of parallel corpora that are not always available, especially for less resourced language pairs. On the other hand, the rule based machine translation process is extremely difficult and fails to analyze accurately a large corpus of unrestricted text. Even though there has been an effort towards building English to Indian Language and Indian Language to Indian Language translation system, we do not have an efficient translation system as of today.

Why should we interest in using computers for translation at all? The first most important reason is that there is too much that needs to be translated and human translator cannot cope. A second reason is that technical materials are too boring for human translators, they don't like translating them, so they will look for help from computers. Another reason is that, in corporation area, there is a major requirement that terminology is used consistently; Computers are consistent but human translator tend to seek variety; they do not like to repeat the same translation and it will be not good for technical translation. Computer based translation can increase the volume and speed of translation and corporate area like to have translations immediately, the next day or even the same day.

Machine Translation System Approaches

Like translation done by humans, MT does not simply substitute words nut the application of linguistic knowledge, morphology, grammar, meaning; all this has to be taken into consideration. Generally MT system is classified into various categories: Direct based, Rule based, Corpus based, Statistical based, Hybrid based, Example based, Knowledge based, Principle based and Online Interactive based systems .

1. **Direct Translation**

   Direct Machine Translation is one of the simplest machine translation approach in which a direct word to word translation is done with the help of a bilingual dictionary.

2. **Rule Based Translation**

   A Rule-Based Machine Translation (RBMT) system consists of a collection of various rules, called grammar rules, a bilingual lexicon or dictionary, and software programs to process the rules.

3. **Interlingua Based Translation**

   In this approach, the translation consists of two stages, where the source Language (SL) is first converted into the Interlingua (IL) form. The main advantage of Interlingua approach is that the analyzer and parser of SL is independent of the generator for the Target Language (TL) and this requires complete resolution of ambiguity in source language text.

4. **Statistical-based Approach**

   Statistical machine translation (SMT) is a data-oriented statistical framework which is based on the knowledge and statistical models which are extracted from bilingual corpora. In this MT, bilingual or multilingual corpora of the languages are required. In SMT, a document is translated according to the probability distribution function which is indicated by p(e|f). Finding the best translation is done by picking the highest probability, as shown in Equation 1.

   $$e = \text{argmax}\ p(e \mid f) = \text{argmax}\ p(f \mid e)\ p(e) \quad \text{.....(i)}$$

5. **Example-based translation**

   Basic idea of this MT is to reuse the examples of already existing translations. An example-based translation is uses a bilingual corpus as its main knowledge base and it is essentially translation by analogy.

6. **Knowledge-Based MT**

   Knowledge-Based Machine Translation (KBMT) requires complete understanding of the source text prior to the translation into the target text. KBMT is implemented on the Interlingua architecture. KBMT must be supported by world knowledge and by linguistic semantic knowledge about the meanings of words and their combinations.

7. **Principle-Based MT**

Principle-Based Machine Translation (PBMT) Systems are based on the Principles a Parameters Theory of Chomsky's Generative Grammar and which employs parsing method.

In this, the parser generates a detailed syntactic structure which contains lexical, phrasal, grammatical information.

8. **Online Interactive Systems**

In this online interactive translation system, the user has authority to give suggestion for the correct translation. This approach is very useful, where the context of a word is not that much clear or unambiguous and where multiple possible meanings for a particular word.

9. **Hybrid-based Translation**

By taking the advantage of statistical MT and rule-based MT methodologies, a new approach was developed, which is termed as "hybrid-based approach". The hybrid approach used in a number of different ways. Translations are performed in the first stage using a rule-based approach which is followed by adjusting or correcting the output using statistical information. Second way in which rules are used to pre-process the input data and for post-process the statistical output of a statistical-based translation system.

Rule Based Machine Translation System

A Rule based Machine Translation (RBMT) System consist of collection of Rules, which is called as grammar rules, a bilingual or multilingual lexicon and software programs to process the rules.

Building RBMT system entails a huge human effort to code all of the linguistic resources, such as source side part-of-speech taggers and syntactic parsers, bilingual dictionaries, source to target transliteration, target language morphological generator, and structural transfer, and so many reordering rules. A RBMT system is always extensible and maintainable. Rules play a major role in various stages of translation, such as syntactic processing, semantic interpretation and contextual processing of language. Generally rules are written with linguistic knowledge gathered from linguistics. Transfer based MT, Interlingua MT and direct based MT are three different approaches that come under the RBMT category.

In the case of English to Indian languages and Indian languages to Indian languages MT system, there have been fruitful attempts with all these approaches. The main idea behind these rule-based approaches is as follows

1. **Direct Translation:**

In Direct Translation method, source language (SL) text is analysed structurally up to the morphological level and is designed for a specific source and target language pair. The performance of direct based MT systems depends on the quality and quantity of the source-target language dictionaries, morphological analysis, text processing software and word-by-word translation with major grammatical adjustments on word order and morphology.

2. **Interlingua Translation:**

The next stage in the development of MT systems is the Interlingua approach, where translation is performed by first representing the SL text into an intermediary (semantic) form called Interlingua. In this approach, the analyzer of parser for the SL is independent on the generator of TL. But this have certain disadvantages that, difficulty in defining the Interlingua and Interlingua does not take advantage of similarities between the languages.

3. **Transfer Based Translation:**

Because of the disadvantages of the Interlingua approach, a rule-based translation approach was discovered, called the transfer approach. On the basis of the structural difference between the source and target language, a transfer system can be broken down into three different stages: i) Analysis ii) Transfer and iii) Generation.

Example: English to Marathi Machine Translation System

The idea is to translate an Input document in English Natural Language to Marathi language document by going through various phases such as pre-processing, lexical phase , syntax phase , semantics phase and finally translating into target language using various mapping rules.

The translation process has 2 major stages involved in it as :

1. Decoding the meaning of source text and
2. Re-encoding this meaning in the target language.

Like translation done by humans, MT does not simply involve substituting words in one language for another, but the application of complex linguistics knowledge: morphology, syntax, semantics and understanding of concepts such as ambiguity.

Proposed Machine translation System consists of 3 major phases:

1. **Pre-Processing Phase:**

   This is the first phase of any machine translation system. This phase makes machine translation process easier and qualitative. The source language text may contain figures, flowcharts, etc that do not require any translation. So only textual portion from the document should be identified. Fixing up punctuation marks and blocking content which does not require translation are also done during pre-processing module. Pre-processing phase consists of main processes as Morphology analysis, Named Entity Recognition (NER) and syntactic analysis. The efficiency of the MT system is always dependent on the quality of pre-processing stage.

2. **Transfer and Generation Phase:**

   This is the second phase of machine translation system. This module composes the meaning representations and assigns them the linguistic inputs. The semantic analyser uses lexicon or dictionary and grammar to create context dependent meaning. The source of knowledge consist of meaning of words, meaning associated with grammatical structures, knowledge about the discourse context and common sense knowledge

3. **Post-Processing Phase:**

   This is the last and most important phase of any machine translation process. Once the text is translated the target text is to be reformatted after post-editing. Post editing is done to make sure that the quality of the translation is upto the mark. Post-editing is unavoidable especially for translation of crucial information.

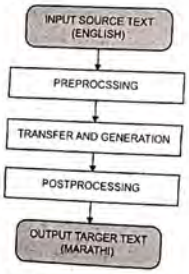


Figure 1: Proposed Architecture of English-Marathi Rule Based Machine Translation System

Algorithm of English to Marathi MT System:

Input: Digital document as input in English Natural language.

Output: Translated Document in Marathi Natural language.

1. Accept a digital document as input.
2. For each sentence in input document apply POS tagging.
3. Generate the parse tree for each sentence.
4. Apply NER on each sentence.
5. Identifies and names thematic relations between a verb and a noun in the sentence & Represents the tokens in the intermediate language based on target language reordering rules.
6. Use a bilingual dictionary to obtain appropriate translation & transliteration of the lemmas.
7. Obtain the proper form of words using inflections.
8. Represent sentences into target language.

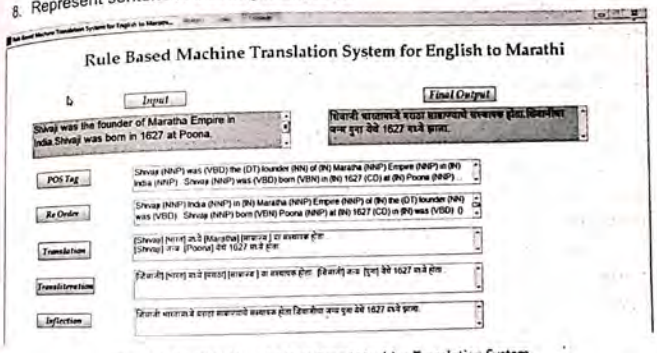


Figure 2: English-Marathi Rule Based Machine Translation System

Language technologies in general and machine translation system in particular are the instruments for narrowing the digital divide and overcoming language barriers to facilitate communication and commerce.

2. Information Retrieval

Information Retrieval remains one of the most challenging problems in NLP. Hundreds of millions of people engage in information retrieval everyday while using a web search engine or searching emails. Traditional database searching is becoming obsolete since most of the times the user is unclear what he or she is searching for. Information retrieval system is one that searches a collection of natural language documents with the goal of retrieving exactly the set of documents that pertain to a user's question"

A lot of readily available information is available through the World Wide Web which gets updated every time and is reached out to people all over the world. Thus, searching applications has changed tremendously from systems designed for specific applications belonging to well defined group to systems which are applicable for common people. However the huge and undefined structure of information present over networks has made it difficult for users to search and find relevant information. Many information retrieval techniques have been developed to deal with this problem.

Information retrieval (IR) may be defined as a software program that deals with the organization, storage, retrieval and evaluation of information from document repositories particularly textual information. The system assists users in finding the information they require but it does not explicitly return the answers of the questions. It informs the existence and location of documents that might consist of the required information. The documents that satisfy the user's requirement are called relevant documents. A perfect IR system will retrieve only relevant documents

Traditionally, the information retrieval system techniques are based on keyword. They use lists of keywords to describe the content of information but they do not say reveal semantic relationships between keywords nor consider the meaning of words and phrases. For example, the search engines accept keywords and in return they show a list of links to documents containing those keywords.

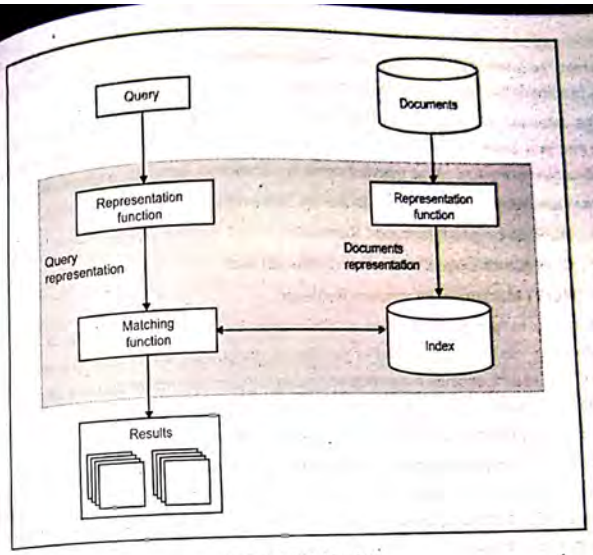


Figure 3: Basic IR system

Basic IR system involves following stages:

1. Indexing the collection of documents.
2. Transforming the query in the same way as the document content is represented.
3. Comparing the description of each document with that of the query.
4. Listing the results in order of relevancy.

In general all information Retrieval Systems consist of mainly two processes as:

1. **Indexing:** Indexing is the process of selecting terms to represent a text which involves tokenization of string, removing frequent words and stemming.
2. **Matching**: Matching is the process of computing a measure of similarity between two text representations. Relevance of a document is computed based on parameters like term frequency and inverse document frequency .

Information retrieval (IR) system aims to retrieve relevant documents to a user query where the query is a set of keywords. The ultimate goal of an information retrieval process is to retrieve the information relevant to a given request.

The criterion for a successful IR system is to retrieve all the relevant information items stored in a given system, and to reject the irrelevant information. However, the results will contain a mixture of the relevant items and irrelevant items for a given request.

There are many variants of the Information Retrieval systems such as :-

1. **BLIR** (Bi-Lingual Information Retrieval)
2. **CLIR** (Cross-Lingual Information Retrieval) and
3. **MLIR** (Multilingual Information Retrieval).

The ability to search and retrieve information in multiple languages is becoming challenging. So multilingual and cross-lingual (language) information retrieval (MLIR and CLIR) search engines have received more research attention and are being used to retrieve information on the Internet on a large scale. CLIR refers to searching, translating and retrieving information between a source language and target language.

