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), and whether 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.	PRP$	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:

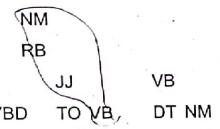
1. Assign all possible tags to words
2. Remove tags according to set of rules of type:
3. if word+1 is an adj, adv, or quantifier and the following is a sentence boundary
4. And
5. word-1 is not a verb like "consider"
6. then
7. eliminate non-adv else eliminate adv.
8. Typically more than 1000 hand-written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

First Stage:

FOR each word

Get all possible parts of speech using a morphological analysis algorithm

Example



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	PREDETERMİNER 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:

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

$$t_i = \arg\max_j P(t_{j-1}, w_i) \quad \ldots(2)$$

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:

$$t_i = \arg\max_j P(t_j \mid t_{j-1}) P(w_i \mid t_j) \quad \ldots(3)$$

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 jti) 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 thus 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

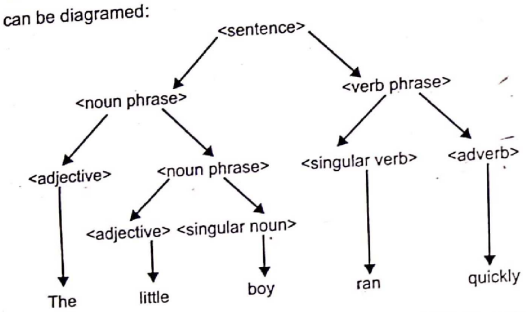
Here, the arrow, →, might be read as "is defined as" and the vertical bar, "|", as "or". Thus, a noun phrase is defined as an adjective followed by another noun phrase or as an adjective followed by a singular noun. This definition of noun phrase is recursive because noun phrase occurs on both sides 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 the 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

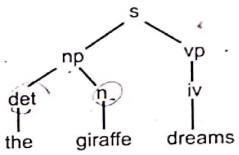
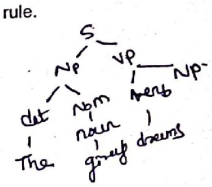


Figure 2: A grammar and a parse tree for "the giraffe dreams"

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

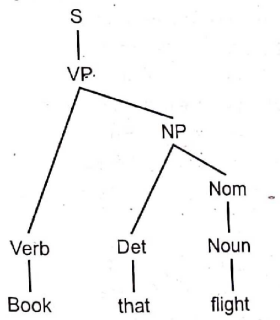


Figure 3. The correct parse tree for the sentence Book that flight according to the grammar in Figure 2.

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

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.

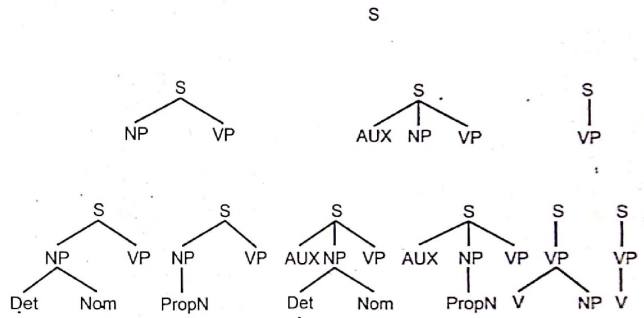


Figure 4: An expanding top-down search space. Each ply is created by taking each tree from the previous ply, replacing the leftmost non-terminal with each of its possible expansions, and collecting each of these trees into a new ply.

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.

