NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) allows machines to break down and interpret human language.
NLP analyzes the grammatical structure of sentences and the individual meaning of words, then uses algorithms to extract meaning and deliver outputs.

Applications:

- Chatbots
- Language Translators
- Text recommendation (Search engines and emails)
- Voice assistants
- Email spam filtering
- Grammar correction
- Sentiment Analysis

Terms:

Automation: Anything that decreases human effort and labour work
Example: zip, fan, refrigerator
AI: Anything that takes decisions just like a human brain. Example: DSA (does not need ML)
AI can be automation, but doesn't always need to be
ML: Technique to enhance AI, using the learning aspect of brain (automating AI processes?)
NLP: Human language interpretation, requires ML

Components of NLP:

Two main components:

1. Natural Language Understanding: allows machines to understand and analyze human language by extracting the metadata from the content. (entities, keywords, emotions, semantics etc.). It maps the input to a useful representation.
2. Natural Language Generation: translates the computerized data to natural language representation, involves text planning, sentence planning and text realization.

Phases of NLP:

1. **Lexical Analysis (or Morphological Analysis):** scanning the input as a stream of characters and converting them into meaningful units called lexicons or morphemes. It divides the whole text into paragraphs, sentences, and words.
   Lexicons: list of all the words and phrases used in a particular language or subject
   Morpheme: The smallest unit of a word. It is a word or a part of a word that has a purpose (meaning). (dog or -s in dogs, both have some meaning or purpose for their presence)

There are two types of morphemes-free morphemes and bound morphemes. "Free morphemes" can stand alone with a specific meaning, for example, eat, date, weak. "Bound morphemes" cannot stand alone with meaning. Example: 'dis', '-s'

Morphemes are composed of two separate classes called (a) bases (or roots) and (b) affixes.

A "base," or "root" is a morpheme in a word that gives the word its principal meaning. An example of a "free base" morpheme is woman in the word womanly. An example of a "bound base" morpheme is -sent in the word dissent. (Because adding dis changes the meaning of the word totally, hence sent is bound but base)
An "affix" is a bound morpheme that occurs before or after a base. An affix that comes before a base is called a "prefix." Some examples of prefixes are ante-, pre-, un-, and dis-
An affix that comes after a base is called a "suffix." Some examples of suffixes are -ly, -er, -ism, and -ness

Step 1: Sentence Segmentation: splits sentences within a text

Step 2: Word Tokenization: break up a string of words into semantically useful units called tokens. (split at every blank space except some collocations like New York)

Step 3: Stemming: Stemming is used to normalize words into its base form or root form. The big problem with stemming is that sometimes it produces the root word which may not have any meaning.

Search engines use stemming for indexing the words. That's why rather than storing all forms of a word, a search engine can store only the stems.

Step 4: Lemmatization: similar to stemming, but produces a root word which has a meaning, because it is dictionary based and chooses lemma based on the context. Better -> good (stemming keeps it unchanged)

Step 5: Removing Stop Words: filtering out high-frequency words that add little or no semantic value to a sentence, for example, which, to, at, for, is, etc.

Let's say you want to classify customer service tickets based on their topics. In this example: "Hello, I'm having trouble logging in with my new password", it may be useful to remove stop words like "hello", "I", "am", "with", "my", so you're left with the words that help you understand the topic of the ticket: "trouble", "logging in", "new", "password".

2. **Syntactic Analysis**
   Syntactic Analysis is used to check grammar, word arrangements, and shows the relationship among the words.
   Step 6: POS Tagging: Tagging is a method to assign each word in a sentence to its corresponding Part of Speech Tag. Helps in parsing and understanding relations. (Often PoS Tagging and Parsing (dependency) are done together)
   Step 7: Parsing: Parsing is the process of identifying the syntactic structure of a text and determining the relationships, represented in the form of a parse tree.
   Parsing refers to the formal analysis of a sentence by a computer into its constituents, which results in a parse tree showing their syntactic relation to one another, which can be used for further processing and understanding.

A parse tree is a tree that highlights the syntactical structure of a sentence according to a formal grammar, for example by exposing the relationships between words or sub-phrases. Depending on which type of grammar we use, the resulting tree will have different features.

Some Parsing Techniques:

1. Constituency Parsing: Process of analyzing grammatical structure of sentences by breaking down into sub-phrases known as constituents.

2. Dependency Parsing: Process of analyzing grammatical structure of sentences by extracting dependencies between the words. (between head words and words which modify those heads)

3. **Semantic Analysis**

   It focuses on identifying the meaning of the language. Firstly, each word is searched for its dictionary meaning in the database. Secondly, all the meanings of each different word are integrated to find a proper correlation between the word structures. (associating each word with the context) This process of determining the correct meaning is called lexical disambiguation.

   Step 8: NER: Finding spans of text that constitute proper nouns and designating the type of entity to them.

   Step 9: Relation Extraction: takes the named entities and tries to find the semantic relations between them.

4. **Discourse Integration**

   It deals with the effect of a previous sentence on the sentence in consideration. In the text, “Jack is a bright student. He spends most of the time in the library.” Here, discourse assigns “he” to refer to “Jack”.

5. **Pragmatic Analysis**

   Process of extraction of insights from the text. Depends on the previous sentences. Focuses on the effects of context. Given a sentence, “Turn off the lights” is an order or request to switch off the lights, depends upon previous sentence (context)

   Pragmatics is the study of meaning of how the sentence is spoken, and discourse is related to the context in which that speech and writing occurs. Pragmatics is specifically concerned with how speakers' shared interests and purposes shape discourse.

   Pragmatics has more to do with the external or physical context, while discourse analysis focuses on the linguistic context.

Why is NLP Hard? (Classical Problems in NLP)

- **Contextual words/phrases:** Same word, different context based meanings
  I ran to the store because we ran out of milk.
- **Homonyms:** Same pronunciation, different words(meanings): their and there
- **Synonyms:** Different words, same meaning
- **Irony and Sarcasm**
- **Domain specific Language**
- **Errors in text and speech:** Spoken: mispronunciations, stutters, accents; Written: grammar and spellings
- **Slang and colloquialism:** open vocabulary; Informal phrases, expressions, idioms, and culture-specific lingo (no dictionary meaning)
- **Ambiguity:** ability of being understood in more than one way
  - **Lexical:** a word can be a noun, verb etc.
  - **Syntactic:** sentences are parsed in different ways.
    Example: The man saw the girl with a telescope
  - **Semantic:** When the meaning of the words themselves can be misinterpreted. (ambiguous phrase or word present in a sentence)
    Example: The car hit the pole while it was moving.

- **Anaphoric (Discourse):** sometimes anaphoric entities cause ambiguity.
  Example:The horse ran up the hill. It was steep. It soon got tired.
- **Pragmatic:** when the context of a phrase gives it multiple interpretations, and the statement is not specific.
  Example: I like you too

UNIT-2

Classical approaches to NLP

1. **Rule based:**
   - oldest approach, tried and true and works well.
   - Tends to focus on pattern matching and parsing, can be thought of as 'fill in the blanks' method
   - Is low precision and high recall (Hence good performance in specific cases (when recall is more important than precision or beta >1))
   - Example: CFG and Regular Expression

2. **Traditional ML based (Probabilistic):**
   - Include probabilistic modeling, likelihood maximization and linear classifiers
   - Involves
     - Training data: a corpus with markup
     - Feature Eng: word type, surrounding words, capitalized, plural etc.
     - Training the model
     - inference (applying model to test data) characterized by finding most probable words, next word, best category, etc.

3. **Neural Networks based:**
   Similar to traditional ML, with a few differences:
   - Feature engineering is generally skipped, as networks will "learn" important features (this is generally one of the claimed big benefits of using neural networks for NLP)
   - Instead, streams of raw parameters (words) are fed into the NN

   Example: RNN and CNN

Data Driven Approach

Focus on data rather than intuition.

Linguistics

Linguistics is the scientific study of language, including its grammar, semantics and phonetics.
Classical linguistics involves devising and evaluating rules of language.
Great progress was made on the 'formal methods' of syntax and semantics, but for the most part, natural language poses resistance for mathematical formulation.

TEXT CLASSIFICATION EVALUATION

Human defined labels (in labelled training data) are called gold labels.
Confusion matrix is a table for visualizing how an algorithm performs with respect to gold labels.

Evaluation Metrics:

Precision: tp/(tp+fp) (correctly identified positives in all determined positives)

Recall: tp/(tp+fn) (correctly identified positives in all actual positives)
Precision and Recall find TP: the things we are looking for, unlike accuracy
Accuracy is an awful metric considering unbalanced datasets
F-measure: incorporating precision and recall into single metric (beta is the weightage)
(beta>1: recall is favoured, else precision)

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} \quad \text{or} \left(\text{with } \beta^2 = \frac{1-\alpha}{\alpha}\right) \quad F = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

Harmonic mean is used because it is a conservative metric; the harmonic mean of two values is closer to the minimum of the two values than the arithmetic mean is. Thus it weighs the lower of the two numbers more heavily.
More than 2 classes:

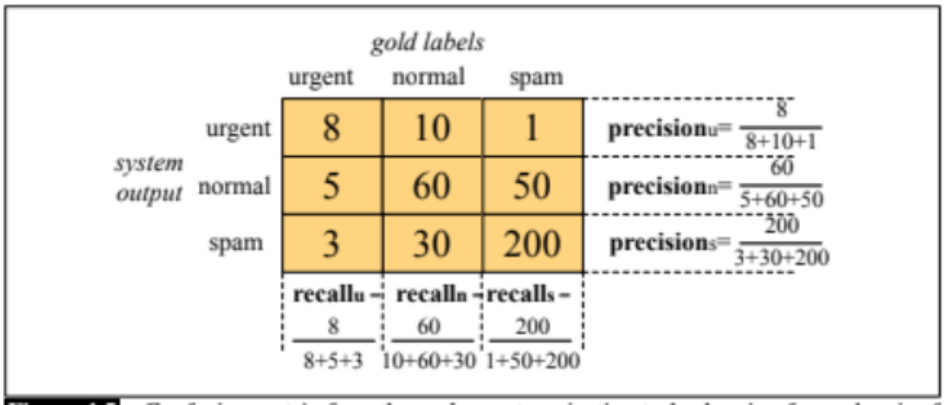


Figure 4.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2

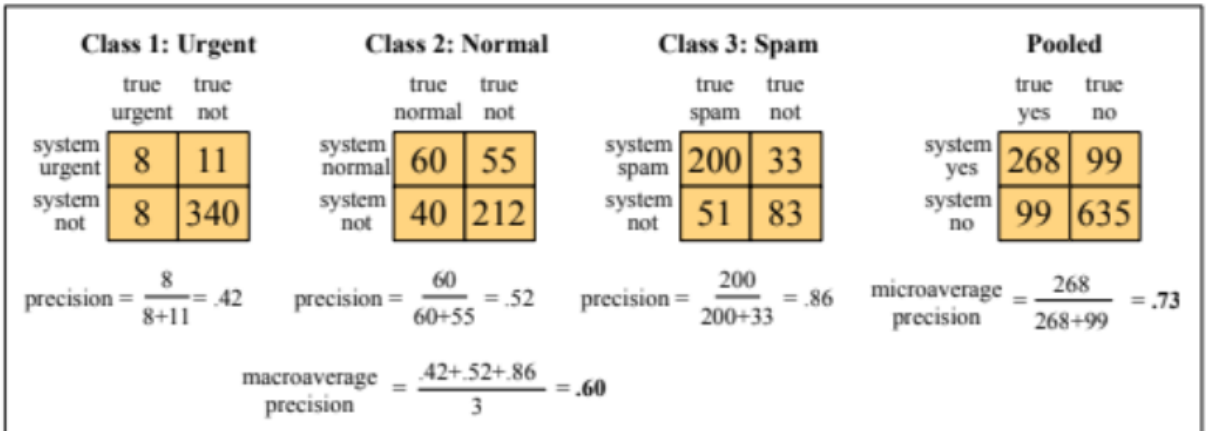


Figure 4.6 Separate confusion matrices for the 3 classes from the previous figure, showing the pooled confusion matrix and the microaveraged and macroaveraged precision.