Cross-lingual IR has become more important in recent years. Cross Language Information Retrieval (CLIR) can be described as the task of retrieving documents across languages. The basic idea behind the cross-lingual IR(CLIR) is to retrieve documents in a language different from the language used by the user to develop the query. This may be desirable even when the user is not a speaker of the language used in the retrieved documents. CLIR uses different translation approaches to translate queries to documents and indexes in other languages. Some CLIR systems use language resources such as bilingual dictionaries to translate the user's original query, while other systems use machine translation to translate the foreign-language documents beforehand, enabling them to be retrieved by the original query.

This task represents one extreme case of the vocabulary mismatch problem, i.e. The vocabulary of a user query and the vocabulary of relevant documents can differ substantially. The bag-of-words (BOW) model seriously suffers from the vocabulary mismatch problem because of different dimensions thus neglecting relations between different words in the same language as well as across languages. Therefore, the challenging task of retrieving documents to queries in other languages requires models other than traditional bag-of-words model.

3. Question Answering System

Humans are always in a quest to extract information related to some topic or entity. Question answering system helps user to find the precise answer to the question articulated in natural language. Question answering system provides explicit, concise and accurate answer to user questions rather than providing a set of relevant documents or web pages as answers as most of the information retrieval system does.

Any question answering system basically consists of three parts as question processing, answer retrieval and answer generation. In question processing users natural language questions are parsed to formulate questions in machine readable form using different approaches. Then in answer retrieval candidate answers are extracted based on intermediate representation of question. Finally in answer generation phase user understandable precise and accurate answer is generated and provided to user.

Any question answering system can be classified into two main types: close domain QAS and open domain QAS. In close domain QAS scope of user question is limited to a particular domain like medicine, movies, history and others. So if a QAS is created for history for history domain it will provide answers to questions related to history only. An open domain QAS mostly works like search engines like Google and all where it provides explicit answers to question belonging to any domain. So in open domain QAS the scope for question is global.

Question in any question answering system can be of varying types. Question can be factoid question for which answers are simple fact about the entity in question. Some questions can be of descriptive type where one needs to full detail about a person, place or any event. There can be simple yes/no type of question which simple provides answers as yes or no. A question can also be an instruction-based question where answers are provided as an instruction to accomplish any task. Question in a QAS can be of many other forms which provide precise answer in the same format as that of question provided.

Example: Question Answering System for Hindi and Marathi language

The Question Answering System for Hindi and Marathi language is as shown in Figure 6.3. Here input to the system is natural language Hindi or Marathi Query. Input query is first tokenized to generate individual token and then these tokens undergo word grouping where two or three corresponding words are merged together if they are related with each other by using the available word grouped list. POS tagging is performed on word

grouped tokenized query text to extract relevant part of speech associated with the query text. POS tagged query text then passes through chunking process where noun and verb grouped present in the query text are extracted. Based on the extracted chunked groups initially query triples are extracted using Subject, Object and Verb (SOV). Then next process is to generate onto triples by fetching relevant onto words from ontology. Finally ontology is traversed to fetch relevant answer based on generate onto triples, if onto triple matches with any onto set in ontology then corresponding answer is fetched and passed to answer generation process to present the answer as natural way as possible mostly in the form of natural language text. The system for Hindi Question Answering System and Marathi Question Answering System both follow the same flow and have the same basic architecture with exception in Marathi that it requires separate case extraction phase, because case marker are mainly attached to noun words as suffixes, before extraction of SOV from query text as Marathi is much more inflected language compared to Hindi.

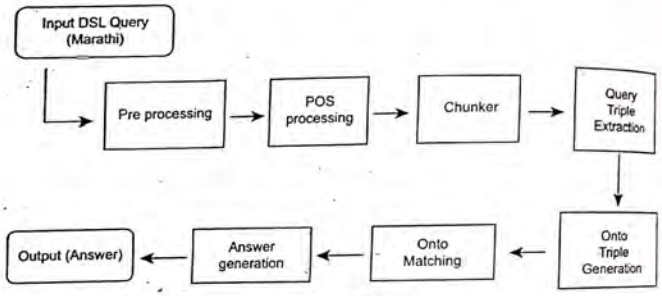


Figure 4: Basic QA system

The overall algorithm for proposed QA system is as follows:

Input: Natural language question in Hindi/Marathi language

Output: Answer in Hindi/Marathi language

Step 1: Tokenize the input question into word tokens.

Step 2: Grouped the correlated words into one merged word.

Step 3: Extract POS tag of each word in the tokenized list.

Step 4: Chunk the POS tagged words into noun and verb groups.

Step 5: Extract query triple from chunked grouped list.

Step 6: Generate onto triple.

Step 7: Traverse ontology to fetch answer.

Step 8: Formulate answer as natural language text.

Sample input and output for Hindi query:

Input Question:शिवाजी की माँ कौन थी?

Answer: शिवाजी की माँ जीजाबाई थी।

Sample input and output for Marathi query:

Input Question: शिवाजिची आई कोण होती?

Answer: शिवाजिची आई जीजाबाई होती.

The question answering system for Marathi natural language using concept of ontology as a formal representation of knowledge base for extracting answers. Ontology is used to express domain specific knowledge about semantic relations and restrictions in the given domains. The ontologies are developed with the help of domain experts and the query is analyzed both syntactically and semantically. The results obtained are accurate enough to satisfy the query raised by the user. The level of accuracy is enhanced since the query is analyzed semantically.

A question answering system is much more effective than traditional search engines as it provides accurate and precise answer to question rather than providing links to relevant documents or set of matching contents. Question answering system has become part of daily life of users, over a period of time many personal assistance software like Siri, Cortana, and Google Now are developed which provide precise and accurate answer to user question.

4. Sentiment Analysis

Sentiment Analysis (SA) is a natural language processing task that deals with finding orientation of opinion in a piece of text with respect to a topic. It deals with analyzing emotions, feelings, and the attitude of a speaker or a writer from a given piece of text. Sentiment Analysis involves capturing of user's behaviour, likes and dislikes of an individual from the text. The target of SA is to find opinions, identify the sentiments they express, and then classify their polarity.

The purpose of sentiment analysis is to determine the attitude or inclination of a communicator through the contextual polarity of their speaking or writing. Their attitude may be reflected in their own judgment, emotional state of the subject, or the state of any emotional communication they are using to affect a reader or listener. It is trying to determine a person's state of mind on the subject they are communicating about. This information can be mined from texts, tweets, blogs, social media, news articles, or comments.

There are different classification levels in SA: document-level, sentence-level and aspect-level. Document-level SA aims to classify an opinion of the whole document as expressing a positive or negative sentiment. Sentence-level SA aims to classify sentiment expressed in each sentence which involves identifying whether sentence is subjective or objective. Aspect-level SA aims to classify the sentiment with respect to the specific aspects of entities which is done by identifying the entities and their aspects

Sentiment Analysis is a natural language processing task that deals with finding orientation of opinion in a piece of text with respect to a topic. It deals with analyzing emotions, feelings, and the attitude of a speaker or a writer from a given piece of text.

Sentiment Classification Techniques

Sentiment classification is a task under Sentiment Analysis (SA) that deals with automatically tagging text as positive, negative or neutral from the perspective of the speaker/writer with respect to a topic. Thus, a sentiment classifier tags the sentence 'the movie is entertaining and totally worth your money!' in a movie review as positive with respect to the movie. On the other hand, the sentence 'The movie is so boring that I was dozing away through the second half.' is labelled as negative. Finally, 'The movie is directed by Nolan' is labelled as neutral. There are two main techniques for sentiment classification: machine learning based and lexicon based. Better performance can be obtained by combining these two methods.

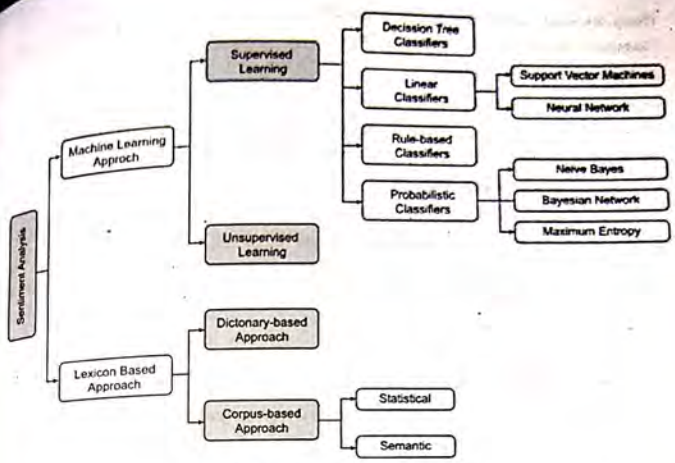


Figure 5: Sentiment Classification Techniques

Machine Learning Approach

The machine learning method uses several learning algorithms to determine the sentiment by training on a known dataset. The machine learning approach applicable to sentiment analysis mostly belongs to supervised classification. In a machine learning based techniques, two sets of documents are needed: training and a test set. A training set is used by an automatic classifier to learn the differentiating characteristics of documents, and a test set is used to check how well the classifier performs.

A) Supervised Learning:

The supervised learning methods depend on the existence of labeled training documents. Supervised learning process: two Steps; Learning (training): Learn a model using the training data testing: Test the model using unseen test data to assess the model accuracy.

There are many kinds of supervised classifiers like Decision Tree Classifiers, Linear Classifiers, Rule-based Classifiers and Probabilistic Classifiers.

I) Probabilistic Classifier

Probabilistic classifiers use mixture models for classification. The mixture model assumes that each class is a component of the mixture. Each mixture component is a generative model that provides the probability of sampling a particular term for that component.

a) Naive Bayes Classifier (NB)

Naive Bayes classification model computes the posterior probability of a class, based on the distribution of the words in the document. The model works with the BOWs feature extraction which ignores the position of the word in the document. It uses Bayes Theorem to predict the probability that a given feature set belongs to a particular label.

P(label | features)=(P(label)*P(features|label))/(P(features))

b) Bayesian Network (BN)

Bayesian Network model which is a directed acyclic graph whose nodes represent random variables and edges represent conditional dependencies. BN is considered a complete model for the variables and their relationships. Therefore, a complete joint probability distribution (JPD) over all the variables is specified for a model. In Text mining, the computation complexity of BN is very expensive; that is why, it is not frequently used.

c) Maximum Entropy Classifier (ME)

The Maxent Classifier also known as a conditional exponential classifier converts labeled feature sets to vectors using encoding. This encoded vector is then used to calculate weights for each

feature that can then be combined to determine the most likely label for a feature set. This classifier is parameterized by a set of X {weights}, which is used to combine the joint features that are generated from a feature-set by an X {encoding}. In particular, the encoding maps each C {(feature set, label)} pair to a vector.

II) Linear classifiers

Score (or probability) of a particular classification is based on a linear combination of features and their weights. A linear classifier determines which class a object belongs by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are

typically presented to the machine in a vector called a feature vector. There are many kinds of linear classifiers; among them is Support Vector Machines (SVM) which is a form of classifiers that attempt to determine good linear separators between different classes.

a) Support Vector Machines Classifiers (SVM)

The main principle of SVMs is to determine linear separators in the search space which can best separate the different classes. Text data are ideally suited for SVM classification because of the sparse nature of text, in which few features are irrelevant, but they tend to be correlated with one another and generally organized into linearly separable categories. The basic SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the output, making it a non-probabilistic binary linear classifier. It does not depend on the probabilities.

b) Neural Network (NN)

Neural Network consists of many neurons where the neuron is its basic unit. A neural network consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand. This is the learning phase. Multilayer neural networks are used for non-linear boundaries. These multiple

layers are used to induce multiple piecewise linear boundaries, which are used to approximate enclosed regions belonging to a particular class. The outputs of the neurons in the earlier layers feed into the neurons in the later layers. The training process is more complex because the errors need to be back-propagated over different layers.

III) Decision tree classifier

Decision tree classifier provides a hierarchical decomposition of the training data space in which a condition on the attribute value is used to divide the data. The condition or predicate is the presence or absence of one or more words. The division of the data space is done recursively until the leaf nodes contain certain minimum numbers of records which are used for the purpose of classification.

IV) Rule based classifiers

In rule-based classifiers, the data space is modeled with a set of rules. The left-hand side represents a condition on the feature set expressed in disjunctive normal form while the right-hand side is the class label. The conditions are on the term presence. Term absence is rarely used because it is not informative in sparse data. There are numbers of criteria in order to generate rules, the training phase construct all the rules depending on these criteria. The most two common criteria are support and confidence. The support is the absolute number of instances in the training data set which are relevant to the rule. The Confidence refers to the conditional probability that the right-hand side of the rule is satisfied if the left-hand side is satisfied. Both decision trees and decision rules tend to encode rules on the feature space, but the decision tree tends to achieve this goal with a hierarchical approach.