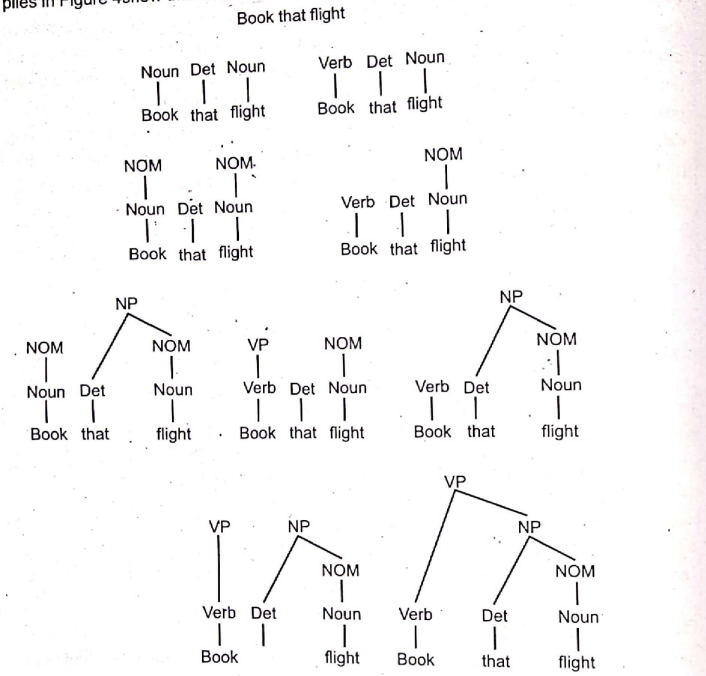


Figure 5: An expanding bottom-up search space for the sentence Book that flight. This figure does not show the final tier of the search with the correct parse tree Figure 1.

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:

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

$P(w_4 = Rainy \mid w_3 = Foggy, w_2 = Sunny, w_1 = sunny)$ ..(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 $\{w_1, w_2 \ldots . w_n\}$:

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

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

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

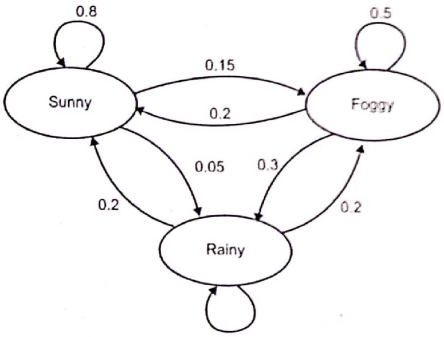
So this now has a profound aect on the number of histories that we have to nd statistics for we now only need $3^2 = 9$ numbers to characterize the probabilities of all of the sequences This assumption may or may not be a valid assumption depending on the situation(in the case of weather it's probably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

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

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

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

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

The probability $P(u_1,\ldots..u_n|w_1,\ldots..w_n)$ can be estimated as $\Pi_{i=1}^{n} P(u_i \mid w_i)$, 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.

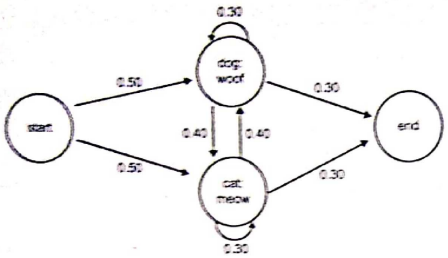


Figure 6: A finite state transition network representing a Markov model

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.

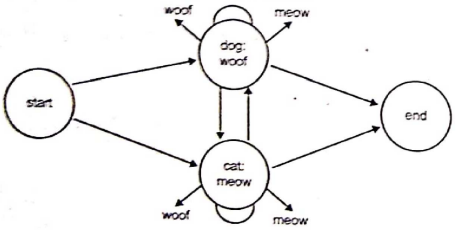


Figure 7: A finite state transition network representing an HMM

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:

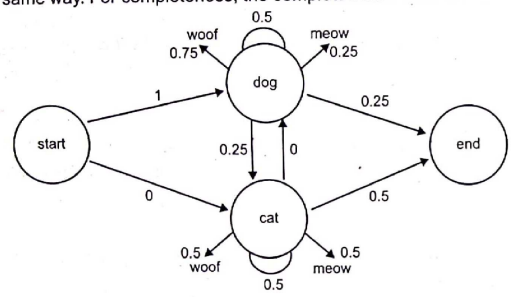


Figure 8: Finite state transition network of the hidden Markov model of our example.

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:

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

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

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

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

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

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

$$T \approx \underset{\bar{T}}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_1 \mid \hat{t}_1) P(\hat{t}_i \mid \hat{t}_{i-1})$$

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

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

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

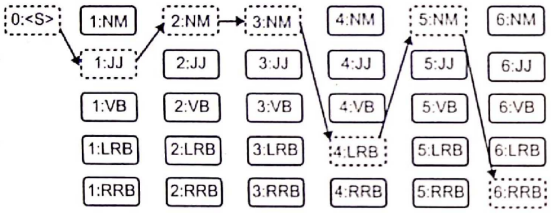
Hence if we were to draw a finite state transition network for this HMM, the observed states would be the tags and the words would be the emitted states similar to our woof and meow example. We have already seen that we can use the maximum likelihood estimates to calculate these probabilities.

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 count 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	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

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 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

  1

  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 li−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:

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

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

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

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 labelling's 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.