In order to derive a single metric that tells us how well the system is doing, we can combine these values in two ways. In macroaveraging, we compute the performance for each class, and then average over classes. In microaveraging, we collect the decisions for all classes into a single confusion matrix, and then compute precision and recall from that table.

RELATION EXTRACTION

Relation extraction algorithms:
5 main classes:

1. **Using patterns:** They are hand-built patterns like hyponyms (Y such as X, X and other Y, Y including X etc), located-in(ORG, LOCATION).
   There can be many relations between two entities e.g. person (founded/invested/visited) organization.
   We combine the use of named entities and specific patterns to find the relations.
   Example:
   PERSON, POSITION of ORG
   PER (named|appointed|chose|etc.) PER Prep? POSITION
   Truman appointed Marshall Secretary of State
   (Pattern to answer "Who holds what office in which organization?")

   Hand-built patterns have the advantage of high-precision and they can be tailored to specific domains.
   On the other hand, they are often low-recall, and it's a lot of work to create them for all possible patterns. We need higher accuracy too.

2. **Supervised Learning**

1. Choose a set of relations we would like to extract
2. Choose a set of named entities between which we want to find the relation.
3. Find and label data:
   Choose a representative corpus, label the named entities in the corpus, hand-label the relations between them; and break into training, dev and test sets
4. Train a classifier on the training set

These steps are somewhat modified to improve efficiency:

1. Find all pairs of named entities
2. Run a classifier whose job is to classify each pair as Yes/No for being related
3. If Yes, run a second classifier to classify the relation.

The first classifier eliminates most pairs.

How to classify the relation now?

- **Feature Based**: feature-based classifier (like logistic regression or random forests), considers word features, named entity features and syntactic struct.
  Example: for the sentence:

American Airlines, a unit of AMR, immediately matched the move, spokesman **Tim Wagner** said

word features (M1 and M2 are the two named entities)

- The headwords of M1 and M2 and their concatenation
  Airlines Wagner Airlines-Wagner
- Bag-of-words and bigrams in M1 and M2
  American, Airlines, Tim, Wagner, American Airlines, Tim Wagner
- Words or bigrams in particular positions
  M2: -1 spokesman
  M2: +1 said
- Bag of words or bigrams between M1 and M2:
  a, AMR, of, immediately, matched, move, spokesman, the, unit

named entity features

- Named-entity types and their concatenation
  (M1: ORG, M2: PER, M1M2: ORG-PER)

syntactic structure

- Dependency-tree paths
  $Airlines \leftarrow_{subj} matched \leftarrow_{comp} said \rightarrow_{subj} Wagner$

- **Neural Networks Based**

Advantage:
Supervised Models can get high accuracies with enough hand-labeled data.

Drawback:
Labeling a large training set is extremely expensive and supervised models are brittle: they don't generalize well to different text genres.

3. **Semi-supervised learning via bootstrapping**

When we don't have a large training data but few seed tuples or few high-precision patterns. We use bootstrapping to use the seeds directly.

1. Gather a set of seed pairs that have relation R
2. Iterate: (Bootstrapping)
   - find sentences with these pairs
   - look at the context between or around the pair and generalize the context to create patterns (replace the constants with variables to create patterns)
   - Use the patterns to find more pairs

This algorithm is called the depri algorithm.

Snowball algorithm, a similar iterative algorithm is there which requires the two related entities to be named entities. In depri, they could be a string of words.
And then it computes confidence for each pattern.

Another semi-supervised algorithm extends both these ideas, called distant supervision.

4. **Distant supervision**

It combines bootstrapping with supervised learning. Instead of few seeds, it uses a large database to get a huge number of seeds. And instead of iterating, lots of features are created from these so many examples constructing a supervised classifier.

Similarity with supervised:

- Lots of features are used in the classifier
- Detailed hand-created database
- Doesn't require iteratively expanding patterns

Similarity with unsupervised:

- Uses very large amounts of unlabeled data (described below)
- Not sensitive to genre issues (unlike supervised)

Working:

1. For a relation to be found, find the tuples associated with that relation in the big database (Example: relation is ‘born in’)
2. Find sentences in unlabeled large corpus with those entities
3. Extract frequent features/patterns (‘took birth’, ‘born’ etc.)
4. Train supervised classifier using these thousands of patterns

5. **Unsupervised**

Open information extraction

1. Use parsed data to decide if two entities can be related or not (trustworthy or not)
2. In a single pass, extract all the relations
3. Rank these relations based on redundancy (if it is occurring a lot of times, it is a real relation)

In semi-supervised and unsupervised,
Because we don’t have any gold labels, we can’t find precision and recall and hence evaluate our model. We can approximate the precision, or calculate precision at different levels of recall. (But there’s no way to evaluate recall, no whole/universal set of test data because)

LANGUAGE MODELING

Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence.
(probability distribution over words)
Language models analyze bodies of text data to provide a basis for their word predictions.
Applications: those which generally produce text as output. Machine translation, PoS tagging, speech and handwriting recognition. (Given in N-gram models)

Types of Language Models:

1. **Statistical Language Models:** Probabilistic models that are able to predict the next word in the sequence, given the preceding words. These models use traditional statistical techniques like N-grams, Hidden Markov Models (HMM) and certain linguistic rules to learn the probability distribution of words

2. **Neural Language Models:** Based on Neural Networks, considered an advanced approach, overcomes the limitations of the statistical approach

N-GRAM LANGUAGE MODELLING

Predicting the future: difficult!
But predicting the next few words someone is going to say?: easier

We will be introducing models that assign a probability to each possible next word, as well as to an entire sentence. (What is the probability of this sentence to appear in any text? (that depends how much sense it makes, order of words etc.))

Why would you want to predict upcoming words, or assign probabilities to sentences?

- Probabilities are essential in any task in which we have to identify words in noisy, ambiguous input, like speech recognition. For a speech recognizer to realize that you

said ***I will be back soonish*** and not *I will* ***be bassoon dish***, it helps to know that ***back soonish*** is a much more probable sequence than ***bassoon dish***.

- For writing tools like spelling correction or grammatical error correction.
  Example:
  Their are two midterms (mistyped): The phrase ***There are*** is much more probable,
  Everything has improve (mistyped): ***has improved*** probable than ***has improve***.

- It is also important in language translations, because we can map each word to the other word in the desired language, but their order may need to be changed according to the desired language. Hence, a probabilistic model of word sequences helps here.

- Probabilities are also important for augmentative and alternative communication systems. People often use such AAC devices if they are physically unable to speak or sign but can instead use eye gaze or other specific movements to select words from a menu to be spoken by the system. Word prediction can be used to suggest likely words for the menu.

Models that assign probabilities to sequences of words are called language models or LMs. The simplest one is the n-gram model. n-gram is a sequence of n words, we will use them to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire sequences.

N-Grams

1. How to calculate p(w|h) (probability of word given history)?
   One way to estimate this probability is from relative frequency counts: take a very large corpus and find p(w following h) / p(h)
   (Problem is that it's not necessary that we find exact h in even a huge corpus, because language is creative and any particular context might have never occurred before!)

2. Similarly, how to calculate the probability of an entire sentence (joint probability of an entire sequence of words)?
   Again, one way is by relative frequency counts: p(that sequence)/p(all permutations of those words)
   (Problem is that's a lot to estimate)

For this reason, we'll need to introduce more clever ways of estimating the probability of a word w given a history h, or the probability of an entire word sequence W.

Better way of calculating probabilities of entire sequences:

$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n-1})$

$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. But using the chain rule doesn't really seem to help us! We don't know any way to compute the exact probability of a word given a long sequence of preceding words. (p(w|h))

Now, the intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

The bigram model, for example, approximates the probability of a word given all the previous words P(wn|w1:n−1) by using only the conditional probability of the preceding word P(wn|wn−1).

The assumption that the probability of a word depends only on the previous word is called a **Markov assumption**. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past. We can generalize the bigram (which looks one word into the past) to trigram (which looks two words into the past) and thus to the n-gram (which looks n−1 words into the past).

Thus, the general equation for this n-gram approximation to the conditional probability of the next word in a sequence is

P(wn|w1:n−1) ≈ P(wn|wn−N+1:n−1)

Hence, probability of an entire sequence becomes: (FOR BIGRAM)

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

An intuitive way to estimate these conditional probabilities is the maximum likelihood estimation or MLE. (The relative frequency count method that we were using)

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \quad \Rightarrow \quad P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

C is the count of bigram/unigram

Let's work through an example using a mini-corpus of three sentences. We'll first need to augment each sentence with a special symbol <s> at the beginning of the sentence, to give us the bigram context of the first word. We'll also need a special end-symbol. </s>

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

Here are the calculations for some of the bigram probabilities from this corpus

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

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

What kinds of linguistic phenomena are captured in these bigram statistics?
Some of the bigram probabilities above encode some facts that we think of as strictly syntactic in nature, like the fact that what comes after ***eat*** is usually a noun or an adjective, or that what comes after ***to*** is usually a verb. Others might be a fact about the personal assistant task, like the high probability of sentences beginning with the words I. And some might even be cultural rather than linguistic, like the higher probability that people are looking for Chinese versus English food.

Some practical issues:

- Trigram /4-gram /5-gram are more common to use, given that we have enough training data. Accordingly, we need to add dummy tokens (Ex: <s><s> for trigram)
- As we are multiplying multiple probabilities (which are less than 1), it can lead to numerical underflow. Hence we always represent and compute language model probabilities in log format as log probabilities. (we add the logs in that case then) (To convert back, simply raise them to e)

Evaluating Language Models

The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called extrinsic evaluation. Extrinsic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand. Thus, for speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription.

Unfortunately, running big NLP systems end-to-end is often very expensive. Instead, it would be nice to have a metric that can be used to quickly evaluate potential improvements in a language model. An intrinsic evaluation metric is one that measures the quality of a model independent of any application. For this we need a test set. The probabilities of an n-gram model come from the corpus it is trained on (called training set)

Test sets and other datasets that are not in our training sets are called held out corpora because we hold them out from the training data.

Whichever model assigns a higher probability to the test set—meaning it more accurately predicts the test set—is a better model.

There should not be training on the test set. Training on the test set introduces a bias that makes the probabilities all look too high, and causes huge inaccuracies in perplexity.

Sometimes we use a particular test set so often that we implicitly tune to its characteristics. We then need a fresh test set that is truly unseen. In such cases, we call the initial test set the development test set or, devset. (This is actually the validation set)

In practice, we often just divide our data into 80% training, 10% development, and 10% test.

An evaluation metric: Perplexity
In practice we don't use raw probability as our metric for evaluating language models, but a variant called perplexity. (inverse probability of the test set, normalized by the number of words)

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned} \qquad \text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

Minimizing perplexity is equivalent to maximizing the test set probability according to the language model.

We also need to include the end-of-sentence marker </s> (but not the beginning-of-sentence marker <s>) in the total count of word tokens N.

Another way to think about perplexity: weighted average branching factor of a language. The branching factor of a language is the number of possible next words that can follow any word.

The perplexity of two language models is only comparable if they use identical vocabularies.

FACTORS AND PROBLEMS TO TAKE CARE OF

- The n-gram model, like many statistical models, is dependent on the training corpus. Hence the probabilities often encode specific facts about a given training corpus.
- Also, n-grams do a better job of modelling the training corpus as we increase the value of N.