B) Unsupervised learning:

Unsupervised learning is that of trying to find hidden structure in unlabelled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. The main purpose of text classification is to classify documents into a certain number of predefined categories. In order to accomplish that, a large number of labelled training documents are used for supervised learning, as illustrated before. In text classification, it is sometimes difficult to create these labelled training documents, but it is easy to collect the

unlabelled documents. The unsupervised learning methods overcome these difficulties.

K-nearest neighbors (KNN) is a simple unsupervised machine learning algorithm. In this algorithm, the objects are classified based on the majority of its neighbor. The class assigned to the object is most among its k nearest neighbors. The KNN classification algorithm classifies the instances of objects based on their similarities to instances in the training data. In KNN, selection is based on majority voting or distance weighted voting. Consider the vector A and set of M labelled instances {ai, bi}. The classifier predicts the class label of A on the predefined N classes. The KNN classification algorithm finds the k nearest neighbors of A and determines the class label of A using majority vote. KNN classifier applies Euclidean distance as the distance metric.

Lexicon-based approach

The lexicon-based approach involves calculating sentiment polarity for a review using the semantic orientation of words or sentences in the review. The semantic orientation is a measure of subjectivity and opinion in text. Sentiment lexicon contains lists of words and expressions used to express people's subjective feelings and opinions. For example, start with positive and negative word lexicons, analyze the document for which sentiment need to find. Then if the document has more positive word lexicons, it is positive, otherwise it is negative. The lexicon based techniques to Sentiment analysis is unsupervised learning because it does not require prior training in order to classify the data.

There are three methods to construct a sentiment lexicon: manual construction, corpus-based methods and dictionary-based methods. The manual construction of sentiment lexicon is a difficult and time-consuming task. In dictionary based techniques the idea is to first collect a small set of opinion words manually with known orientations, and then to grow this set by searching in the WordNet dictionary for their synonyms and antonyms. The newly found words are added to the seed list. The next iteration starts. The iterative process stops when no more new words are found. The dictionary based approach have a limitation is that it can't find opinion words with domain specific orientations. Corpus based techniques rely on syntactic patterns in large corpora. Corpus-based methods can produce opinion words with relatively high accuracy. Most of these corpus based methods need very large labeled training data. This approach has a major advantage that the dictionary-based approach does not have. It can help find domain specific opinion words and their orientations

Basic sentiment analysis of text documents follows a straightforward process:

- Break each text document down into its component parts (sentences, phrases, tokens and parts of speech)
- Identify each sentiment-bearing phrase and component
- Assign a sentiment score to each phrase and component (-1 to +1)
- Optional: Combine scores for multi-layered sentiment analysis

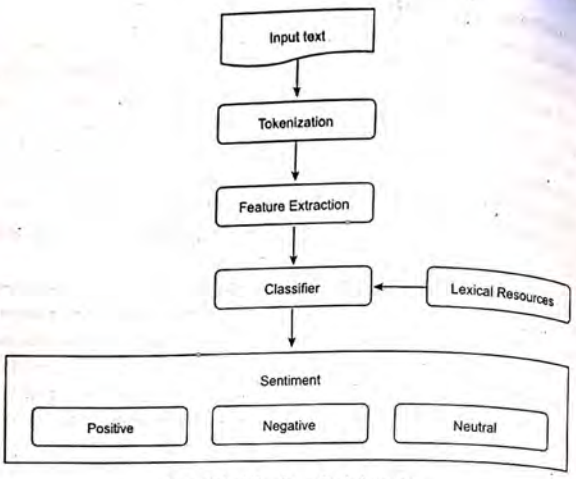


Figure 6: Basic Sentiment Analysis system

Example: Sentiment classification for Hindi language document using HindiSentiWoedNet

Problem Statement:

The proposed system extract sentiment associated with Hindi documents using Improved-HindiSentiWordNet (HSWN). The overall polarity of the document is calculated which can be positive, negative or neutral. The input to the system is a single text document in Hindi and output will be overall polarity associated with document.

The system consists of two stages:

1. Improving HindiSentiWordNet (HSWN) .
2. Sentiment extraction

During the first stage , the existing HSWN is improved with the help of English SentiWordNet, where sentimental words which are not present in the HSWN are translated to English and then searched in English SentiWordNet to retrieve their polarity.

In the second stage, sentiment is extracted by finding the overall polarity of the document; which can be positive, negative or neutral. Here during preprocessing tokens are extracted from sentences and stop words are removed. Rules are devised for handling negation and discourse relation which highly influenced the sentiments expressed in the document. Finally, overall sentiment orientation of the document is determined by aggregating the polarity values of all the sentimental words in the document.

Input: Accepts a single document in Hindi as input.

Output: Overall sentiment associated with document.

Input Text:- सूरज बड़जात्या की रची दुनिया की फिल्म है। यह मूवी अच्छा नहीं है। फिल्म कई जगह चमक छोड़ती है मगर बात बन नहीं पाती। फिल्म का अंतिम भाग अच्छा नहीं था।

:Pre-Processed Text	:Negation & Discourse Handled Text
Sentence 0 : सूरज बड़जात्या रची दुनिया फिल्म	Sentence No.0 : सूरज बड़जात्या रची दुनिया फिल्म
Sentence 1 : मूवी अच्छा नहीं	Sentence No.1 : मूवी !अच्छा नहीं
Sentence 2 : फिल्म जगह चमक छोड़ती मगर बात बन नहीं पाती	Sentence No.2 : मगर बात !बन नहीं पाती
Sentence 3 : फ़िल्म अंतिम भाग अच्छा नहीं	Sentence No.3 : फ़िल्म अंतिम भाग !अच्छा नहीं

Extracted Polarity:

	Words found in improved HSWN : सूरज अच्छा नहीं बात बन नहीं अच्छा नहीं	
नहीं (P 0.0) (N 0.125)	!अच्छा (P 1.0) (N 0.375) (TP -0.625)	सूरज (P 0.125) (N 0.0) (TP 0.125) ((TP -0.125
नहीं (P 0.0) (N 0.125) (TP	!बन (P 0.01) (N 0.0) (TP -0.01)	बात (P 0.25) (N 0.0) (TP 0.25) (-0.125
	(नहीं (P 0.0) (N 0.125) (TP -0.125	अच्छा (P 1.0) (N 0.375) (TP -0.625)(
	Total Positive polarity : 1.125	**Positive words count : 2**
	Total Negative polarity : 2.385	**Negative words count : 6**
		Overall polarity : -1.26

Overall Sentiment: Negative

Figure 7: Sentiment classification for Hindi documents

Sentiment Analysis has been quite popular and has lead to building better products, understanding user's opinion, executing and managing of business decisions. People rely and make decisions based on the reviews and opinions.

5. Text Categorization or Classification

Text categorization is the process of assigning tags or categories to text according to its content. It's one of the fundamental tasks in Natural Language Processing (NLP) with broad applications such as sentiment analysis, topic labelling, spam detection, and intent detection.

Unstructured data in the form of text is everywhere: emails, chats, web pages, social media, support tickets, survey responses, and more. Text can be an extremely rich source of information, but extracting insights from it can be hard and time-consuming due to its unstructured nature. Businesses are turning to text classification for structuring text in a fast and cost-efficient way to enhance decision-making and automate processes.

Text classification is a process of dividing a given set of documents into one or more predefined classes. This classification of text is done automatically. Usually machine learning techniques are used for automatic text classification. There are mainly two types of techniques namely supervised and unsupervised learning methods. Supervised learning methods assign predefined class label to the testing documents using classification algorithms whereas in unsupervised learning methods grouping of testing documents are done using techniques like clustering. There is also a semi-supervised learning method where, parts of the documents are labelled by the external mechanism.

Text classifiers can be used to organize, structure, and categorize pretty much anything. For example, new articles can be organized by topics, support tickets can be organized by urgency, chat conversations can be organized by language, brand mentions can be organized by sentiment, and so on.

Text Classification Techniques

A growing number of machine learning approaches or more specifically supervised learning methods have been applied to text classification which include Decision tree(C 4.5), K-Nearest Neighbor (K-NN), Bayesian approaches (Naïve and non-Naïve approaches), Neural Networks (NN), Regression based methods, Vector-based methods etc. Several clustering techniques are also available for text classification like K-means, Suffix Tree Clustering, Label Induction Grouping (LINGO) algorithm, Semantic Online Hierarchical Clustering (SHOC) etc.

A) Supervised Learning methods

The supervised learning techniques are explained below:

a) **Decision Tree**

Decision tree methods reconstruct the manual categorization of the training documents in the form of a tree structure where the nodes represent questions and the leaves represent the corresponding category of documents. When the tree has created, a new document can simply be categorized by placing it in the root node of the tree and let it run through the query structure until it reaches a certain leaf.

b) **K-Nearest Neighbor (KNN)**

KNN is a statistical approach for text classification where objects are classified by voting several labeled training examples with their smallest distance from each object. The KNN classification method is outstanding with its simplicity and is widely used techniques for text categorization.

c) **Neural Network**

Neural network is also called artificial neural network is a mathematical model inspired by biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. Different neural network approaches have been applied to document categorization problems. While some of them use the simplest form of neural networks, known as perceptions, which consist only of an input and an output layer, others build more sophisticated neural networks with a hidden layer between the two others.

d) **Naïve Bayes (NB)**

A naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes theorem with strong independence assumptions. A naïve Bayes classifier assumes that the presence or absence of a particular feature of a class is unrelated to the presence or absence of any other feature. Depending on the precise nature of the probabilistic model, Naïve Bayes classifiers can be trained very efficiently in a supervised learning setting.

e) **Vector Based Methods**

The two types of vector-based methods: The centroid algorithm and support vector machines. From these two algorithms centroid is simplest.

Centroid Algorithm: During the learning stage only the average feature vector for each category is calculated and set as centroid-vector for the category. This algorithm is also appropriate if number of categories is very large. Centroid algorithm computes similarity of test document with each centroid using cosine similarity measure. It assigns a document, class with whose centroid a document has greatest similarity.

Support Vector Machine (SVM): The main idea of SVM is to find a hyper-plane that best separates the documents and the margin, distance separating the border of subset and the nearest vector document, is large as possible. The nearest samples of the hyper-plane named support vectors are selected. The calculated hyper-plane permits to separate the space in two areas. To classify the new documents, calculate the area of the space and assign them the corresponding category.

B) Clustering Techniques

Clustering of documents is mainly used to minimize the amount of text by categorizing or grouping similar data items. This grouping is a common way for human processing information, and one of the good techniques for clustering helps to build different varieties which provide automated tools.

The following is a brief introduction to some of the clustering techniques:

a) **K-means Algorithm**

It is an algorithm to classify or to group your objects based on attributes/features into K number of group. K is a positive integer number. The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid. Thus, the purpose of K-mean clustering is to classify the data.

b) **Label Induction Grouping (LINGO) Algorithm**

Lingo algorithm is based on vector space model. First it extracts the user readable and frequent words/phrases from the input documents. Further by performing the Reduction of Original Term Document Matrix with Singular Value Decomposition (SVD) method to reduce the term document matrix, and then it find the labels of clusters and then assigns documents to that cluster labels based on the similarity value.

C) Ontology Based Classification

Traditional classification methods ignore relationship between words, they consider each term independent of each result. But, there exist a semantic relation between terms such as synonym, hyponymy etc. Therefore, for better classification results, there is need to understand the context of the text document. The ontology has different meaning for different users, in this classification task. Ontology stores words that are related to particular domain. Therefore, with the use of domain specific ontology, it becomes easy to classify the document even if the document does not contain the class name in it.

Example : Automatic Text Categorization of Marathi Language Documents

The designed system takes input as a set of Marathi Language text documents. These documents undergo preprocessing steps which include input validation, tokenization, stop word removal, stemming and morphological analysis. Then the features are extracted from preprocessed tokens. Then at last supervised learning methods and ontology based classification are applied to get output as classified Marathi documents as per class labels. The supervised learning methods considered here are Naïve Bayes (NB), Modified K-Nearest Neighbor (MKNN) and Support Vector Machine (SVM).

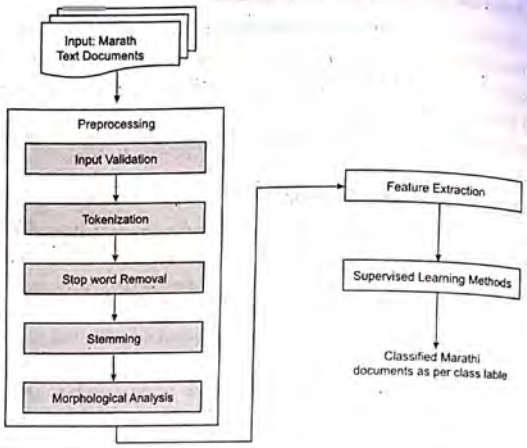


Figure 8: Text Categorization system for Marathi Documents

The input to the system would be a set of Marathi documents. The documents would be of different domains like Politics, Entertainment, Literature, Sports, History and Technology.

Sample Document 1:

आधीच ठरलेल्या इशाऱ्याप्रमाणे भेटीच्या वेळी तीन तोफांचे बार प्रतापगडावरून काढण्यात आले, आणि खानाच्या छावणीच्या जवळपासच्या झाडाझुडुपांमधे दडून बसलेल्या मावळ्यांनी हल्ला करून खानाच्या सैन्याची दाणादाण उडविली. खानाचा मुलगा फाजलखान आणि इतर काही सरदार लपूनछपून वाईच्या मुक्त छावणीपर्यंत आले. इथे खानाचा जनाना होता. ते पाठलागावर असलेल्या नेताजीच्या सैन्यापासून वाचण्यासाठी खजिना, हत्ती व इतर जड सामान टाकून विजापूरला जनान्यासकट पळाले.

शिवाजीराजांना जनतेत मिळालेला आदर आणि प्रेम अनेक शतकांनंतरही टिकून आहे त्यामागचे त्यांचे सहिष्णू वृत्ती हे फार महत्त्वाचे कारण आहे. अफझलखानाच्या मृत्यूनंतर त्यांनी त्याच्या शवाचे अंत्यसंस्कार इस्लामी पद्धतीने करून त्याची एक कबर प्रतापगडावर बांधली आणि त्या कबरीच्या कायम देखभालीची व्यवस्था केली.

अफझलखानच्या मृत्यूनंतर शिवाजीराजांनी दोरोजी नावाच्या सरदाराला कोकणपट्ट्यातील आणखी कित्ते आणि प्रदेश जिंकण्यास पाठवले. स्वतः राजे सातारा प्रांतात घुसून कोल्हापुरापर्यंत गेले व त्यांनी पन्हाळा जिंकून घेतला.

Sample Document 2:

तेंडुलकर हा नियमितपणे गोलंदाजी करत नसला तरी त्याने १३२ कसोटी सामन्यांमध्ये ३७ बळी आणि ३६५ एकदिवसीय सामन्यांमध्ये १४२ बळींची कामगिरी केली आहे. ज्यावेळेस महत्त्वाचे गोलंदाज अपयशी ठरत असतात त्यावेळेस सचिनला गोलंदाजी देण्यात येते. आणि बऱ्याच वेळेस तो बळी मिळविण्यात यशस्वी ठरतो. जरी त्याची गोलंदाजीची सरासरी ५० च्या वर असली, त्याला 'जम बसलेली फलंदाजांची जोडी फोडण्याचा हातगुण असणारा गोलंदाज' समजण्यात येते.(End of text)

Sample Document 3:

लता मंगेशकर (जन्म: सप्टेंबर २८, इ.स. १९२९) भारताच्या महान गायिका आहेत. त्या भारताच्या हिंदी चित्रपटसृष्टीच्या ख्यातनाम गायक-गायिकांपैकी एक आहेत. हिंदी संगीतविश्वात त्यांना 'लता दीदी' म्हणून ओळखले जाते. लता मंगेशकरांच्या कारकिर्दीची सुरूवात इ.स. १९४२मध्ये झाली आणि ती कारकीर्द सहा दशकांपेक्षा अधिक काळ टिकून आहे. त्यांनी ९८० पेक्षा अधिक हिंदी चित्रपटांची गाणी गायली असून, विसाहून अधिक प्रादेशिक भारतीय भाषांमध्ये (प्रामुख्याने मराठी) गायन केले आहे. लता मंगेशकरांचे कुटुंब संगीतासाठी प्रसिद्ध असून, सुप्रसिद्ध गायिका आशा भोसले, उषा मंगेशकर, मीना मंगेशकर आणि ख्यातनाम संगीतकार-गायक हृदयनाथ मंगेशकर ही त्यांची सख्खी भावंडे आहेत. लता मंगेशकरांचे वडील मास्टर दीनानाथ मंगेशकर हे मराठी नाट्य-संगीताचे प्रसिद्ध गायक होते.

भारताचा सर्वोच्च पुरस्कार 'भारतरत्न' प्राप्त होणाऱ्या गायक-गायिकांमध्ये लता दीदी या दुसऱ्या गायिका आहे.

The Output of text categorization system is as follows :

Categories in which the documents are classified:

Sample Document 1 : शिवाजीराजा

Sample Document 2 : गोलंदाजी

Sample Document 3 : गायिका

The tremendous development in the Information Technology has led to the availability of large number of documents on the Internet. Classifying such documents according to class wise or topic wise manually is time consuming and complex task. Also, managing and retrieval of such documents is a difficult task. To overcome these difficulties, automatic text classification systems can be used.

6. Text Summarization

Summarization means to reduce the size of the document without changing its meaning. It is one of the most researched areas among the Natural Language Processing (NLP) community. A good summary should cover the most vital information of the original document or a cluster of documents, while being coherent, non-redundant and grammatically readable. Automatic text summarization is the data science problem of creating a short, accurate, and fluent summary from a longer document. Summarization methods are greatly needed to consume the ever-growing amount of text data available online. In essence, summarization is meant to help us consume relevant information faster. Furthermore, applying text summarization reduces reading time, accelerates the process of researching for information, and increases the amount of information that can fit in an area.

Summarization techniques are categorized into extractive and abstractive techniques on the basis of whether the exact sentences are considered as they appear in the original text

Types of Text Summarization:

A. Extraction-based summarization:

The extractive text summarization technique involves pulling keyphrases from the source document and combining them to make a summary. The extraction is made according to the defined metric without making any changes to the texts.

Here is an example:

Source text: Joseph and Mary rode on a donkey to attend the annual event in Jerusalem. In the city, Mary gave birth to a child named Jesus.

Extractive summary: Joseph and Mary attend event Jerusalem. Mary birth Jesus.

As you can see above, the words in bold have been extracted and joined to create a summary — although sometimes the summary can be grammatically strange.

B. Abstraction-based summarization :

The abstraction technique entails paraphrasing and shortening parts of the source document. When abstraction is applied for text summarization in deep learning problems, it can overcome the grammar inconsistencies of the extractive method.

The abstractive text summarization algorithms create new phrases and sentences that relay the most useful information from the original text — just like humans do.

Therefore, abstraction performs better than extraction. However, the text summarization algorithms required to do abstraction are more difficult to develop; that's why the use of extraction is still popular.

Here is an example:

Abstractive summary: Joseph and Mary came to Jerusalem where Jesus was born.

Text summarization algorithm :

Usually, text summarization in NLP is treated as a supervised machine learning problem (where future outcomes are predicted based on provided data).

Typically, here is how using the extraction-based approach to summarize texts can work:

1. Introduce a method to extract the merited keyphrases from the source document. For example, you can use part-of-speech tagging, word sequences, or other linguistic patterns to identify the keyphrases.
2. Gather text documents with positively-labeled keyphrases. The keyphrases should be compatible to the stipulated extraction technique. To increase accuracy, you can also create negatively-labeled keyphrases.
3. Train a binary machine learning classifier to make the text summarization. Some of the features you can use include:
   - Length of the keyphrase
   - Frequency of the keyphrase
   - The most recurring word in the keyphrase
   - Number of characters in the keyphrase
4. Finally, in the test phrase, create all the keyphrase words and sentences and carry out classification for them.

Example : Abstractive text summarisation for MARATHI documents

The idea is to summarize an input Marathi document by creating semantic graph called rich semantic graph(RSG) for the original document, reducing the generated semantic graph, and further generating the final abstract summary.

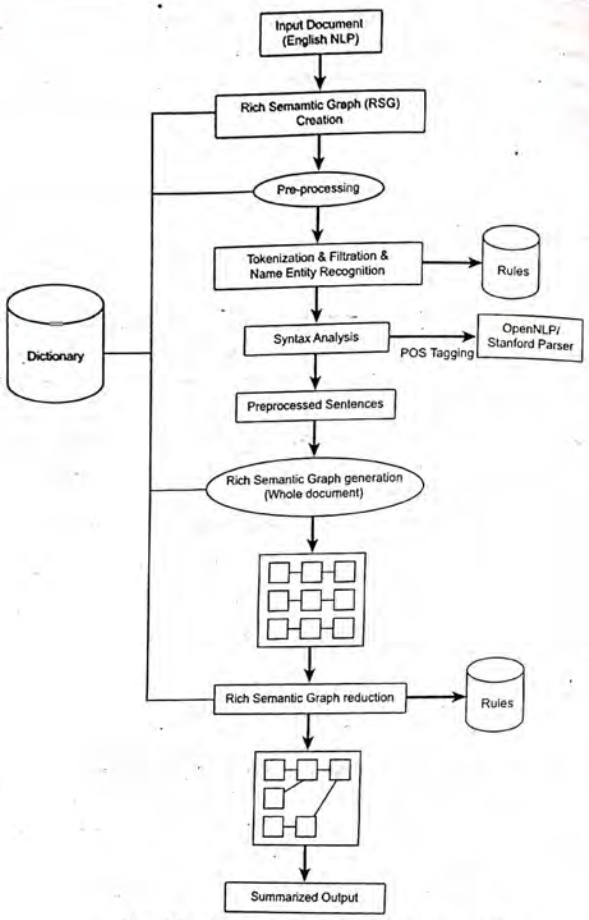


Figure 9: Text Summarization system for Marathi Documents

This approach consists of the following phases:

a. **Marathi text document as Input.**

b. **Rich Semantic Graph (RSG) creation pha:**

In Rich semantic graph creation step, analysis of the input text document is done, finds the sentence and produces tokens for the complete document. For every word it creates POS tags and detects the words into predefined categories such as person's name, location and organization. After this it generates the graph for every sentence and then it concatenates rich semantic sub-graphs. At last the sub-graphs are mixed together to show the complete document correctly.

c. **Rich Semantic Graph (RSG) reduction phase:**

Rich semantic graph reduction phase targets to reduce the obtained rich semantic graph of the source document to more reduced graph. Here a set of rules are applied on the obtained rich semantic graph to reduce it by merging, consolidating or deleting the graph nodes.

d. **Summary generation from reduced RSG:**

The summarized text generation phase targets to obtain the abstractive text summar from the reduced Rich Semantic Graph (RSG). To reach the target, this phase accesses the domain ontology; it has the data required in the same domain of RSG to obtain the final output. In addition, the Word Net ontology is used to obtain multiple texts according to the word synonyms. The obtained multiple texts are accessed and ranked, the most ranked text is considered.

Input : Single text document in Marathi Language.

मुरलीधर देविदास आमटे उर्फ बाबा आमटे हे एक थोर मराठी समाजसेवक होते.बाबा आमटेंच्या जन्म डिसेंबर २३ १९१४ रोजी महाराष्ट्रातील वर्धा जिल्ह्यात झाला.बाबा आमटेंना भारत सरकार कडून १९७१ मध्ये पद्मश्री पुरस्कार प्राप्त झाला आहे. तसेच बाबा आमटेंना भारत सरकार कडून १९८६ मध्ये पद्मविभूषण पुरस्कार प्राप्त झाला आहे. Baba Amte was a very good human being.

Output : Reduced Maeningful summary.

मुरलीधर देविदास आमटे उर्फ बाबा आमटे हे एक थोर मराठी समाजसेवक होते. त्यांचा जन्म डिसेंबर २३ १९१४ रोजी महाराष्ट्रातील वर्धा जिल्हात झाला. त्यांना भारत सरकार कडून १९७१ मध्ये पद्मश्री पुरस्कार आणि १९८६ मध्ये पद्मविभूषण पुरस्कार प्राप्त झाला आहे

7. Named Entity Recognition

The term "named entity", now widely used in Natural Language Processing, was first introduced for Sixth Message Understanding Conference (MUC-6) in 1995. During that time the MUC was focused on the information extraction were structured information of company and defence activities has been extracted from the unstructured text, from sources such as newspaper articles. During this task people felt a need for a system that can identify the names that includes names of persons, organisations, locations, and many different entities also numeric expressions which includes date, time, currency, numbers, percentages etc. For named entity all phrases in the text were supposed to be marked as person, location, organization, time or quantity. So identifying references to these entities in text was recognized as one of the important sub-tasks of IE and was called "Named Entity Recognition and Classification (NERC)". Natural languages are the languages which have naturally

Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a subtask of information extraction that seeks to locate and classify named entities in text. NER involves identification of proper names in texts, and classification into a set of predefined categories of interest.