Hence,

- We should be sure to use a training corpus that has a similar genre to whatever task we are trying to accomplish.
- It is equally important to get training data in the appropriate dialect or variety, especially when processing social media posts or spoken transcripts. (using den for then, for example)
- Handling the problem of sparsity: because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it. That is, we'll have many cases of "zero probability n-grams" that should really have some non-zero probability.
  What's the problem with it?

1. We are underestimating the probability of all sorts of words that might occur, which will hurt the performance of any application we want to run on this data.
2. If the probability of any word in the test set is 0, the entire probability of the test set is 0. And perplexity is infinite

This was about when bigram probability of a few words is zero. But what if the unigram probability of some words is zero (words never seen before)

Unknown Words

In a closed vocabulary system, we know what all words can occur. No unknown words.
In other cases we have to deal with words we haven't seen before, the unknown words or out of vocabulary words. The percentage of OOV words that appear in the test set is called the OOV rate. An open vocabulary system is one in which we model these potential unknown words in the test set by adding a pseudo-word called <UNK>.
There are two ways to train the probabilities of the unknown word model.

1. Have a vocabulary fixed in advance: Then convert any OOV in training data to <UNK>.
2. Vocabulary not fixed in advance, created as we go through the training data: Any word below a frequence of (pre-defined) n, may be considered as an OOV, hence <UNK>
   Then, in both cases, estimate the probabilities of <UNK> from its counts just like any other regular word in the training set.

Smoothing

To prevent a language model from assigning zero probability to the unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen.
This modification is called smoothing or discounting.

Laplace smoothing (Add one smoothing)

Add one to all the bigram counts, before we normalize them into probabilities.
In case of unigram counts,

$$P(w_i) = \frac{c_i}{N} \qquad P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

In case of bigram: $$P^*_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

Instead of changing both the numerator and denominator, it is convenient to describe how a smoothing algorithm affects the numerator (how the count is changed), by defining an adjusted count c*. This adjusted count is easier to compare directly with the MLE counts and can be turned into a probability like an MLE count by normalizing by N (and not N+V this time).
To define this count, since we are only changing the numerator in addition to adding 1 we'll also need to multiply by a normalization factor N/(N+V)

$$c_i^* = (c_i + 1)\frac{N}{N + V}$$

We can now turn c* into P* by normalizing by N.

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$ (Bigram)

Another parameter used: relative discount:

$$d_c = \frac{c^*}{c}$$

(shows how much the count has been reduced)

Add-one smoothing makes a very big change to the counts. The sharp change in counts and probabilities occurs because too much probability mass is moved to all the zeros.

Add k smoothing
One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fractional count k (.5? .05? .01?). This algorithm is therefore called add-k smoothing.
Add-k smoothing requires that we have a method for choosing k; this can be done, for example, by optimizing on a devset. Not very efficient.

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

Backoff and Interpolation
If we are trying to compute P(wn|wn−2 wn−1) but we have no examples of a particular trigram wn−2 wn−1 wn, we can instead estimate its probability by using the bigram probability P(wn|wn−1). Similarly, if we don't have counts to compute P(wn|wn−1), we can look at the unigram P(wn).
(Sometimes using less context is a good thing, helping to generalize more for contexts that the model hasn't learned much about)

There are two ways to use this n-gram "hierarchy".
In **backoff,** we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram.
In **interpolation**, we always mix the probability estimates from all the n-gram estimators, weighing and combining the trigram, bigram, and unigram counts.

Two levels of interpolation:
Simple linear Interpolation, combining different n-grams by linearly interpolating all the models (simple weights)

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

Such that $\sum_i \lambda_i = 1$

Conditional linear interpolation, each λ weight is computed by conditioning on the context (previous words).
This way, if we have particularly accurate counts for a particular bigram, we assume that the counts of the trigrams based on this bigram will be more trustworthy, so we can make the λs for those trigrams higher and thus give that trigram more weight in the interpolation.

We

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2:n-1})P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2(w_{n-2:n-1})P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2:n-1})P(w_n)\end{aligned}$$

use the held-out corpora to find the optimal values of lambda. (Validation set)
(λ values that maximize the likelihood of the held-out corpus)
(EM algorithm, an iterative learning algorithm is one algorithm used)

In order for a backoff model to give a correct probability distribution, we have to discount the higher-order n-grams to save some probability mass for the lower order n-grams.
We'll need a function α to distribute this probability mass to the lower order n-grams.

This kind of backoff with discounting is also called Katz backoff. We rely on a discounted probability P* if we've seen this n-gram before otherwise backoff.

$$P_{\mathrm{BO}}(w_n|w_{n-N+1:n-1}) = \begin{cases} P^*(w_n|w_{n-N+1:n-1}), & \text{if } C(w_{n-N+1:n}) > 0 \\ \alpha(w_{n-N+1:n-1})P_{\mathrm{BO}}(w_n|w_{n-N+2:n-1}), & \text{otherwise.} \end{cases}$$

Katz backoff is often combined with a smoothing method called Good-Turing.

PARTS OF SPEECH TAGGING AND NAMED ENTITY RECOGNITION

Named entity is anything that can be referred to with a proper name: a person, a location, an organization.

Tasks in which we assign, to each word xi in an input word sequence, a label yi, so that the output sequence Y has the same length as the input sequence X are called **sequence labelling tasks**.

Parts of speech and named entities are useful clues to sentence structure and meaning.

- Knowing whether a word is a noun or a verb tells us about likely neighbouring words and syntactic structure, making part-of-speech tagging a key aspect of parsing.
- Knowing if a named entity like Washington is a name of a person, a place, or a university is important to many natural language understanding tasks.

PoS Tagging is useful in syntactic as well as semantic analysis, NER in semantic analysis.

PART OF SPEECH TAGGING: Taking a sequence of words and assigning each word a part of speech like NOUN or VERB

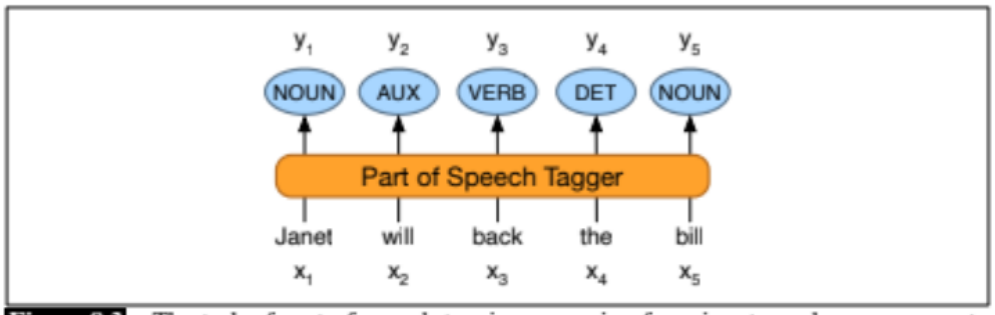


Figure 8.3 The task of part-of-speech tagging: mapping from input words $x_1, x_2, ..., x_n$ to output POS tags $y_1, y_2, ..., y_n$.

Tagging is a disambiguation task; words are ambiguous —have more than one possible part-of-speech—and the goal is to find the correct tag for the situation.
For example, book can be a verb (book that flight) or a noun (hand me that book).

Why is PoS Tagging hard?
The accuracy of parts of speech tagging is extremely high (over 97%).
85-86% of the word types are unambiguous (Janet is always NNP (proper noun), hesitantly is always RB(adverb)). But the ambiguous words, though accounting for only 14-15% of the vocabulary, are very common, and 55-67% of word tokens in running text are ambiguous. Particularly ambiguous common words include that, back, down, put and set.

Nonetheless, many words are easy to disambiguate, because their different tags aren't equally likely. Hence, given an ambiguous word, choose the tag which is most frequent in the training corpus. (assigning each token to the class it occurred in most often in the training set).
This idea provides us a baseline-
Most Frequent Class Baseline: Always compare a classifier against a baseline at least as good as the most frequent class baseline

How is it done?

- Rule Based: Dictionary is constructed with possible tags for each word.
  (Pre-decided) Rules guide the tagger to disambiguate. These are a set of handwritten rules and use contextual information to assign PoS tags to words. These rules are called 'Context Frame Rules'. Example rule: "If an ambiguous/unknown word X is preceded by a determiner and followed by a noun, tag it as an adjective."

- Stochastic/Probabilistic: A (training) text corpus is used to derive useful probabilities. The simplest stochastic approach finds out the most frequently used tag for a specific word in the annotated training data and uses this information to tag that word in the unannotated text.
  But sometimes this approach comes up with sequences of tags for sentences that are not acceptable according to the grammar rules of a language.

Hence, Probabilities are calculated for various tag sequences possible for the sentences and the sequence with the maximum probability is selected to assign the PoS tags.
Among the common models are Hidden Markov Model (HMM) and Maximum Entropy Model (MEM).

- Transformation based: They use a predefined set of handcrafted rules as well as automatically induced rules generated during training.

HIDDEN MARKOV MODEL

An HMM is a probabilistic sequence model: given a sequence of units, it computes a probability distribution over possible sequences of labels and chooses the best label sequence.
Markov Chain: It is a model that tells us about the probabilities of sequences of random variables/states and makes an assumption that only the current state is what matters while predicting the future. Formally, markov chain consists of:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

The Markov chain is useful when we need to find the probability of a sequence of observable events. (like the words in a sequence (n-grams was all about it))

In our case, we need to find the probability of sequences of hidden variables (the tags). We need to develop a markov chain of tags with the help of the observed events i.e. words. Here, HMM is useful.

For any model, such as an HMM, that contains hidden variables, the task of determining the hidden variables sequence corresponding to the sequence of observations (observed events) is called decoding. And that is what we want here.

Decoding: Given as input an HMM λ = (A,B) and a sequence of observations O = o1,o2,...,oT , find the most probable sequence of states Q = q1q2q3 ...qT .

Where, A = transition probability matrix (transition between states)
B = emission probability sequence (probability of an observation being generated from a hidden state) or (a word being generated by a tag)
[hidden variable (the PoS tag) is a state, and word is an observation]

For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence t1 ...tn that is most probable given the observation sequence of n words w1 ...wn:

$$\hat{t}_{1:n} = \underset{t_1 \dots t_n}{\operatorname{argmax}} P(t_1 \dots t_n | w_1 \dots w_n)$$

[tag sequence for which P(tag sequence| word sequence) is max]

To simplify further by applying some assumptions, we need to invert the form of conditional probability, hence apply Bayes Theorem. (That's why we needed B) (Similar happened in Naive Bayes)

$$\hat{t}_{1:n} = \underset{t_1 \dots t_n}{\operatorname{argmax}} P(w_1 \dots w_n | t_1 \dots t_n) P(t_1 \dots t_n)$$

(denominator ignored)

Applying assumptions:

1. Probability of a word appearing depends only on its own tag and is independent of neighbouring words and tags.

$$P(w_1 \dots w_n | t_1 \dots t_n) \approx \prod_{i=1}^{n} P(w_i | t_i)$$

2. Markov assumption: Bigram assumption: probability of a tag is dependent only on the previous tag, rather than the entire tag sequence.

$$P(t_1 \dots t_n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1})$$

Hence, we get

$$\hat{t}_{1:n} = \underset{t_1 \dots t_n}{\operatorname{argmax}} P(t_1 \dots t_n | w_1 \dots w_n) \approx \underset{t_1 \dots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i | t_i)}^{\text{emission}} \overbrace{P(t_i | t_{i-1})}^{\text{transition}}$$

Viterbi Algorithm: decoding algorithm for PoS tagging (efficient algorithm which uses HMM rather than brute force (try all combinations and choose with maximum probability)
A matrix is considered with each column having an observation, and each row having a tag (state). Dynamic programming is applied to find the most optimal path.