For example: Mahatma Gandhi is very famous in India as "Bapu" or The full name of him is Mohandas Karamchand Gandhi. He was a great freedom fighter who led India as a leader of the nationalism against British rule. He was born on the 2nd of October in 1869 in Porbandar, Gujarat. He was a leader of Congress, as it was the only national party in India.

Tag Names Tagged Entities

Person	*Mahatma Gandhi, Bapu, Mohandas Karamchand Gandhi*
Location	*India, Porbandar, Gujarat*
Organisation	*Congress*
Date	*2nd, October, 1869*

There are many NER applications which include Information Extraction, Question Answering, Information Retrieval, Automatic Summarization, Machine Translation etc. The Named Entities can be known to us, if we perform computations on the natural language. The task of extracting and retrieving important information and details become easier and faster only if the Nes are already known. In defining IE tasks, people noticed

that it is essential to recognize information units such as names including person, organization, and location names, and numeric expressions including time, date, money, and percentages. Identifying references to these entities in text was acknowledged as one of IE's important sub-tasks and was called **"Named Entity Recognition (NER).**

NER Entity Classes (Tagset)

Named Entities (NEs) are noun phrases in natural language text. Natural language text is a sequence of sentences, where sentence in turn is a sequence of words and punctuation combined to add semantic to the text. Further, a word is a character sequence. In NER the aim is to distinguish between character sequence that represent noun phrases and character sequence that represents normal text. A proper noun in the text that is used to refer to a person, company, location etc. can be tagged as Named Entity (NE).

The main types of NEs are shown in fig below.

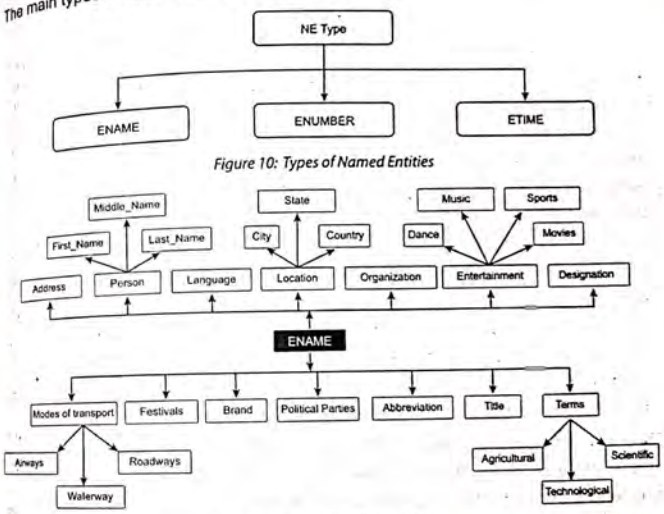


Figure 10: Types of Named Entities

Figure 11: Expansion of NAME Tagset

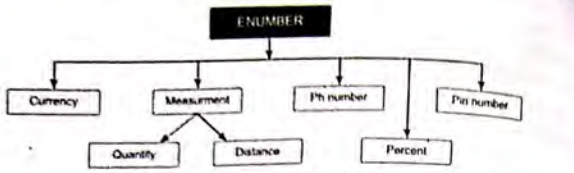


Figure 12: Expansion of NUMBER Tagset

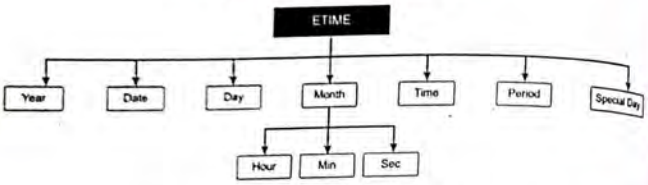


Figure 13: Expansion of TIME Tag set

Proper nouns play vital role in discovery of semantics hidden in the text. Identification of proper nouns in the raw text and classification of identified proper nouns in appropriate category is an important sub task of information extraction called Named Entity Recognition (NER). Any entity such as person, organization, geographical location, government agency, event that has a name is treated as a named entity. Expressions such as money, percentage, phone numbers, date, time, URLs, addresses are also treated as named entities even if they are not exactly like traditional proper nouns.

The NER system extracts name entity present in the given Marathi text. To extract name entity from the given text we are using two approaches in natural language processing. One of the approaches is machine learning where we are going to use HMM to predict the correct NER tag for the given word. The second approach is Rule Based system where handcrafted rules are written which helps is identifying the correct NER tag of the word.

The input to the system consists of Marathi text related to news domain which consist of text related to politics, sports, entertainment etc. Input text is processed to create an annotated NER data set by linguistic experts. In HMM based approach the input set is divided into two parts: training and testing.

In the training phase of HMM the annotated text is tokenized and the parameters for Viterbi algorithm like transition, emission etc are computed. In the testing phase correct

NER tag is predicted using Viterbi algorithm. In the Rule Based approach first the input text is tokenized and POS tagged. By using the NER tag set, correct NER tags are assigned to words in the sentence. The words which are left to be tagged are then tagged by using handcrafted rules. Finally the output of the system is NER tag text.

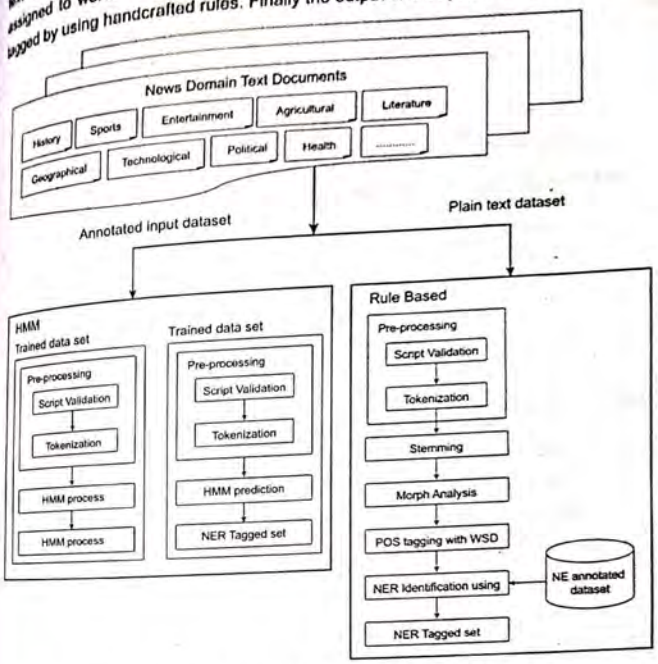


Figure 14: Named Entity Recognition Framework for Marathi Language

Supervised machine learning algorithm consists of train and test phase. In training phase prediction model is generated on the basis of various input parameters and state. In testing phase prediction model is used to predict the labels for the input. HMM is used as supervised machine learning algorithm to predict the NER tagged for Marathi text.

The input to the HMM module consists of annotated dataset which is further divided into test and train dataset. The training phase of HMM module gets the input as annotated dataset which is tokenized and then HMM algorithm is applied which results in creation of HMM model. In the testing phase the input can be the test dataset sentences or random user input which are tokenized and provided to HMM model which with the help of viterbi algorithm is used to find the correct tag for the user input.

Input: Marathi Plain Text (test data set).

1. सुबश्रीने आज ब्लँकेट विणण्याचे उपक्रम चालू केले.
2. ऑगस्ट पासून जानेवारी पर्यंत हे चालू होता.
3. सुबश्रीने जानेवारी पासून सुरू असलेला रेकॉर्ड आज पूर्ण केला.

Output: NER tagged Data.

1. <NER सुबश्रीने/Name> आज ब्लँकेट विणण्याचे उपक्रम चालू केले.
2. <NER ऑगस्ट/Month> पासून <NER जानेवारी/Month> पर्यंत हे चालू होता.
3. <NER सुबश्रीने/Name> <NER जानेवारी/Month> पासून सुरू असलेला रेकॉर्ड आज पूर्ण केला.

Expected Questions

Q. Write a note on :

1. Machine translation.
2. Information retrieval vs Information Extraction.
3. Question answering system.
4. Text Categorization.
5. Text Summarization.
6. sentiment analysis.
7. Named Entity Recognition.

Q. List various applications of NLP and discuss any 2 applications in detail.

Lab Manual NLP : Computational Lab II

COURSE OBJECTIVES:

1. To formulate the problems and solutions of NLP and establish their relation to linguistics and statistics.
2. To implement various language Models.
3. To design systems that uses NLP techniques
4. To train and evaluate empirical NLP systems.

COURSE OUTCOMES:

At the end of the course student should be able to:

1. Model linguistic phenomena with formal grammar.
2. Design, implement, and analyze NLP algorithms
3. Apply NLP techniques to design real world NLP applications, such as machine translation,
4. Implement text categorization, text summarization, information extraction...etc.
5. Implement proper experimental methodology for training and evaluating empirical NLP systems.

The objective of Natural Language Processing lab is to introduce the students with the basics of NLP which will empower them for developing advanced NLP tools and solving practical problems in this field.

List of experiments:

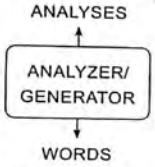
1. Word Analysis
2. Word generation
3. Stop word removal
4. Stemming
5. Morphology
6. POS Tagging
7. Chunking
8. N-gram language model

Experiment No 1:

TITLE: WORD ANALYSIS

THEORY :

A word can be simple or complex. For example, the word 'cat' is simple because one cannot further decompose the word into smaller parts. On the other hand, the word 'cats' is complex, because the word is made up of two parts: root 'cat' and plural suffix '-s'



Analysis of a word into root and affix(es) is called as Morphological analysis of a word. It is mandatory to identify the root of a word for any natural language processing task. A root word can have various forms. For example, the word 'play' in English has the following forms: 'play', 'plays', 'played' and 'playing'. Hindi shows more number of forms for the word **खेल** (khela) which is equivalent to 'play'. The forms of **खेल** (khela) are the following:

खेल (khela), **खेला** (khelaa), **खेली** (khelii), **खेलूंगा** (kheluungaa), **खेलूंगी** (kheluungii), **खेलेगा** (khelegaa), **खेलेगी** (khelegii), **खेलते** (khelate), **खेलती** (khelatii), **खेलने** (khelane), **खेलकर** (khelakar)

Thus we understand that the morphological richness of one language might vary from one language to another. Indian languages are generally morphologically rich languages and therefore morphological analysis of words becomes a very significant task for Indian languages.

Morphology is of two types:

1. **Inflectional morphology**
   Deals with word forms of a root, where there is no change in lexical category. For example, 'played' is an inflection of the root word 'play'. Here, both 'played' and 'play' are verbs.
2. **Derivational morphology**
   Deals with word forms of a root, where there is a change in the lexical category. For example, the word form 'happiness' is a derivation of the word 'happy'. Here, 'happiness' is a derived noun form of the adjective 'happy'.

Morphological Features:

All words will have their lexical category attested during morphological analysis. A noun and pronoun can take suffixes of the following features: gender, number, person, case For example, morphological analysis of a few words is given below:

LANGUAGE
INPUT:WORD
OUTPUT:ANALYSIS

Hindi
लड़के (ladake)
रा=लड़का(ladakaa), cat=n, gen=m, num=sg, case=obl

Hindi
लड़के (ladake)
रा=लड़का(ladakaa), cat=n, gen=m, num=pl, case=dir

Hindi
लड़कों (ladakoM)
रा=लड़का(ladakaa), cat=n, gen=m, num=pl, case=obl

English
boy
rt=boy, cat=n, gen=m, num=sg

English

boys

rt=boy, cat=n, gen=m, num=pl

A verb can take suffixes of the following features: tense, aspect, modality, gender, number, person

LANGUAGE

INPUT:WORD

OUTPUT:ANALYSIS

Hindi

हँसी(hansii)

rt=हँस(hans), cat=v, gen=fem, num=sg/pl, per=1/2/3 tense=past, aspect=pft

English

toys

rt=toy, cat=v, num=pl, per=3

'rt' stands for root.

'cat' stands for lexical category. The value of lexical category can be noun, verb, adjective, pronoun, adverb, preposition. 'gen' stands for gender. The value of gender can be masculine or feminine.

'num' stands for number. The value of number can be singular (sg) or plural (pl).

'per' stands for person. The value of person can be 1, 2 or 3

The value of tense can be present, past or future. This feature is applicable for verbs.

The value of aspect can be perfect (pft), continuous (cont) or habitual (hab). This feature is not applicable for verbs.

'case' can be direct or oblique. This feature is applicable for nouns. A case is an oblique case when a postposition occurs after noun. If no postposition can occur after noun, then the case is a direct case. This is applicable for hindi but not english as it doesn't have any postpositions. Some of the postpositions in hindi are: का(kaa), की(kii), के(ke), को(ko), में(meM)

Program :

```
wordListH = [{"word":"लड़का","rt":"लड़का","cat":"n","gen":"m","num":"sg","case":"obl","per":"","tense":
"","aspect":""},\
             {"word":"लड़के","rt":"लड़का","cat":"n","gen":"m","num":"pl","case":"dir","per":"","tense":
"","aspect":""},\
             {"word":"लड़कों","rt":"लड़का","cat":"n","gen":"m","num":"pl","case":"obl","per":"","tense":
"","aspect":""},\
             {"word":"हँसी","rt":"हँस","cat":"v","gen":"fem","num":"sg/pl","case":"obl","per":"1/2/3","t
ense":"past","aspect":"pft"},\
             {"word":"हँसी","rt":"हँसी","cat":"n","gen":"","num":"pl","case":"obl","per":"","tense":"",
"aspect":""}
             ]

wordListE = [{"word":"boy","rt":"boy","cat":"n","gen":"m","num":"sg","case":"","per":"","tense":"","as
pect":""},\
             {"word":"boys","rt":"boy","cat":"n","gen":"m","num":"pl","case":"","per":"","tense":"","as
pect":""},\
             {"word":"toys","rt":"toy","cat":"v","gen":"","num":"pl","case":"","per":"3","tense":"","as
pect":""}
             ]

'''
'rt' stands for root.
'cat' stands for lexical category. The value of lexical category can be noun, verb, adjective, pronou
n, adverb, preposition. 'gen' stands for gender. The value of gender can be masculine or feminine.
'num' stands for number. The value of number can be singular (sg) or plural (pl).
'per' stands for person. The value of person can be 1, 2 or 3
'tense' can be present, past or future. This feature is applicable for verbs.
'aspect' can be perfect (pft), continuous (cont) or habitual (hab).
'case' can be direct or oblique. This feature is applicable for nouns. A case is an oblique case when
a postposition occurs after noun. If no postposition can occur after noun, then the case is a direct
case. This is applicable for hindi but not english as it doesn't have any postpositions. Some of the
postpositions in hindi are: का(kaa), की(kii), के(ke), को(ko), में(meM)
'''

def processHindi(word):
    for x in wordListH:
        if word == x["word"]:
            if x['cat']=="v":
                print("Root of word is : ",x['rt']," category is verb ,","gender is ",x['gen']," , num
ber is ",x['num']," , tense is ",x['tense']," , person is ",x['per']," , aspect is ",x["aspect"] )
            else:
                print("Root of word is : ",x['rt']," category is noun ,","gender is ",x['gen']," , num
ber is ",x['num']," , case is : ",x['case'])

def processEnglish(word):
    for x in wordListE:
        if word == x["word"]:
            if x['cat']=="v":
                print("Root of word is : ",x['rt']," category is verb ,","gender is ",x['gen']," , num
ber is ",x['num']," , tense is ",x['tense']," , person is ",x['per']," , aspect is ",x["aspect"] )
            else:
                print("Root of word is : ",x['rt']," category is noun ,","gender is ",x['gen']," , num
ber is ",x['num'])

wordLangugae = input("Langugae H for hindi E for English : ")
print("You selected :",wordLangugae)
```

```
Langugae H for hindi E for English : E
You selected : E
```

```
word = input("Enter the word : ")
print("Your word is",word)
```

```
Enter the word : toys
Your word is toys
```

```
if wordLangugae == "H":
    processHindi(word)
else:
    processEnglish(word)
```

```
Root of word is :  toy  category is verb , gender is    , number is  pl  , tense is    , person is  3
, aspect is
```

Experiment No 2:

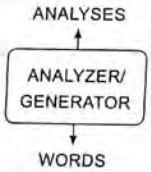
TITLE: WORD GENERATION

THEORY

A word can be simple or complex. For example, the word 'cat' is simple because one cannot further decompose the word into smaller parts. On the other hand, the word 'cats' is complex, because the word is made up of two parts: root 'cat' and plural suffix '-s'

Morphological analysis and generation: Inverse processes.

- Analysis may involve non-determinism, since more than one analysis is possible.
- Generation is a deterministic process. In case a language allows spelling variation, then till that extent, generation would also involve non-determinism.



Language	Input: Analysis	Output: Word
Hindi	rt=**लड़का**(ladakaa), cat=n, gen=m, num=sg, case=obl	**लड़के**(ladake)
Hindi	rt=**लड़का**(ladakaa), cat=n, gen=m, num=pl, case=dir	**लड़के**(ladake)
English	rt=boy, cat=n, num=pl	boys
English	rt=play, cat=v, num=sg, per=3, tense=pr	plays

Experiment No 3:

TITLE: STOP WORD REMOVAL

THEORY

In computing, stop words are words which are filtered out before or after processing of natural language data (text). Through "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.

Any group of words can be chosen as the stop words for a given purpose. For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on. In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as "The Who", "The The", or "Take That". Other search engines remove some of the most common words—including lexical words, such as "want"—from a query in order to improve performance.

A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

Stop word removal is one of the most commonly used pre-processing steps across different NLP applications. The idea is simply removing the words that occur commonly across all the documents in the corpus. Typically, articles and pronouns are generally classified as stop words. These words have no significance in some of the NLP tasks like information retrieval and classification, which means these words are not very discriminative.

On the contrary, in some NLP applications stop word removal will have very little impact. Most of the time, the stop word list for the given language is a well hand-curated list of words that occur most commonly across corpuses.

While the stop word lists for most languages are available online, these are also ways to automatically generate the stop word list for the given corpus. A very simple way to build a stop word list is based on word's document frequency (Number of documents the word presents), where the words present across the corpus can be treated as stop words. Enough research has been done to get the optimum list of stop words for some specific corpus.

Program

```
import nltk
nltk.download('stopwords') # run it only once
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /home/nbuser/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
example_sent = "This is a sample sentence, showing off the stop words filtration."

stop_words = set(stopwords.words('english'))

print("stop words: ",stop_words)
```

```
stop words:  {'some', 'no', "you're", 'is', "don't", 'which', 'being', "you'd", 'both', "wouldn't", 'a
ny', 'with', 'its', "weren't", "needn't", "that'll", 'i', 'myself', "you'll", 'of', 'yourself', 'as',
'herself', 'won', 'her', 'again', "hasn't", 'there', 'if', 'other', "haven't", 'will', "isn't", 'at',
'ours', 'when', 'ourselves', 'yours', 'each', 'most', 'doesn', 'than', 'she', 'in', "didn't", 'my',
e', 'for', 'should', 'you', 'over', 'wasn', 'few', 'about', 'are', "mustn't", 'had', 'does', 'becaus
'didn', 'who', 'by', 've', 'shouldn', 'between', 'out', 'once', 'the', 'only', 'him', 'were', 'those',
p', 'just', 'm', 'll', 'haven', 'their', 'here', 'that', 'very', 'during', 'a', 'how', 'your', 'be', 'u
'isn', "mightn't", "aren't", 'hadn', 'what', 'off', 'theirs', 'while', 'below', 'own', "she's", 'd',
'having', 'do', 'so', 'hers', "shouldn't", 'shan', 'don', 'hasn', 'it', 'this', 'at', 'our', 'has', 't
o', 'can', 'further', 'why', 'ain', 'needn', 'through', 'more', 'on', 'now', 'they', 'such', 're', 'bu
t', 'his', 'same', "hadn't", 'we', 'too', "you've", 'them', 'have', 'did', "should've", 'before', 'fro
m', 'and', 'all', 'into', 't', 'ma', 'mightn', 'couldn', 'yourselves', 'against', "it's", 'then', 'abo
ve', 'or', "couldn't", "wasn't", 'themselves', "won't", 'himself', 'whom', 'under', "mustn't", 'o', 'are
n', 'an', 'not', 'doing', 'where', 's', 'he', 'wouldn', 'nor', 'am', "doesn't", 'itself', 'until',
'y', 'after', 'me', 'been', "shan't", 'weren', 'down', 'these'}
```

```
word_tokens = word_tokenize(example_sent)

filtered_sentence = [w for w in word_tokens if not w in stop_words]

filtered_sentence = []
stop_words_list_ofsentence = []
```

```
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
    else:
        stop_words_list_ofsentence.append(w)

print("token list of sentence",word_tokens)
print("Stop word list of sentence",stop_words_list_ofsentence)
print("token list of sentence with stop words removed",filtered_sentence)
```

```
token list of sentence ['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop',
'words', 'filtration', '.']
Stop word list of sentence ['is', 'a', 'off', 'the', 'is', 'a', 'off', 'the']
token list of sentence with stop words removed ['This', 'This', 'sample', 'sentence', ',', 'showing',
'stop', 'words', 'filtration', '.', 'This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'fi
ltration', '.']
```


Experiment No 4:

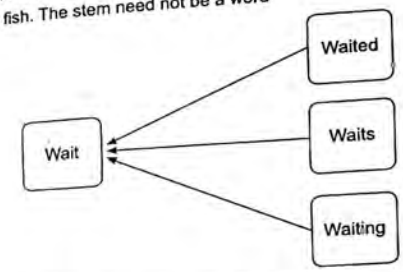
TITLE: STEMMING

THEORY

The idea of stemming is a sort of normalizing method. Many variations of words carry the same meaning, other than when tense is involved.

In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not 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.

A stemmer for English operating on the stem cat should identify such strings as cats, catlike, and catty. A stemming algorithm might also reduce the words fishing, fished, and fisher to the stem fish. The stem need not be a word



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 words with 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

However, the two words differ in their flavour. 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.

The reason why we stem is to shorten the lookup, and normalize sentences.

Consider:

I was taking a ride in the car.

I was riding in the car.

This sentence means the same thing. in the car is the same. I was is the same. the ing denotes a clear past-tense in both cases, so is it truly necessary to differentiate between ride and riding, in the case of just trying to figure out the meaning of what this past-tense activity was? No, not really. This is just one minor example, but imagine every word in the English language, every possible tense and affix you can put on a word. Having individual dictionary entries per version would be highly redundant and inefficient, especially since, once we convert to numbers, the "value" is going to be identical.

One of the most popular stemming algorithms is the Porter stemmer, which has been around since 1979. It is the most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective . 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:

(F)	Rule		Example	
	SSES	→ SS	caresses	→ Caress
	IEF	→ 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:

Will map replacement to replac, but not cement to c.

Program

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /home/nbuser/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()

example_words = ["python","pythoner","pythoning","pythoned","pythonly"]

for w in example_words:
    print(ps.stem(w))
```

```
python
python
python
python
pythonli
```

```
# how let's try stemming a typical sentence, rather than some words:

new_text = "It is important to by very pythonly while you are pythoning with python. All pythoners hav
e pythoned poorly at least once."

words = word_tokenize(new_text)

for w in words:
    print(ps.stem(w))
```

```
It
is
import
to
by
veri
pythonli
while
you
are
python
with
python
.
all
python
have
python
poorli
at
least
onc
.
```