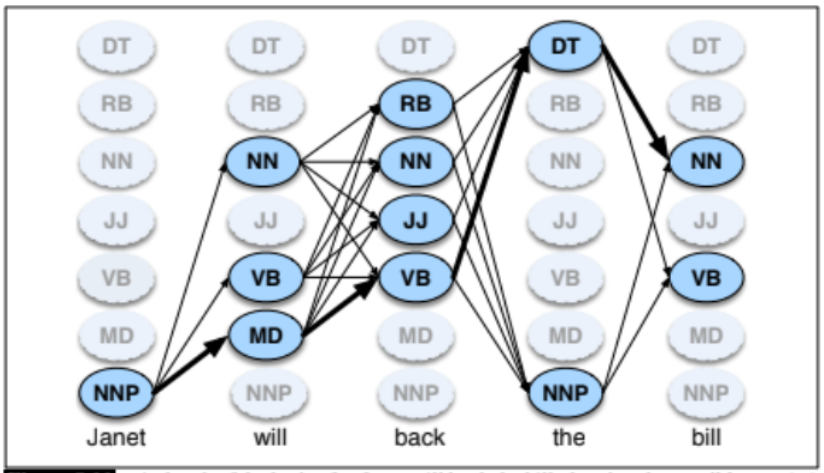


Figure 8.11 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the *B* matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

Given that we had already computed the probability of being in every state at time t −1, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state qj at time t, the value vt(j) is computed as

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

***Check Class Numerical**

MAXIMUM ENTROPY MODEL

…

NAMED ENTITY RECOGNITION (NER): The task of NER is to find spans of text that constitute proper names and tag the type of the entity. Example: PER (person), LOC (location), ORG (organization), or GPE (geo-political entity). (can also be date, time, price etc.)

(PoS tagging determines that the word is a noun, being a proper noun is a grammatical property of the word, NER will provide the semantic property: the type of entity)

How is NER useful?

- Building semantic representations, like extracting events and the relationship between participants.
- In sentiment analysis we might want to know a consumer's sentiment toward a particular entity.
- Entities are a useful first stage in question answering, or for linking text to information in structured knowledge sources like Wikipedia.

How is NER hard?

- Ambiguity of segmentation: in pos tagging, each word got one tag, here named entities can be over spans of text, difficult to define boundaries of what an entity is
- Type ambiguity: a noun can be a name, a place, an organization etc.

How is it done?

BIO tagging: It allows to treat NER like word-by-word sequence labelling task (like PoS Tagging), through tags that capture both the named entities and the boundary.
Iterate through the sentence and assign tags as follows:
Any token that begins a span of interest is labelled with B.
Any token that is inside the span of interest is labelled with I.
Any token that is outside the span of interest is labelled with O.
(Span of interest in any named entity)
Then the Named Entity follows.
For n named entities present, max 2*n+1 tags possible (n B, n I, 1 O)
(IO and BIOES tagging are variants, IO loses some information by not having the B tag, and BIOES adds and end tag E also, plus tag S for span with only one word)
Example:

[PER **Jane Villanueva**] of [ORG **United**] , a unit of [ORG **United Airlines Holding**] , said the fare applies to the [LOC **Chicago**] route.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Figure 8.7 NER as a sequence model, showing IO, BIO, and BIOES taggings.

UNIT-3

SYNTACTIC ANALYSIS

Context Free Grammars (12.1, 12.2), Grammar rules for English (12.3), Treebanks (12.4), Normal Forms for grammar(12.5), Ambiguity (13.1), Syntactic Parsing (NPTEL Lec 23), Dynamic Programming parsing (CKY: NPTEL LEC 24 25, UPTO 7 MIN), Probabilistic CFG, Probabilistic CYK (LEC 24,25), Dependency Grammar (NPTEL Lec 27) – Shallow parsing (13.5) – Probabilistic Lexicalized CFGs – Feature structures, Unification of feature structures (pdfs)

Syntax refers to the way words are arranged together, and relations between them.

Constituency
There's something called 'Constituency' which has a role to play in this arrangement. Constituency is the idea that groups of words can behave as single units, or constituents.

Example: Consider a Noun Phrase, a constituent in English. A Noun phrase is a sequence of words surrounding at least one noun. Examples:

Harry the Horse	a high-class spot such as Mindy's
the Broadway coppers	the reason he comes into the Hot Box
they	three parties from Brooklyn

Why are these a constituent (why can they be considered as a single group)?

1. They can all appear in similar syntactic environments, for example, for a noun phrase: before a verb. (This is true for the whole phrase but not individual words, hence the phrase is a constituent) (* means not possible)

three parties from Brooklyn *arrive...*	*from *arrive...*	*as *attracts...*
a high-class spot such as Mindy's *attracts...*	*the *is...*	*spot *sat...*
the Broadway coppers *love...*		
they *sit*		

2. The whole phrase can be preposed (placed at beginning) or postposed (or in between, according to the syntactic environment) in a sentence as a single entity only, not individual words.
   Example: ***On September seventeenth***, I'd like to fly from Atlanta to Denver
   On September*, I'd like to fly ***seventeenth*** from Atlanta to Denver

To check if it's a noun phrase or verb phrase etc, try substituting with corresponding PoS. Now, how can we model these constituents? (which words together can make a NP, which cannot, etc)? Using CFG

Context Free Grammar
Also called Phrase-structure Grammar and formalism of the grammar is equivalent to Backus-Naur Form (BNF)

CFG consists of:
- Production Rules (how to group or order symbols together). Two types:
  - Grammar (set of rules generating non terminals on RHS)
  - Lexicon of words and symbols (set of rules which introduce terminals)
    (Both are considered as a single set in the formal definition of CFG)
- Symbols. Two categories:
  - Terminals correspond to words in the language
  - Non-terminals express abstraction over terminals.
    Hence, the non-terminal associated with each word in the lexicon is its lexical category, or **part of speech.** Also called pre-terminals.

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of **rules** or productions, each of the form $A \rightarrow \beta$, where A is a non-terminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$
S	a designated **start symbol** and a member of N

- A start symbol

Grammar Rules		Examples
$S \rightarrow$	*NP VP*	I + want a morning flight
$NP \rightarrow$	*Pronoun*	I
|	*Proper-Noun*	Los Angeles
|	*Det Nominal*	a + flight
Nominal $\rightarrow$	*Nominal Noun*	morning + flight
|	*Noun*	flights
$VP \rightarrow$	*Verb*	do
|	*Verb NP*	want + a flight
|	*Verb NP PP*	leave + Boston + in the morning
|	*Verb PP*	leaving + on Thursday
$PP \rightarrow$	*Preposition NP*	from + Los Angeles

Figure 12.3 The grammar for $\mathcal{L}_0$, with example phrases for each rule.

Noun $\rightarrow$	*flights* | *breeze* | *trip* | *morning*
Verb $\rightarrow$	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective $\rightarrow$	*cheapest* | *non-stop* | *first* | *latest* | *other* | *direct*
Pronoun $\rightarrow$	*me* | *I* | *you* | *it*
Proper-Noun $\rightarrow$	*Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American*
Determiner $\rightarrow$	*the* | *a* | *an* | *this* | *these* | *that*
Preposition $\rightarrow$	*from* | *to* | *on* | *near*
Conjunction $\rightarrow$	*and* | *or* | *but*

The lexicon for $\mathcal{L}_0$.

(Nominal describes words/groups of words that function together as a noun, like adj+noun)

The two set of productions: Grammar and Lexicon

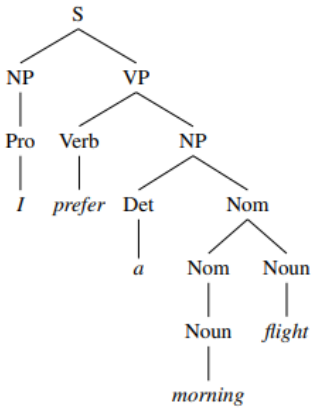
CFG can be used to generate a set of strings, through a series of rule expansions which is called **derivation** of the string. A derivation is generally expressed through a **parse tree**. We can also represent a parse tree in a more compact format called **bracketed notation**.
Example: $[_S\ [_{NP}\ [_{Pro}\ \text{I}]]\ [_{VP}\ [_V\ \text{prefer}]\ [_{NP}\ [_{Det}\ \text{a}]\ [_{Nom}\ [_N\ \text{morning}]\ [_{Nom}\ [_N\ \text{flight}]]]]]]$

The **formal language** defined by a CFG is the set of strings that are derivable from the designated start symbol, and all those strings/sentences are called **grammatical sentences**, the rest are called **ungrammatical sentences**.
This logic is a very simple model of how natural languages work. This is because determining whether a given sentence is part of a given natural language often depends on the context. (But here we present a clear cut boundary)
The use of formal languages to model natural languages is called **generative grammar**.

A CFG can be thought of in two ways: as a device for generating sentences (string generation) and as a device for assigning a structure to a given sentence (parsing, string given).
The problem of mapping from a string of words to its parse tree is called **syntactic parsing**.

Grammar Rules for English

- **Sentence Level Construction**
  4 most common constructions for english sentences:
  Declarative Structure: (NP VP) (NP is subject)
  Examples:
  ~ I prefer a morning flight **(Parse tree based on above grammar)** ⇒
  ~ I want a flight from Ontario to Chicago
  ~ The flight should be eleven a.m. tomorrow



  Imperative Structure: (VP) (No subject)
  Always used for commands/suggestions, hence imperative
  Examples:
  ~ Show the lowest fare
  ~ Give me Sunday's flights arriving in Las Vegas from New York
  ~ List all flights between five and seven p.m.

  Yes-No question Structure: (S → Aux NP VP) (Aux is auxiliary verb)
  Used for asking/requesting/suggesting (Pragmatic functions)
  Examples:
  ~ Do any of these flights have stops?
  ~ Does American's flight eighteen twenty five serve dinner?
  ~ Can you give me the same information for United?
  ….

Treebanks
Treebank is a syntactically annotated corpus. This means that every sentence in the collection is paired with a corresponding parse tree. Example: Penn Treebank
….

Normal Forms for Grammar
It is possible to have two distinct CFGs that generate the same language. They are called equivalent grammars.
There are two types of equivalences:
- Strong equivalence: generate same set of strings along with assigning the same phrase structure to each sentence (hence they only differ in what non-terminal symbols are used, rules are same)
- Weak equivalence: generate same set of strings but not phrase structure

Hence, it is useful to have a normal form for grammars, in which each production takes a particular form (can map to one standard form)

For example: Chomsky Normal Form, it is an epsilon free grammar with productions only of the form A->BC or A->a.
Every CFG can be converted into weakly equivalent CNF.
Chomsky normal form grammars are binary branching, that is they have binary trees.
Sometimes using binary branching can actually produce smaller grammars.
For example, VP -> VBD NP PP* (This is kinda a RE) is represented in the Penn Treebank by this series of rules:
VP → VBD NP PP
VP → VBD NP PP PP
VP → VBD NP PP PP PP
VP → VBD NP PP PP PP PP
...
but could also be generated by the following two-rule grammar:
VP → VBD NP PP
VP → VP PP

The generation of a symbol A with a potentially infinite sequence of symbols B with a rule of the form A → A B is known as **Chomsky-adjunction.** (Recursion)

Ambiguity in Parsing

Called as Structural Ambiguity: when a sentence can have more than one parse tree.
Two kinds:
Attachment ambiguity: if a constituent can be attached to a parse tree at more than one place.
Ex: I shot an elephant in my pajamas, the man saw the girl with a telescope.