```
def stemSentence (new_text):
    words = word_tokenize(new_text)

    for w in words:
        print(ps.stem(w))
```

```
sent = "hello man what is dogy"
```

```
stemSentence(sent)
```

```
hello
man
what
is
dogi
```


Experiment No 5

TITLE: MORPHOLOGY

THEORY

Morphology is the study of the way words are built up from smaller meaning bearing units i.e., morphemes. A morpheme is the smallest meaningful linguistic unit. Morphemes are considered as smallest meaningful units of language. These morphemes can either be a root word(play) or affix(-ed). Combination of these morphemes is called morphological process. So, word "played" is made out of 2 morphemes "play" and "-ed". Thus finding all parts of a word(morphemes) and thus describing properties of a word is called "Morphological Analysis". For example, "played" has information verb "play" and "past tense", so given word is past tense form of verb "play"

For eg:

बच्चों(bachchoM) consists of two morphemes, बच्चा(bachchaa) has the information of the root word noun "बच्चा"(bachchaa) and ओं(oM) has the information of plural and oblique case.

played has two morphemes play and -ed having information verb "play" and "past tense", so given word is past tense form of verb "play".

बच्चा	- ओं
लड़का	- ओं
play	- ed
want	- ed

Words can be analysed morphologically if we know all variants of a given root word. We can use an 'Add-Delete' table for this analysis.

बच्चा – आ → बच्च + ओं → बच्चों

Delete Add

Analysis of a word :

बच्चों (bachchoM) = बच्चा(bachchaa)(root) + ओं(oM)(suffix)

(ओं=3 plural oblique)

A linguistic paradigm is the complete set of variants of a given lexeme. These variants can be classified according to shared inflectional categories (eg: number, case etc) and arranged into tables.

Paradigm for बच्चा

CASE/NUM	singular	plural
direct	**बच्चा** (bachchaa)	**बच्चे** (bachche)
oblique	**बच्चे** (bachche)	**बच्चों** (bachchoM)

Algorithm to get **बच्चों**(bachchoM) from **बच्चा**(bachchaa)

1. Take Root **बच्च**(bachch)**आ**(aa)
2. Delete **आ**(aa)
3. output **बच्च**(bachch)
4. Add **ओं**(oM) to output
5. Return **बच्चों** (bachchoM)
6. Therefore **आ** is deleted and **ओं** is added to get **बच्चों**
7. Add-Delete table for **बच्चा**

DELETE	ADD	NUMBER	CASE	VARIANTS
आ(aa)	आ(aa)	sing	dr	बच्चा(bachchaa)
आ(aa)	ए(e)	plu	dr	बच्चे(bachche)
आ(aa)	ए(e)	sing	ob	बच्चे(bachche)
आ(aa)	ओं(oM)	plu	ob	बच्चों(bachchoM)

Experiment No 6

TITLE: POS TAGGING

THEORY

POS tagging or part-of-speech tagging is the procedure of assigning a grammatical category like noun, verb, adjective etc. to a word. In this process both the lexical information and context play an important role as the same lexical form can behave differently in a different context.

part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context—i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.

Part-of-speech tagging is harder than just having a list of words and their parts of speech, because some words can represent more than one part of speech at different times, and because some parts of speech are complex or unspoken. This is not rare—in natural languages (as opposed to many artificial languages), a large percentage of word-forms are ambiguous. For example, even "dogs", which is usually thought of as just a plural noun, can also be a verb:

The sailor dogs the hatch.

Correct grammatical tagging will reflect that "dogs" is here used as a verb, not as the more common plural noun. Grammatical context is one way to determine this; semantic analysis can also be used to infer that "sailor" and "hatch" implicate "dogs" as 1) in the nautical context and 2) an action applied to the object "hatch" (in this context, "dogs" is a nautical term meaning "fastens (a watertight door) securely").

Types of POS taggers

POS-tagging algorithms fall into two distinctive groups:

1. Rule-Based POS Taggers
2. Stochastic POS Taggers

E. Brill's tagger, one of the first and most widely used English POS-taggers, employs rule-based algorithms. Let us first look at a very brief overview of what rule-based tagging is all about.

Rule-Based Tagging

Automatic part of speech tagging is an area of natural language processing where statistical techniques have been more successful than rule-based methods.

Typical rule-based approaches use contextual information to assign tags to unknown or ambiguous words. Disambiguation is done by analyzing the linguistic features of the word, its preceding word, its following word, and other aspects.

For example, if the preceding word is an article, then the word in question must be a noun. This information is coded in the form of rules.

Example of a rule:

If an ambiguous/unknown word X is preceded by a determiner and followed by a noun, tag it as an adjective.

Defining a set of rules manually is an extremely cumbersome process and is not scalable at all. So we need some automatic way of doing this.

The Brill's tagger is a rule-based tagger that goes through the training data and finds out the set of tagging rules that best define the data and minimize POS tagging errors. The most important point to note here about Brill's tagger is that the rules are not hand-crafted, but are instead found out using the corpus provided. The only feature engineering required is a set of rule templates that the model can use to come up with new features.

Stochastic Part-of-Speech Tagging

The term 'stochastic tagger' can refer to any number of different approaches to the problem of POS tagging. Any model which somehow incorporates frequency or probability may be properly labelled stochastic.

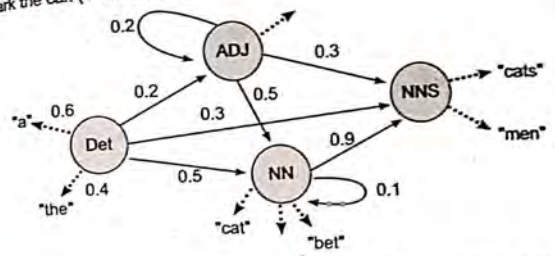
The simplest stochastic taggers disambiguate words based solely on the probability that a word occurs with a particular tag. In other words, the tag encountered most frequently in the training set with the word is the one assigned to an ambiguous instance of that word. The problem with this approach is that while it may yield a valid tag for a given word, it can also yield inadmissible sequences of tags.

An alternative to the word frequency approach is to calculate the probability of a given sequence of tags occurring. This is sometimes referred to as the n-gram approach, referring to the fact that the best tag for a given word is determined by the probability that it occurs with the n previous tags. This approach makes much more sense than the one defined before, because it considers the tags for individual words based on context.

The next level of complexity that can be introduced into a stochastic tagger combines the previous two approaches, using both tag sequence probabilities and word frequency measurements. This is known as the Hidden Markov Model (HMM).

For example the word "Park" can have two different lexical categories based on the context.

1. The boy is playing in the park. ('Park' is Noun)
2. Park the car. ('Park' is Verb)



Assigning part of speech to words by hand is a common exercise one can find in an elementary grammar class. But here we wish to build an automated tool which can assign the appropriate part-of-speech tag to the words of a given sentence. One can think of creating handcrafted rules by observing patterns in the language, but this would limit the system's performance to the quality and number of patterns identified by the rule crafter. Thus, this approach is not practically adopted for building POS Tagger. Instead, a large corpus annotated with correct POS tags for each word is given to the computer and algorithms then learn the patterns automatically from the data and store them in the form of a trained model. Later this model can be used to POS tag new sentences.

POS tag list

```
CC      coordinating conjunction
CD      cardinal digit
DT      determiner
EX      existential there (like: "there is" ... think of it like "there exists")
FW      foreign word
IN      preposition/subordinating conjunction
JJ      adjective       'big'
JJR     adjective, comparative  'bigger'
JJS     adjective, superlative  'biggest'
LS      list marker     1)
MD      modal   could, will
NN      noun, singular 'desk'
NNS     noun plural     'desks'
NNP     proper noun, singular   'Harrison'
NNPS    proper noun, plural     'Americans'
PDT     predeterminer   'all the kids'
POS     possessive ending       parent\'s
PRP     personal pronoun        I, he, she
PRP$    possessive pronoun      my, his, hers
RB      adverb  very, silently,
RBR     adverb, comparative     better
RBS     adverb, superlative     best
RP      particle        give up
TO      to      go 'to' the store.
UH      interjection    errrrrrrrm
VB      verb, base form take
VBD     verb, past tense        took
VBG     verb, gerund/present participle taking
VBN     verb, past participle   taken
VBP     verb, sing. present, non-3d     take
VBZ     verb, 3rd person sing. present  takes
WDT     wh-determiner   which
WP      wh-pronoun      who, what
WP$     possessive wh-pronoun   whose
WRB     wh-abverb       where, when
```

Program

```
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
import nltk

nltk.download('averaged_perceptron_tagger')

s = "This is a simple sentence"
tokens = word_tokenize(s)
tokens_pos = pos_tag(tokens)

print(tokens_pos)
```



```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/rbuser/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('simple', 'JJ'), ('sentence', 'NN')]
```

```
def postaggsentence(text):
    sent_text = sent_tokenize(text) # this gives us a list of sentences
    # now loop over each sentence and tokenize it separately
    for sentence in sent_text:
        tokenized_text = word_tokenize(sentence)
        tagged = pos_tag(tokenized_text)
        print("POS tagged :",tagged)
```

```
text = "this is a test sentence . we are tesing dog was tagged in wall"
```

```
postaggsentence(text)
```

```
POS tagged : [('this', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('test', 'NN'), ('sentence', 'NN'), ('.',
'.')]
POS tagged : [('we', 'PRP'), ('are', 'VBP'), ('tesing', 'VBG'), ('dog', 'NN'), ('was', 'VBD'), ('tagge
d', 'VBN'), ('in', 'IN'), ('wall', 'NN')]
```

```
def postaggedword(word):
    print (pos_tag([word]))
```

```
word = "this"
```

```
postaggedword(word)
```

```
[('this', 'DT')]
```

Experiment No 7

TITLE: CHUNKING

THEORY

Chunking is an analysis of a sentence which identifies the constituents (noun groups, verbs, verb groups, etc.) which are correlated. These are non-overlapping regions of text. Usually, each chunk contains a head, with the possible addition of some function words and modifiers either before or after depending on languages. These are non-recursive in nature i.e. a chunk cannot contain another chunk of the same category.

Some of the groups possible are:

1. Noun Group
2. Verb Group

For example, the sentence 'He reckons the current account deficit will narrow to only 1.8 billion in September.' can be divided as follows:

[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only 1.8 billion] [PP in] [NP September]

Each chunk has an open boundary and close boundary that delimit the word groups as a minimal non-recursive unit.

Chunking of text involves dividing a text into syntactically correlated words. For example, the sentence 'He ate an apple ' can be divided as follows:

[NP He] [VP ate] [NP an apple]

Each chunk has an open boundary and close boundary that delimit the word groups as a minimal non-recursive unit. This can be formally expressed by using IOB prefixes.

[B-NP He] [B-VP ate] [B-NP an] [I-NP apple]

NP VP NP

Chunking of text involves dividing a text into syntactically correlated words.

Eg. He ate an apple to satiate his hunger.

[NP He] [VP ate] [NP an apple] [VP to satiate] [NP his hunger]

Eg: दरवाज़ा खुल गया

[NP दरवाज़ा] [VP खुल गया]

Chunk Types

The chunk types are based on the syntactic category part. Besides the head a chunk also contains modifiers (like determiners, adjectives, postpositions in NPs).

The basic types of chunks in English are:

Chunk Type	Tag Name
1. Noun	NP
2. Verb	VP
3. Adverb	ADVP
4. Adjectival	ADJP
5. Prepositional	PP

the basic Chunk Tag Set for Indian Languages

Sl. No	Chunk Type	Tag Name
1	Noun Chunk	NP
2.1	Finite Verb Chunk	VGF
2.2	Non-finite Verb Chunk	VGNF
2.3	Verb Chunk (Gerund)	VGNN
3	Adjectival Chunk	JJP
4	Adverb Chunk	RBP

A. NP Noun Chunks

Noun Chunks will be given the tag NP and include non-recursive noun phrases and postposition for Indian languages and preposition for English. Determiners, adjectives and other modifiers will be part of the noun chunk.

Eg

(इस/DEM किताब/NN में/PSP)NP

'this' 'book' 'in'

((in/IN the/DT big/ADJ room/NN))NP

B. Verb Chunks

The verb chunks are marked as VP for English, however they would be of several types for Indian languages. A verb group will include the main verb and its auxiliaries, if any.

For English:

I (will/MD be/VB loved/VBD)VP

The types of verb chunks and their tags are described below.

1. VGF Finite Verb Chunk

The auxiliaries in the verb group mark the finiteness of the verb at the chunk level. Thus, any verb group which is finite will be tagged as VGF. For example,

Eg: मैंने घर पर (खाया/VM)VGF

'I erg''home' 'at''meal' 'ate'

2. VGNF Non-finite Verb Chunk

A non-finite verb chunk will be tagged as VGNF.

Eg: सेब (खाता/VM हुआ/VAUX)VGNF लड़का जा रहा है

'apple' 'eating' 'PROG' 'boy' go' 'PROG' 'is'

3. VGNN Gerunds

A verb chunk having a gerund will be annotated as VGNN.

Eg: शराब (पीना/VM)VGNN सेहत के लिए हानिकारक है sharAba

'liquor' 'drinking' 'heath' 'for' 'harmful' 'is'

C. JJP/ADJP Adjectival Chunk

An adjectival chunk will be tagged as ADJP for English and JJP for Indian languages. This chunk will consist of all adjectival chunks including the predicative adjectives.

Eg:

यह लड़की है (सुन्दर/JJ)JJP

The fruit is (ripe/JJ)ADJP

Note: Adjectives appearing before a noun will be grouped together within the noun chunk.

D. RBP/ADVP Adverb Chunk

This chunk will include all pure adverbial phrases.

Eg:

वह (धीरे-धीरे/RB)RBP चल रहा था

'he' 'slowly' 'walk' 'PROG' 'was'

He walks (slowly/ADV)/ADVP

PP Prepositional Chunk

This chunk type is present for only English and not for Indian languages. It consists of only the preposition and not the NP argument.

Eg:

(with/IN)PP a pen

IOB prefixes

Each chunk has an open boundary and close boundary that delimit the word groups as a minimal non-recursive unit. This can be formally expressed by using IOB prefixes: B-CHUNK for the first word of the chunk and I-CHUNK for each other word in the chunk. Here is an example of the file format:

Tokens	POS	Chunk-Tags
He	PRP	B-NP
ate	VBD	B-VP
an	DT	B-NP
apple	NN	I-NP
to	TO	B-VP
satiate	VB	I-VP
his	PRP$	B-NP
hunger	NN	I-NP

Pogram

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /home/nbuser/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /home/nbuser/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/nbuser/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.

True
```

```
from nltk.tokenize import PunktSentenceTokenizer

da = open("albert.txt",'r')
train_text = da.read()

sample_text = train_text
```

```
print(train_text)
```

```
custom_sent_tokenizer = PunktSentenceTokenizer(train_text)

tokenized = custom_sent_tokenizer.tokenize(sample_text)

print(tokenized)
```

```
['\ufeff\nAlbert Einstein (14 March 1879 - 18 April 1955) was a German-born theoretical physicist who developed the general th
ory of relativity, one of the two pillars of modern physics (alongside quantum mechanics).', '[2]\nHe received the Nobel Prize
in Physics in 1921, but not for relativity.', '[3] His theories of special and general relativity are of great importance to ma
ny branches of physics and astronomy.', 'They have been given experimental confirmation by many experiments and observations.',
'Einstein is well known for his theories about light, matter, gravity, space, and time.', 'His most well known equation is {\d
isplaystyle E=mc^{2}} E=mc^{2}.', 'It means that energy and mass are different forms of the same thing.', 'Einstein published
ore than 300 scientific papers and over 150 non-scientific works.', 'He received honorary doctorate degrees in science, medicine
e and philosophy from many European and American universities.', 'Near the beginning of World War II, he warned President Franklin
lin D. Roosevelt that Germany might be developing an atomic weapon, and recommended that the U.S. begin nuclear weapons research
h.', '[4] That research, begun by a newly established Manhattan Project, resulted in the U.S. becoming the first and only count
ry to have nuclear weapons during the war.', 'On praise of his genius, Einstein humbly stated, "I am not a genius, I am just to
rious.', 'I ask many questions.', 'and when the answer is simple, then God is answering.""]
```

```
def process_content():
    try:
        for i in tokenized:
            print(i)
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            chunkGram = r"""NLP: {<VB>?<.*>*?<NNP>}"""
            chunkParser = nltk.RegexpParser(chunkGram)
            chunked = chunkParser.parse(tagged)

            print(chunked)
            for subtree in chunked.subtrees(filter=lambda t: t.label() == 'Chunk'):
                print(subtree)

            #chunked.draw() wont work in notebook

    except Exception as e:
        print(str(e))

process_content()
```