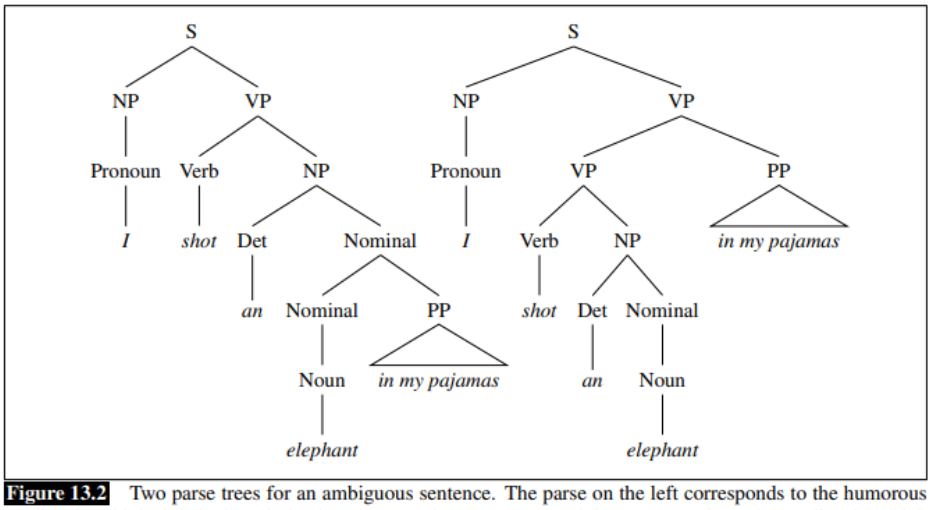


Figure 13.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

In the first case, (an elephant in my pajamas) comes as NP (Gadbad)
In the second case, (shot an elephant) is a VP and then (in my pajamas) is a PP

Coordination ambiguity: in this, phrases can be conjoined by a conjunction like and.
Ex. old men and women (both men and women are old, or just the men are old)

Syntactic Parsing (Constituency Parsing): Top Down and Bottom up Approaches (NPTEL LEC-23)
Top down vs Bottom up
Top down never explores options that will not lead to a full parse (it backtracks as soon as the PoS category doesn't match the terminal according to the lexicon). But it can explore

many options that never connect to the actual sentence. (TopDown considers each rule with required LHS and checks it, hence there is no connection with the actual sentence).

Bottom up never explores options that do not connect to the actual sentence (It starts from the words, hence words determine what rules to check upon, rules are not chosen randomly (as was being done in top down: all rules with required LHS)). But it can explore options that can never lead to a full parse (it choses rules according to words, those rules' LHS may not be heading to S (i.e. to be parsed)

Instead of backtracking, an optimal polynomial time algorithm can be used considering the context-free nature of the grammar. The dynamic programming advantage arises from the **context-free nature** of our grammar rules - the results can be cached and stored later irrespective of context.
DP algorithms based on both top-down and bottom-up search can achieve O(n^3) recognition time (n: length of the sentence)

One popular algorithm: CKY, a bottom up algorithm, and requires normalizing the grammar.

Dynamic Programming Parsing (CKY) (NPTEL LECTURE-24)
Step 1: Convert to CNF

- Convert terminals within rules to dummy non-terminals
- Convert unit productions
- Make all rules binary and add them to new grammar

Original Grammar

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP
Pronoun → I | he | she | me
Noun → book | flight | meal | money
Verb → book | include | prefer
Proper-Noun → Houston | NWA

Chomsky Normal Form

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
PP → Prep NP
Pronoun → I | he | she | me
Noun → book | flight | meal | money
Verb → book | include | prefer
Proper-Noun → Houston | NWA

CKY algorithm suffers from ambiguities, there can be more than one start symbol in the final top right cell. We can't say which parse is more probable.

Hence, we use Probabilistic Context Free Grammar (PCFG) in which each rule is assigned a probability. Hence the grammar has 5 tuples now.

R: Rules/productions of the form $X \rightarrow \gamma$, $X \in N$ and $\gamma \in (T \cup N)*$
$P(R)$ gives the probability of each rule.

$$\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$$

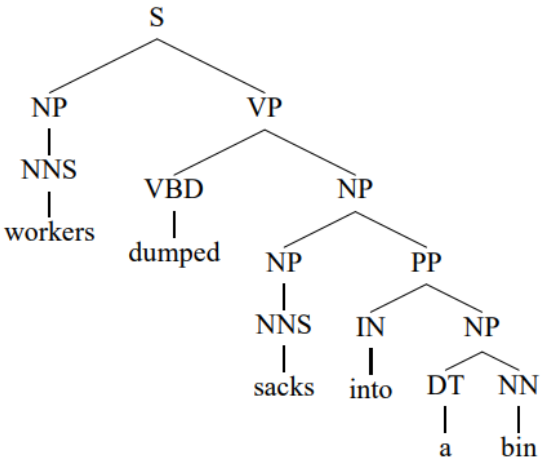
Features of PCFG:

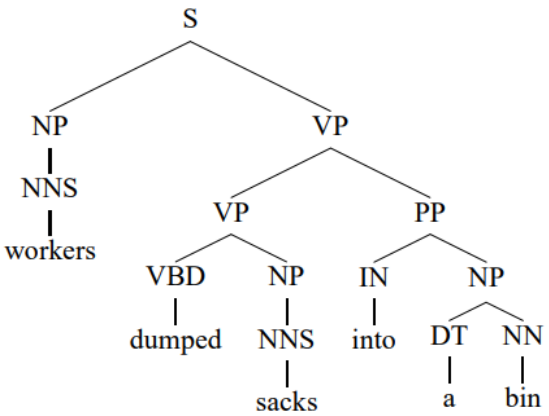
- As the number of possible trees increases, Pcfg can give an idea **about the plausibility of a particular parse.**
- But the probability estimates are purely based on structural factors, and do not factor in lexical co-occurrence. Thus PCFG doesn't give a very good idea of the **plausibility of the sentence**. (Hence, in general language models are preferred over PCFGs to find the probability of a sentence. It's only good to find the probability of a parse tree)
- However it can be useful in real texts by assigning a very low probability to grammatical mistakes. (PCFG rules out nothing: no clear cut boundary)
- All else being equal, the probability of a larger tree is less than that of a smaller tree (another issue)

How to use this PCFG?
(CKY for PCFG: Lec 25)

Lexicalized PCFG

Problems with PCFG
The PCFGs generate lexical items as an afterthought, conditioned only on the POS directly above them in the tree. This is a very strong independence assumption, which leads to non-optimal decisions being made by the parser in many important cases of ambiguity.
(This assumption was also made in HMM, ughh)
For example:
PP attachment ambiguity: workers dumped sacks into a bin.
(NP workers) (VP (VP dumped sacks) (PP into a bin)) [PP in VP]
(NP workers) (VP (Verb dumped) (NP sacks (PP into a bin))) [PP in NP]





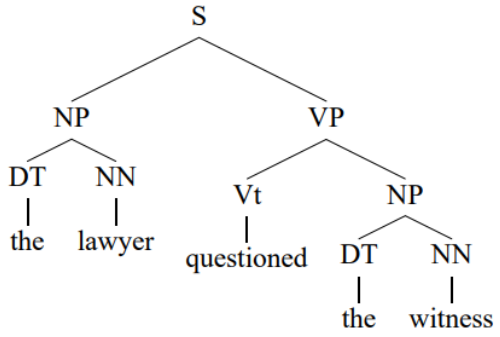
It can be inferred from the lexical items that PP in VP is much better than PP in NP. (this is the case for lexeme 'into', opposite for 'of'). This information is not used in PCFG.

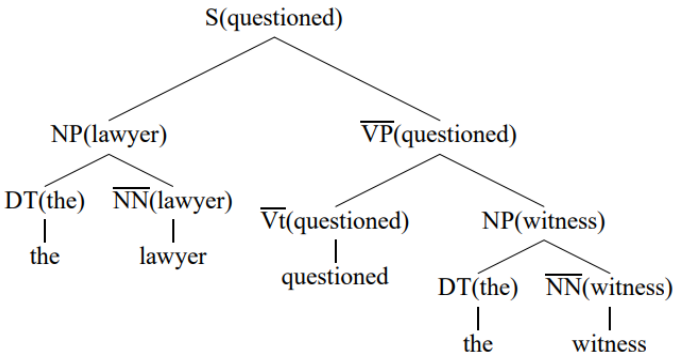
Example: Coordination Ambiguity
There are two parse trees with the same context rules but just the difference in order. PCFG will assign the same probability to both of them, but from the lexical items it can be easily inferred which one is the better one/correct one. (e.g. old men and women)
Hence one issue with PCFG is lack of sensitivity to lexical information.

Similarly, PCFG also lacks sensitivity to structural information.
There is generally a huge bias towards close attachment (attaching to the most recent VP/NP etc.) versus when attached to a distant phrase. But this is also not considered in PCFG.

Example: president of a company in Africa. (company in africa, or president in africa? Close attachment says company)
Example: John was believed to have been shot by Bill (Bill modifies which verb? Shoot or believe? Close attachment says 'shot by Bill')

Lexicalization of a Treebank
A solution to the above problem: including lexical items in non-terminals (This will lead to huge numbers of rules but, this will be not be talked about)
How is it done?
Identify the head of each context free rule which refers to the most important child or the center of the rule. Propagate that lexical item to the parent and so on (bottom up).





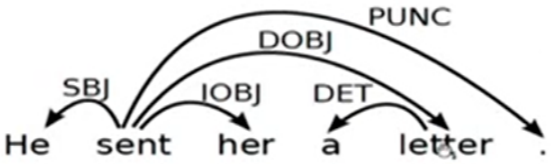
Lexicalized PCFG in Chomsky Normal Form
6 tuples: G = (N, Σ, R, S, q, γ)
N: finite set of non-terminals
Sigma: finite set of terminals
R: rules, Three forms:

1. $X(h) \rightarrow_1 Y_1(h)\ Y_2(m)$ *where* $X, Y_1, Y_2 \in N,\ h, m \in \Sigma.$
2. $X(h) \rightarrow_2 Y_1(m)\ Y_2(h)$ *where* $X, Y_1, Y_2 \in N,\ h, m \in \Sigma.$
3. $X(h) \rightarrow h$ *where* $X \in N,\ h \in \Sigma.$

q: parameter (probability) associated with each rule, with condition that it is greater than 0 and for each LHS each rule's probability sums up to 1.
Gamma: For each $X \in N$, $h \in \Sigma$, there is a parameter $\gamma(X, h)$. We have $\gamma(X, h) \geq 0$, and Sigma (sum over all) $X \in N, h \in \Sigma\ \gamma(X, h) = 1$. It is the probability of choosing X(h) as the root of the tree.
Hence, to calculate the probability of the tree, calculate just as a normal PCFG and then multiply by gamma(X,h).

Dependency Parsing (NPTEL LEC-27)
Dependency parsing is another family of grammar formalisms where phrase structures do not play a direct role. The syntactic structure of a sentence is solely described in terms of the words in the sentence and an associated set of binary, directed, grammatical relations between them.
The tree doesn't include phrase structure but words only.
Advantage of dependency parsing is their ability to deal with morphologically rich languages which have a relatively free word order. For each different order, there would be another rule in phrase structure grammar but just one link in dependency parsing. A dependency

grammar approach abstracts the word order information, representing only the information that is necessary for the parse.
Another advantage is that head dependent relations provide an approximation to the semantic relationship between predicates and their arguments that makes them directly useful for many applications.

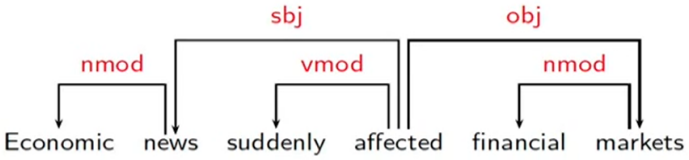


The dependencies are between head words and dependent words.

- The head word can replace the whole construction. (Whole construction = head + dependent) This is because head determines the syntactic category of the whole construction. (letter governs and can replace ‘a letter’). The dependent can be optional.
- Dependent specifies head. Example: ‘he sent’ is a head, while ‘her’ specifies sent to whom, hence the dependent specifies the head.
- Head words can manipulate the form of dependent words as well as their position (it’s relative to the head word)

These relations can be:
Endocentric: one word can satisfy the requirement of the whole construction (markets in ‘financial markets’, affected in ‘suddenly affected’)
Exocentric: one word cannot the fulfil the requirement of the whole construction (affected in ‘news affected’)

Construction	Head	Dependent
Exocentric	Verb	Subject (sbj)
	Verb	Object (obj)
Endocentric	Verb	Adverbial (vmod)
	Noun	Attribute (nmod)



Phrase Structure Grammars explicitly represent the phrases (non-terminals) and structural categories. (non-terminal labels)
Dependency Structures explicitly represent the head-dependent relations (the directed arcs) and functional categories (the arc labels)

A dependency structure can be defined as a dependency graph. (nodes and arcs)
Conditions for these dependency graphs:

- Graph should be acyclic (head is head and dependent is dependent!)
- Graph should be connected (no node is isolated)
- Graph should follow single head constraint (no multiple heads allowed for a node, but one head can have multiple dependents)
- Graph should be projective

If i->j and j->*k (* means not necessarily a direct edge) then j and k should be on the same side of i. This rule needs to be followed in not all languages.

Feature Structures

(1) Need

We have a grammar for 'The gangster dies'. Now we want it to generate 'The gangsters die' also. For this we add two more rules containing these new words in the lexicon. But now even the sentences 'The gangsters dies' and 'The gangster die' will also be generated.
To prevent this (to ensure singular NP comes with singular VP), we need to segregate the two types of NPs and VPs.
Because of this, the whole grammar eventually doubles in size. Although the grammar produces the desired results now, there's a problem with this approach:

1. Size

- What if we want to add 'you' and 'I', we need to have separate rules for the first, second and third person.
- Here we have the DT as 'The', in case we allowed 'a', we had to take care of when to use a, when an, and when the → more rules
- Adding transitive verbs and pronouns (e.g. he, him). To differentiate between 'he shot him' and 'him shot he', more rules needed.

Hence, eventually the size of the grammar will explode and rules will be difficult to read.

2. We lose the generalization.

- In the above grammar, NPsg and NPplur are just as similar to each other as with VPsg. The generalization that they are both NPs is lost.
- Sentence contains an NP followed by a VP, lost.

Hence, a better way (the solution to this):
Consider that non-terminals are no longer atomic category symbols, but a set of properties. Certain rules can then impose constraints on the individual properties that a category involved in that rule may have. Hence the above grammar can be written as:

$S \longrightarrow NP\ VP$: number of NP = number of VP
$NP \longrightarrow Det\ N$
$VP \longrightarrow IV$
$Det \longrightarrow the$
$N \longrightarrow gangster$: number of N = singular
$N \longrightarrow gangsters$: number of N = plural
$IV \longrightarrow dies$: number of IV = singular
$IV \longrightarrow die$: number of IV = plural

(number means singular or plural)

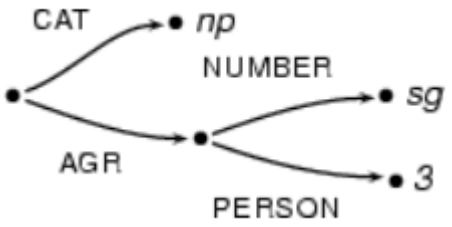
Such sets of properties are commonly represented as feature structures.

(2) Representation

1. Attribute value matrices
2. Directed Graphs (features on arcs, values on nodes) (Feature values need not to be atomic)

$$\begin{bmatrix} \text{NUMBER} & sg \\ \text{PERSON} & 3 \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT} & np \\ \text{AGR} & \begin{bmatrix} \text{NUMBER} & sg \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix}$$



Here, the paths ⟨HEAD, AGR⟩ and ⟨HEAD, SUBJ, AGR⟩ both lead to the same node, i.e., they *share that value*. This property of feature structures that several features can share one value is called ***reentrancy*****.**

In attribute value matrices, reentrancy is commonly expressed by coindexing the values which are shared.

$$\begin{bmatrix} \text{CAT} & s \\ \text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{NUM} & sg \\ \text{PERS} & 3 \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Operations on feature structures:

(3) Subsumption

- A feature structure A subsumes B iff all content of A is in B (subsumes: is a subset of)
- Feature structures with the same content but different order are also the same. (order doesn't matter)

(4) Unification

- Union operation (new features contains information of both and nothing more)
- If the two feature structures contain contradictory information, then the union is undefined. Hence, it is called a partial operation. (Example: CAT vp, CAT np)
- Another way to define: unification of two features structures is the smallest feature that is subsumed by both.
  Hence, F union G if exists, possesses following properties:
  1. F subsumes (F union G)
  2. G subsumes (F union G)
  3. If H is a feature structure such that (F subsumes H) and (G subsumes H), then ((F union G) subsumes H) too

$$\begin{bmatrix} \text{AGREE} & \begin{bmatrix} \text{NUMBER} & \text{singular} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREE} & \begin{bmatrix} \text{NUMBER} & \textit{singular} \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREE} & \begin{bmatrix} \text{PERSON} & \textit{third} \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \text{AGREE} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \textit{singular} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREE} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

These are F1, F2 and F3.
Below are Union of F1&F2, and F2&F3

$$\begin{bmatrix} \text{AGREE} & \begin{bmatrix} \text{NUMBER} & \text{singular} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREE} & \begin{bmatrix} \text{NUMBER} & \textit{singular} \\ \text{PERSON} & \textit{third} \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \text{AGREE} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \textit{singular} \\ \text{PERSON} & \textit{third} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREE} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

(5) Generalization

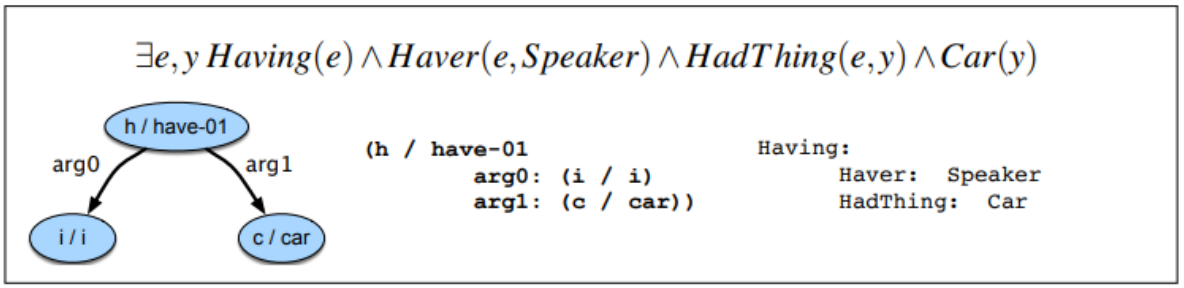
- Intersection operation. It is not a partial operation but whole. In case two features have contradictory info also, the result is produced: empty feature structure.
- Largest feature substructure that subsumes both the feature structure.
  Conditions:
  1. F intersection G subsumes F
  2. It also subsumes G
  3. If H is a feature structure such that it subsumes both, then H subsumes union of both

UNIT-4

SEMANTICS AND PRAGMATICS

Requirements for representation, First-Order Logic, Description Logics, Semantic attachments – Word Senses, Relations between Senses, Thematic Roles, selection restrictions – Word Sense Disambiguation, WSD using Supervised, Dictionary & Thesaurus, Word Similarity using Thesaurus and Distributional methods (NPTEL LEC 33, 34)

The meaning of linguistic expressions can be captured in formal structures called mental representations.
The process whereby such representations are created and assigned to linguistic inputs is called semantic parsing or semantic analysis.



Different mental representations shown, first order logic (top), using directed graphs (Abstract Meaning Representation (AMR)), frame-based (right) of the sentence *'I have a car'*

These representations can be viewed from at least two distinct perspectives in all of these approaches: as representations of the meaning of the particular linguistic input *'I have a car'*, and as representations of the state of affairs in some world (feeded in the machine). It is this dual perspective that allows these representations to be used to link linguistic inputs to the world and to our knowledge of it.

Why meaning representations are needed, what is expected from them, how will they help?

- **Verifiability:**
  Does Maharani serve vegetarian food?
  Verifiability is a system's ability to compare the state of affairs described by a representation to the state of affairs in some world as modelled in a knowledge base.
  For this comparison, it needs a suitable representation:
  e.g. Serves(Maharani,VegetarianFood)
  System will match this against its knowledge base of facts about particular restaurants, and finds a representation matching this proposition.

- **Ambiguity:** Words and sentences can have different meaning representations depending upon the context. They are not free from ambiguity.
  e.g. I wanna eat someplace that's close to ICSI.
  But our meaning representation must be unambiguous, so that the system can reason over a representation that means either one thing or the other in order to decide how to answer.

  Vagueness, related to ambiguity**:** meaning representation leaves some part of the meaning underspecified. Vagueness does not give rise to multiple representations.
  e.g. I want to eat Italian food.
  While Italian food may provide enough information to provide recommendations, it is nevertheless vague as to what the user really wants to eat.
  Sometimes vagueness is appropriate.

- **Canonical Form:** Distinct inputs that mean the same thing should have the same meaning representation. This approach greatly simplifies reasoning, since systems need only deal with a single meaning representation for a potentially wide range of expressions. For example: active/passive sentences, or using synonyms.

- **Inference and Variables:**
  Can vegetarians eat at Maharani?
  Knowledge Base contains 'Maharani servers veg'.
  This fact and the question don't have the same meaning but have a common-sense connection (between what vegetarians eat and what vegetarian restaurants serve)
  To answer this question, we need to connect the meaning representation of this request with this fact about the world in the knowledge base.
  A system must be able to draw valid conclusions (use inference) based on the meaning of the input and background knowledge.

  I'd like to find a restaurant where I can get vegetarian food.
  This request does not make reference to any particular restaurant; the user wants information about an unknown restaurant that serves vegetarian food. Since no restaurants are named, simple matching is not going to work. Answering this request requires the use of variables, using some representation like the following:
  Serves(x,VegetarianFood)
  Matching succeeds only if the variable x can be replaced by some object in the knowledge base in such a way that the entire proposition will then match.

- **Expressiveness:**
  A meaning representation scheme must be expressive enough to handle a wide range of subject matter, ideally any sensible natural language utterance. First Order Logic is one such representation.

How does a meaning representation meet these expectations, bridging the gap from formal representations to representations that tell us something about some state of affairs in the world?
Through a **model**. A model is a formal construct of the particular state of world affairs. Expressions in a meaning representations can map to elements of a model, example, objects, their properties and relations between them. If this mapping is accurate, this bridges that gap.

Terminologies:
The vocabulary of a meaning representation consists of two parts: the non-logical vocabulary and the logical vocabulary.

Non-logical: Open-ended set of names for the objects, properties, and relations that make up the world we're trying to represent. These appear in various schemes as predicates, nodes, labels on links.
Logical: Closed set of symbols, operators, quantifiers, links, etc., that provide the formal means for composing expressions in a given meaning representation language.

Each element of the non-logical vocabulary must have a **denotation** in the model, meaning that every element corresponds to a fixed, well-defined part of the model.

The **domain** of a model is the set of objects that are being represented. Each distinct concept, category, or individual denotes a unique element in the domain.
Properties of objects in a model is a set of objects possessing that property.
Relations between objects in a model is a set of tuples of objects in that relation.
This approach to properties and relations is called **extensional**, because we define concepts by their extension, their denotations

Interpretation: Mapping from the non-logical vocabulary of our meaning representation to the proper denotations in the model.