Albert Einstein (14 March 1879 – 18 April 1955) was a German-born theoretical physicist who developed the general theory of relativity, one of the two pillars of modern physics (alongside quantum mechanics).

```
(S
  /NN
  Albert/NNP
  Einstein/NNP
  (/(
  14/CD
  March/NNP
  1879/CD
  –/NNP
  18/CD
  April/NNP
  1955/CD
  )/)
  was/VBD
  a/DT
  German-born/JJ
  theoretical/JJ
  physicist/NN
  who/WP
  developed/VBD
  the/DT
  general/JJ
  theory/NN
  of/IN
  relativity/NN
  ,/,
  one/CD
  of/IN
  the/DT
```

two/CD
pillars/NNS
of/IN
modern/JJ
physics/NNS
(/(
alongside/IN
quantum/NN
mechanics/NNS
)/)
./.)

[2]
He received the Nobel Prize in Physics in 1921, but not for relativity.
(S
[/RB
2/CD
]/NN
He/PRP
received/VBD
the/DT
Nobel/NNP
Prize/NNP
in/IN
Physics/NNPS
in/IN
1921/CD
,/,
but/CC
not/RB
for/IN
relativity/NN
./.)

[3] His theories of special and general relativity are of great importance to many branches of physics and astronomy.
(S
[/RB
3/CD
]/VBD
His/PRP$
theories/NNS
of/IN
special/JJ
and/CC
general/JJ
relativity/NN
are/VBP
of/IN
great/JJ
importance/NN
to/TO
many/JJ
branches/NNS
of/IN
physics/NNS
and/CC
astronomy/NN
./.)

They have been given experimental confirmation by many experiments and observations.
(S
They/PRP
have/VBP
been/VBN
given/VBN
experimental/JJ

confirmation/NN
by/IN
many/JJ
experiments/NNS
and/CC
observations/NNS
./.)
Einstein is well known for his theories about light, matter, gravity, space, and time.
(S
Einstein/NNP
is/VBZ
well/RB
known/VBN
for/IN
his/PRP$
theories/NNS
about/IN
light/NN
,/,
matter/NN
,/,
gravity/NN
,/,
space/NN
,/,
and/CC
time/NN
./.)
His most well known equation is {\displaystyle E=mc^{2}} E=mc^{2}.
(S
His/PRP$
most/JJS

well/RB
known/VBN
equation/NN
is/VBZ
{/(
\displaystyle/JJ
E=mc^/NNP
{/(
2/CD
}/)
}/)
E=mc^/NNP
{/(
2/CD
}/)
./.)
It means that energy and mass are different forms of the same thing.
(S
It/PRP
means/VBZ
that/IN
energy/NN
and/CC
mass/NN
are/VBP
different/JJ
forms/NNS
of/IN
the/DT
same/JJ
thing/NN
./.)
Einstein published more than 300 scientific papers and over 150 non-scientific

works.
(S
Einstein/NNP
published/VBD
more/JJR
than/IN
300/CD
scientific/JJ
papers/NNS
and/CC
over/IN
150/CD
non-scientific/JJ
works/NNS
./.)
He received honorary doctorate degrees in science, medicine and philosophy from many European and American universities.
(S
He/PRP
received/VBD
honorary/JJ
doctorate/NN
degrees/NNS
in/IN
science/NN
,/,
medicine/NN
and/CC
philosophy/NN
from/IN
many/JJ
European/JJ
and/CC

American/JJ
universities/NNS
./.)
Near the beginning of World War II, he warned President Franklin D. Roosevelt that Germany might be developing an atomic weapon, and recommended that the U.S. begin nuclear weapons research.
(S
Near/IN
the/DT
beginning/NN
of/IN
World/NNP
War/NNP
II/NNP
,/,
he/PRP
warned/VBD
President/NNP
Franklin/NNP
D./NNP
Roosevelt/NNP
that/IN
Germany/NNP
might/MD
(NLP
be/VB
developing/VBG
an/DT
atomic/JJ
weapon/NN
,/,
and/CC
recommended/VBD

that/IN
the/DT
U.S./NNP)
begin/JJ
nuclear/JJ
weapons/NNS
research/NN
./.)

[4] That research, begun by a newly established Manhattan Project, resulted in the U.S. becoming the first and only country to have nuclear weapons during the war.

(S
[/RB
4/CD
]/NNS
That/IN
research/NN
,/,
begun/VBN
by/IN
a/DT
newly/RB
established/VBN
Manhattan/NNP
Project/NNP
,/,
resulted/VBD
in/IN
the/DT
U.S./NNP
becoming/VBG
the/DT
first/JJ

and/CC
only/RB
country/NN
to/TO
have/VB
nuclear/JJ
weapons/NNS
during/IN
the/DT
war/NN
./.)

On praise of his genius, Einstein humbly stated, "I am not a genius, I am just curious.

(S
On/IN
praise/NN
of/IN
his/PRP$
genius/NN
,/,
Einstein/NNP
humbly/RB
stated/VBD
,/,
"/FW
I/PRP
am/VBP
not/RB
a/DT
genius/NN
,/,
I/PRP
am/VBP

```
just/RB
curious/JJ
./.)
I ask many questions.
(S I/PRP ask/VBP many/JJ questions/NNS ./.)
and when the answer is simple, then God is answering."
(S
  and/CC
  when/WRB
  the/DT
  answer/NN
  is/VBZ
  simple/JJ
  ,/,
  then/RB
  God/NNP
  is/VBZ
  answering/VBG
  ./.
  "/NN)
```

```
data = 'The little yellow dog will then walk to the Starbucks, where he will introduce them to Michael.'
data_tok = nltk.word_tokenize(data) #tokenisation
data_pos = nltk.pos_tag(data_tok) #POS tagging
```

```
cfg_1 = "CUSTOMCHUNK: {<VB>*<.*>*<NNP>}" #should return 'walk to the Starbucks', etc.
chunker = nltk.RegexpParser(cfg_1)
data_chunked = chunker.parse(data_pos)
```

```
data_chunked
```

```
Tree('S', [('The', 'DT'), ('little', 'JJ'), ('yellow', 'JJ'), ('dog', 'NN'), ('will', 'MD'), ('then', 'RB'), Tree('CUSTOMCHUNK', [('walk', 'VB'), ('to', 'TO'), ('the', 'DT'), ('Starbucks', 'NNP')]), (',', ','), ('where', 'WRB'), ('he', 'PRP'), ('will', 'MD'), Tree('CUSTOMCHUNK', [('introduce', 'VB'), ('them', 'PRP'), ('to', 'TO'), ('Michael', 'NNP')]), ('.', '.')])
```


Experiment No 8

TITLE: N-GRAM Language Model

THEORY

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus.

Using Latin numerical prefixes, an n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". English cardinal numbers are sometimes used, e.g., "four-gram", "five-gram", and so on.

An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a (n − 1)–order Markov model.[2] n-gram models are now widely used in probability, communication theory, computational linguistics (for instance, statistical natural language processing), computational biology (for instance, biological sequence analysis), and data compression. Two benefits of n-gram models (and algorithms that use them) are simplicity and scalability – with larger n, a model can store more context with a well-understood space–time trade-off, enabling small experiments to scale up efficiently.

N-grams of texts are extensively used in text mining and natural language processing tasks. They are basically a set of co-occurring words within a given window and when computing the n-grams you typically move one word forward (although you can move X words forward in more advanced scenarios). For example, for the sentence "The cow jumps over the moon". If N=2 (known as bigrams), then the N-grams would be:

the cow
cow jumps
jumps over
over the
the moon

So you have 5 n-grams in this case. Notice that we moved from the->cow to cow->jumps to jumps->over, etc, essentially moving one word forward to generate the next bigram.

If N=3, the n-grams would be:

the cow jumps

cow jumps over

jumps over the

over the moon

So you have 4 n-grams in this case. When N=1, this is referred to as unigrams and this is essentially the individual words in a sentence. When N=2, this is called bigrams and when N=3 this is called trigrams. When N>3 this is usually referred to as four grams or five grams and so on.

How many N-grams in a sentence?

If X=Num of words in a given sentence K, the number of n-grams for sentence K would be:

$N\,grams_K = X - (N - 1)$

What are N-grams used for?

N-grams are used for a variety of different task. For example, when developing a language model, n-grams are used to develop not just unigram models but also bigram and trigram models. Google and Microsoft have developed web scale n-gram models that can be used in a variety of tasks such as spelling correction, word breaking and text summarization.

Program

```
import re
from nltk.util import ngrams
s = "Natural-language processing (NLP) is an area of computer science " \
    "and artificial intelligence concerned with the interactions " \
    "between computers and human (natural) languages."
s = s.lower()
s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)
tokens = [token for token in s.split(" ") if token != ""]
output = list(ngrams(tokens, 3))
```

```
print (output)
```

```
[('natural', 'language', 'processing'), ('language', 'processing', 'nlp'), ('processing', 'nlp', 'i
s'), ('nlp', 'is', 'an'), ('is', 'an', 'area'), ('an', 'area', 'of'), ('area', 'of', 'computer'), ('o
f', 'computer', 'science'), ('computer', 'science', 'and'), ('science', 'and', 'artificial'), ('and',
'artificial', 'intelligence'), ('artificial', 'intelligence', 'concerned'), ('intelligence', 'concerne
d', 'with'), ('concerned', 'with', 'the'), ('with', 'the', 'interactions'), ('the', 'interactions', 'b
etween'), ('interactions', 'between', 'computers'), ('between', 'computers', 'and'), ('computers', 'an
d', 'human'), ('and', 'human', 'natural'), ('human', 'natural', 'languages')]
```



```
output = list(ngrams(tokens, 3))
print(output)
```

```
[('my', 'dog', 'also'), ('dog', 'also', 'likes'), ('also', 'likes', 'eating'), ('likes', 'eating', 'so
usage.')]
```

```
output = list(ngrams(tokens, 2))
print(output)
```

```
[('natural', 'language'), ('language', 'processing'), ('processing', 'nlp'), ('nlp', 'is'), ('is', 'a
n'), ('an', 'area'), ('area', 'of'), ('of', 'computer'), ('computer', 'science'), ('science', 'and'),
('and', 'artificial'), ('artificial', 'intelligence'), ('intelligence', 'concerned'), ('concerned', 'w
ith'), ('with', 'the'), ('the', 'interactions'), ('interactions', 'between'), ('between', 'computer
s'), ('computers', 'and'), ('and', 'human'), ('human', 'natural'), ('natural', 'languages')]
```