Example:

Domain	$\mathcal{D} = \{a,b,c,d,e,f,g,h,i,j\}$
Matthew, Franco, Katie and Caroline	a,b,c,d
Frasca, Med, Rio	e,f,g
Italian, Mexican, Eclectic	h,i,j
Properties	
Noisy	$Noisy = \{e,f,g\}$
Frasca, Med, and Rio are noisy	
Relations	
Likes	$Likes = \{\langle a,f\rangle, \langle c,f\rangle, \langle c,g\rangle, \langle b,e\rangle, \langle d,f\rangle, \langle d,g\rangle\}$
Matthew likes the Med	
Katie likes the Med and Rio	
Franco likes Frasca	
Caroline likes the Med and Rio	
Serves	$Serves = \{\langle f,j\rangle, \langle g,i\rangle, \langle e,h\rangle\}$
Med serves eclectic	
Rio serves Mexican	
Frasca serves Italian	

Figure 15.2 A model of the restaurant world.

We're deliberately using meaningless, non-mnemonic names for our domain elements to emphasize the fact that whatever it is that we know about these entities has to come from the formal properties of the model and not from the names of the symbols.

Verify the representation asserting that Matthew likes Frasca.
- First use our interpretation function to map the symbol Matthew to its denotation a, Frasca to e, and the Likes relation to the appropriate set of tuples.
- We then check that set of tuples for the presence of tuple <a,e>

This simple search will not work for complex meanings, we would need the help of operators like conjunction, disjunction, negation etc. Hence, determining the truth of a complex

expression would be done from the meanings of the parts (by consulting a model) and the meaning of an operator by consulting a truth table. This is called **truth-conditional semantics.**

First Order Logic
Flexible, well understood, computationally tractable meaning representation.

Representing Objects
A **term** represents an object. Term can be a function, a constant or a variable.
Constants are the specific objects in the world being described. (e.g. Proper nouns) One constant can only refer to one object. Vice versa is not true however. They are capitalized (initials)
Functions represent objects in terms of relations/associations with another object. They are syntactically similar to predicates, but are actually terms only because they refer to unique objects.
Variables let us make assertions and draw inferences about objects without having to make reference to any particular named object. Represented as single lower case letters.

Representing Relations
Predicates represent relations.
One place predicates (with one argument) are used to assert **PROPERTIES** of the object.
e.g. Restaurant(Maharani)
Predicate refers to the name of the relation and the relation with the arguments when applied, is called **an atomic formula.**

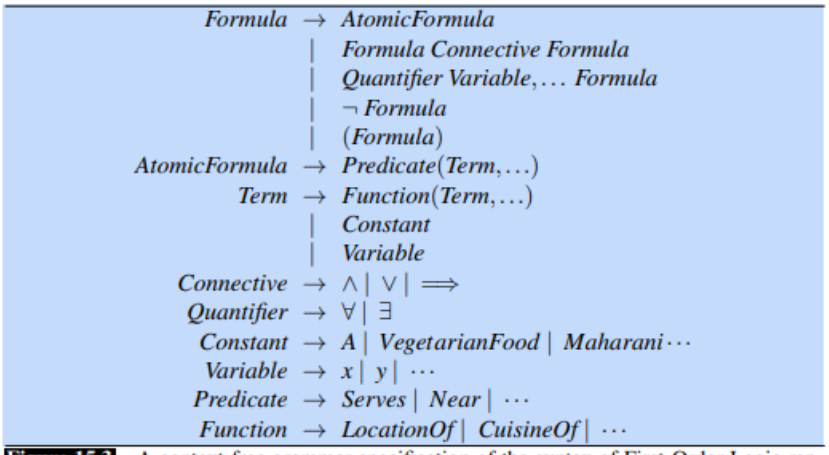
Quantifiers
The variables can be used in two ways: to refer to particular anonymous objects and to refer generically to all objects in a collection. These two uses are made possible through the use of operators known as quantifiers.

Existential quantifier: the first case (refer to an unknown/indefinite object)
e.g. a restaurant that serves Mexican food near ICSI.
The first order representation of this sentence will be:
$\exists$ xRestaurant(x) $\wedge$ Serves(x,MexicanFood) $\wedge$ Near((LocationOf(x),LocationOf(ICSI))
(The existential quantifer states that for the above sentence to be true, there must be atleast one restaurant to fulfill the conditions)

Universal quantifier: the second case; the substitution of any object in the knowledge base for the universally quantified variable should result in a true formula.

Hence, variables in logical formulas must be either existentially ($\exists$) or universally ($\forall$) quantified. To **satisfy an existentially quantified variable**, at least one substitution must result in a true sentence. To **satisfy a universally quantified variable**, all substitutions must result in true sentences.

Lambda Notation (doubtful)
Abstraction of FOL formulas.
(Lambda followed by variable)* followed by the formula
Example: λx.P(x) when applied on formula A -> λx.P(x)(A) = P(A) (This step is called lambda reduction)
Example: λx.λy.Near(x, y) when applied on term Bacaro -> λx.λy.Near(x, y)(Bacaro)
= λy.Near(Bacaro, y).
Further, λy.Near(Bacaro, y)(Centro) = Near(Bacaro,Centro);



Formula	→	*AtomicFormula*
	|	*Formula Connective Formula*
	|	*Quantifier Variable*, ... *Formula*
	|	¬ *Formula*
	|	(*Formula*)
AtomicFormula	→	*Predicate*(*Term*, ...)
Term	→	*Function*(*Term*, ...)
	|	*Constant*
	|	*Variable*
Connective	→	∧ | ∨ | ⟹
Quantifier	→	∀ | ∃
Constant	→	*A* | *VegetarianFood* | *Maharani* ···
Variable	→	*x* | *y* | ···
Predicate	→	*Serves* | *Near* | ···
Function	→	*LocationOf* | *CuisineOf* | ···

Figure 15.3 A context-free grammar specification of the syntax of First-Order Logic representations. Adapted from Russell and Norvig (2002).

Inference

Modus Ponens:
a
a->b
b
Where a and b are formulas
Modus Ponens can be put to practice in two ways:
Forward Chaining: As individual facts are added to the knowledge base, modus ponens is used to fire all applicable implication rules until all facts have not been deducted.
Advantage is that facts will be present in the knowledge base when needed, less response time.
Disadvantage is that those facts which will never be needed may be inferred and stored.

Backward Chaining: Modus Ponens run in reverse.
Firstly, check if query formula is true by finding its presence in the knowledge base.
Example: Serves(Leaf ,VegetarianFood) needs to be verified
If this is present, verified. Stop. If it's present, that means it's true.

If this is not present in the knowledge base.
Search for applicable implication rules in the knowledge base.
An applicable rule is one whereby the consequent of the rule matches the query formula.
If there are any such rules, then the query can be proved if the antecedent of any one of them can be shown to be true. This can be performed recursively by backward chaining on the antecedent as a new query. (Because antecedent being true cannot have consequent being false) (T -> F produces F)
(However, F->T produces True, that means antecedent can be false also and consequent true, but this case cannot be checked because false antecedent can also produce false consequent. Hence this method is not exhaustive.)
Backward chaining is from queries to known facts.
There's something called reasoning backward which is from known consequents to unknown antecedents.
(Both are different)

While forward and backward reasoning are sound, neither is complete. This means that there are valid inferences that cannot be found by systems using these methods alone. Fortunately, there is an alternative inference technique called resolution that is sound and complete. Unfortunately, inference systems based on resolution are far more computationally expensive than forward or backward chaining systems.

Until now, we talked about how we interpret the mental representations with the help of the models which are present in the knowledge base, but how do we come up with the mental representation from the sentence (the language)?
This is dealt in **lexical semantics**. It is concerned with meaning related connections between lexical items. A lexical item, or a lexeme consists of a form (the orthographic (the written form) and phonological (pronunciation) part) and a sense (the meaning).
The issue that comes in this task is that of having different senses of a word.
One word can have several meanings. They are called homonyms.
Our task here would be to find connection between lexemes or word senses and word sense disambiguation (so as to connect the correct sense of the word), to get a proper mental representation.

Word Senses
A sense is a discrete representation of one aspect of the meaning of the word. Example: bank

How can we define the meaning of a word sense?
A word can be represented as an embedding, a point in semantic space, such that the meaning of a word can be defined by its co-occurrences, the counts of words that often occur nearby. But that doesn't tell us how to define the meaning of a word sense. (both interpretations of bank will get mixed there)

- Dictionary and Thesaurus give textual definitions for each sense, which is called gloss.
- Defined through relations between various senses. (For example, defining right and left with respect to each other. Any other way to do so? (no))

(Hence disambiguating word senses also involves finding relations between them)
All such relations are listed in an online database called WordNet.

How to differentiate between two senses of a word?
Use them in the same sentence, if the sentence seems ill formed, the two senses are different/discrete

- Which of those flights serve breakfast?
- Does Air France serve Philadelphia?
- *Does Air France serve breakfast and Philadelphia?

This kind of conjunction of antagonistic readings is called **zeugma**.

Relations between senses
Homonymy: Same form but different meanings. Form consists of orthographic as well as phonological part. Ex: bat/bank.
If the pronunciation is the same (written form may be different) and the meanings are different, they are called homophones. Example: write and right, piece and peace
If the orthographic part is the same (pronunciation may be different) and the meanings are different, they are called homographs. Example: bass (a fish) and bass (related to music)

Polysemy: Same form but different meanings, but still those meanings are semantically related (not too different).
Example:
The bank was constructed in 1875 out of red brick.
I withdrew the money from the bank.
Sense 1: the building belonging to the financial institution
Sense 2: the financial institution
Polysemy in which one aspect of an entity/concept is referred to other aspects of that entity itself, is called metonymy. The above example is **metonymy**.
'Thomas Hardy wrote this book' (the author) vs I love Thomas Hardy (the works of author)

‘The chicken was domesticated in Asia’(animal) vs ‘The chicken was overcooked’ (meat)
The two meanings of the word ‘serve’ in the zeugma test can be considered polysems.

Synonymy: Different forms but same meaning. e.g. couch/sofa, water/H2O
Synonymy is actually a relationship between senses rather than words.
Example: big and large seem synonyms in:

- How big is that plane?
- Would I be flying on a large or small plane?

But not here:

- Miss Nelson, for instance, became a kind of big sister to Benjamin.
- *Miss Nelson, for instance, became a kind of large sister to Benjamin.

In this case, big has a sense of being older, or grown up.
Hence, some senses of big and large are (nearly) synonymous while other ones are not.
Synonyms cannot be substituted with each other at all places, some are preferred over others depending upon the context. Another issue.
Another example: what is the cheapest first class fare vs what is the cheapest first class price (fare is preferred)

Antonomy: Two words with opposite meaning. It can be binary opposition/at opposite ends of same scale (fast/slow, big/little), or opposite in direction (reversives, e.g. rise/fall, up/down). Otherwise they are very similar to synonyms, sharing all other aspects, hence sometimes difficult to distinguish between the two. (syn and ant)

Taxonomic Relations: (taxonomy: class-subclass)
Hyponym: a word is hyponym of another if it’s more specific or subclass of the other.
e.g. car is a hyponym of vehicle; mango of fruit
Hypernym: converse of hyponym; fruit is hypernym of mango
To not to confuse the two terms, hypernym is better called as superordinate, and hypo as subordinate.
Formally, class denoted by the superordinate extensionally includes the class denoted by the hyponym. OR
A sense A is a hyponym of a sense B if everything that is A is also B, and hence being an A entails being a B, or $\forall x\, A(x) \Rightarrow B(x)$.
Hypernym/Hyponym is a transitive relation, also called IS-A relation, it can be called that B subsumes A.

Meronymy: part-whole relation; wheel is a part, a meronym of a car; car is a holonym of wheel (an asymmetric transitive relation between senses)

<u>WordNet</u>
A hierarchically organized lexical web database and most commonly used resource for sense relations. It consists of three databases: for nouns, verbs, and one for adjectives and adverbs. It doesn't include closed class words.
WordNet provides a list of synonyms and gloss for each sense, but not their pronunciation.

The set of synonyms for a WordNet **sense** is called a synset.
Example: *1. bass1 - (the lowest part of the musical range)* (From WordNet)
$\{bass^1,\ deep^6\}$, is one of the synset entries for the word Bass.

Hence, each synset can be represented by one gloss. Glosses are properties of a synset. Hence they are synsets, not wordforms, lemmas, or individual senses, that participate in most of the lexical sense relations in WordNet. Synset is a fundamental unit of WordNet entries.

In wordnet, each synset is also labelled with a lexicographical category drawn from their semantics. There are 26 categories for nouns, 15 for verbs, 2 for adjectives and 1 for adverbs. These categories are called supersenses because they act as coarse semantic categories/groupings of senses which can be useful when word senses are too fine grained.

Category	Example	Category	Example	Category	Example
ACT	*service*	GROUP	*place*	PLANT	*tree*
ANIMAL	*dog*	LOCATION	*area*	POSSESSION	*price*
ARTIFACT	*car*	MOTIVE	*reason*	PROCESS	*process*
ATTRIBUTE	*quality*	NATURAL EVENT	*experience*	QUANTITY	*amount*
BODY	*hair*	NATURAL OBJECT	*flower*	RELATION	*portion*
COGNITION	*way*	OTHER	*stuff*	SHAPE	*square*
COMMUNICATION	*review*	PERSON	*people*	STATE	*pain*
FEELING	*discomfort*	PHENOMENON	*result*	SUBSTANCE	*oil*
FOOD	*food*			TIME	*day*

Figure 18.2 Supersenses: 26 lexicographic categories for nouns in WordNet.

WordNet represents all kinds of sense relations (hypernymy, synonymy etc.). Here synsets are related.

Relation	Also Called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	*breakfast*^{1} $\rightarrow$ *meal*^{1}
Hyponym	Subordinate	From concepts to subtypes	*meal*^{1} $\rightarrow$ *lunch*^{1}
Instance Hypernym	Instance	From instances to their concepts	*Austen*^{1} $\rightarrow$ *author*^{1}
Instance Hyponym	Has-Instance	From concepts to their instances	*composer*^{1} $\rightarrow$ *Bach*^{1}
Part Meronym	Has-Part	From wholes to parts	*table*^{2} $\rightarrow$ *leg*^{3}
Part Holonym	Part-Of	From parts to wholes	*course*^{7} $\rightarrow$ *meal*^{1}
Antonym		Semantic opposition between lemmas	*leader*^{1} $\Longleftrightarrow$ *follower*^{1}
Derivation		Lemmas w/same morphological root	*destruction*^{1} $\Longleftrightarrow$ *destroy*^{1}

Figure 18.3 Some of the noun relations in WordNet.

Relation	Definition	Example
Hypernym	From events to superordinate events	*fly*^{9} $\rightarrow$ *travel*^{5}
Troponym	From events to subordinate event	*walk*^{1} $\rightarrow$ *stroll*^{1}
Entails	From verbs (events) to the verbs (events) they entail	*snore*^{1} $\rightarrow$ *sleep*^{1}
Antonym	Semantic opposition between lemmas	*increase*^{1} $\Longleftrightarrow$ *decrease*^{1}

Figure 18.4 Some verb relations in WordNet.

When we said that wordnet is hierarchically organized, we meant that with the help of hypernym/homonym relations we can traverse a hierarchy from the root sense to a very broad concept. Below figure shows an example.

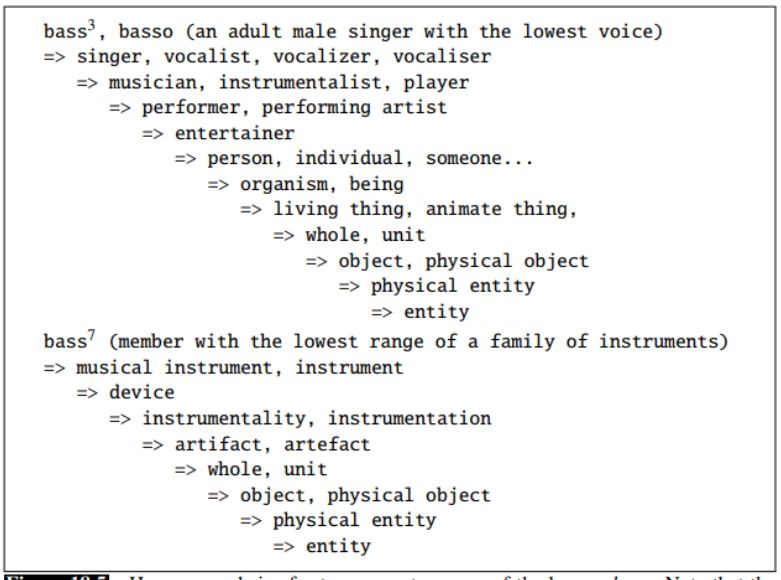


Figure 18.5 Hyponymy chains for two separate senses of the lemma *bass*. Note that the chains are completely distinct, only converging at the very abstract level *whole, unit*.

Now talking about synonymy, it is a binary relation. Two words can either be synonyms or not, they can either be part of the same synset or not. But what if we want to know how synonymous they are., in short, the word similarity. (Actually sense similarity, NOT words) This can be done by checking if they have similar hypernym (how many relations are common) or if their glosses are same, etc.

Word Similarity

There are two classes of algorithms to find this:

- Distributional: by comparing words based on their distributional context/pattern in the corpora
- Thesaurus based: based on whether words are nearby in wordnet hierarchy.

Thesaurus Based:

To find word similarity, first we find the similarity between various pairs of senses (in the pair, first sense from first word, and the second sense from second word), and then take the maximum of it, to represent the similarity of the two words.

This sense similarity can be calculated in terms of path length in the hierarchy. (number of edges between the two senses in the shortest path in the hierarchy)
Similarity = 1/ (1+pathlength)

$$sim_{path}(c_1, c_2) = \frac{1}{1+pathlen(c_1, c_2)}$$

$$sim(w_1, w_2) = max_{c_1 \in senses(w_1), c_2 \in senses(w_2)} sim(c_1, c_2)$$

Another measure of sense similarity in terms of path length, LC similarity:

$$sim_{LC}(c_1, c_2) = -log(pathlen(c_1, c_2)/2d)$$

d: maximum depth of the hierarchy

The problem with the previous measures is that the measure depended only on the number of edges between them, irrespective of where they occur in the hierarchy.

But as we move down the hierarchy, we are moving to very specific concepts, hence the same path length should result in more similarity as compared to that at a higher level of hierarchy. For that we use the concept probability model. In this, each occurrence of the word increments the count of all its hypernyms (hence all the parents). Finally convert them to probabilities by dividing by the root count. Then each concept probability is mapped to the information content it contains. More the probability of a sense, less the information it contains, and vice versa. We use -log() function for this mapping. Now with this we can find the similarity between two words/senses by seeing the information content of the lowest common ancestor **(subsumer/hypernym)** of them. This is called resnik similarity. This similarity consider how much information two senses share.

The similarity is now changing according to the depth. But there's still one problem, similarity of any two nodes with the lowest common ancestor will have the same similarity. Resnik similarity considers what two senses share in common but not what they don't share. Hence, we have another similarity measure: Lin Similarity.

Lin Similarity:
Resnik says the more information they share, the more similar they are.
Lin says the more information they don't share, the less similar they are.

$$sim_{Lin}(c_1,c_2) = \frac{2logP(LCS(c_1,c_2))}{logP(c_1)+logP(c_2)}$$

The information content common to c_1 and c_2, normalized by their average information content.

There are some variations.
JC Similarity: assigns all the edges a weight and calculates the similarity based on those edge weights rather than node weights (Info content associated with them)
These edges are simply IC(c1)-IC(c2) for the edge between c1 and c2.
Adding the weighted edge lengths: IC(LCS(c1,c2)) - IC(c1) + IC(LCS(c1,c2)) - IC(c2)
And finally invert them.

$$dist_{JC}(c,hypernym(c)) = IC(c) - IC(hypernym(c))$$

$$\begin{aligned} dist_{JC}(c_1,c_2) &= dist_{JC}(c_1,LCS(c_1,c_2)) + dist_{JC}(c_2,LCS(c_1,c_2)) \\ &= IC(c_1) - IC(LCS(c_1,c_2)) + IC(c_2) - IC(LCS(c_1,c_2)) \\ &= IC(c_1) + IC(c_2) - 2 \times IC(LCS(c_1,c_2)) \end{aligned}$$

$$sim_{JC}(c_1,c_2) = \frac{1}{IC(c_1)+IC(c_2)-2\times IC(LCS(c_1,c_2))}$$

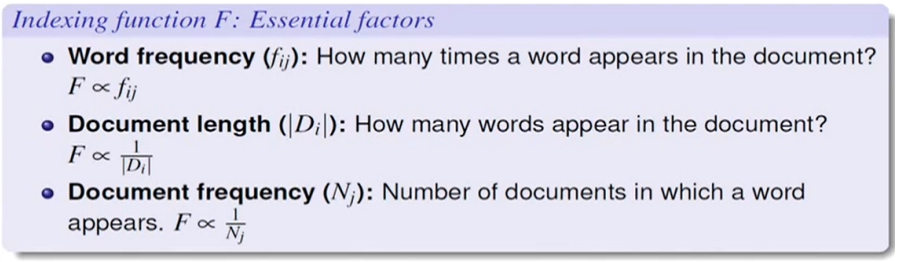
Another approach (not metric) for computing similarity between senses which is not based on Hierarchy but Glosses: Lesk Algorithm.
It states that two concepts are similar if their glosses contain similar words. Whenever an N-gram is common between the two glosses, a score of N^2 is added to the similarity.

These methods of finding similarity are fairly simple given the wordnet. But the problem is that they find the similarity between the senses, not the words. And what we are given in a natural language are words. We can approximate the sense similarities to words but this doesn't always work. Hence to be able to use wordnet, the problem to solve is to know the sense of a word.

Distributional Methods
You know a word by the company it keeps!
If we use one-hot representation for the words in vocabulary (size V), it will be a vector of length V with each word Wi having a 1 at index i and all else 0. With such a representation though, we can't find the similarity (ANDing them or their dot product gives 0).
Hence we move to the distributional representation, where for each word we add the counts of related words. Example, for banking, the vector contains 2 for government, etc.
In the corpus, select the target and the context, count the target-context co-occurrences, (weigh them, optional), build the distributional matrix, (then reduce the matrix dimensions, optional) and compute the vector distances on the matrix.
Example: when words are targets and documents are context (word*document) ← In data mining…
Documents as context is easier, find the occurrences in documents.
When words are context, we need to define a window size in which we will be looking for co-occurences. We also choose the window shape-> rectangular (uniform weightage with distance), or triangular (weightage to words decreases with distance) etc.

Next, how to weigh the context. (in case when documents are context)
Firstly the weight should be directly proportional to the term frequency/occurrence of words. But in case of documents the term frequency would be proportional to the size. Hence weight should be inversely proportional to the size of the document.

Next, if a word occurs in 50 documents vs a word that occurs in 3 documents, the first one is considered as generic and given less weight. Hence, weight is inversely proportional to the number of documents the word occurs in. This is also called the inverse document frequency.



….

<u>Word Sense Disambiguation</u>
Given a sentence, find the correct sense of the ambiguous words, from the wordnet. This is done by looking at the context of the word's use.

Algorithms:
1. Knowledge Based Approaches
   - Overlap Based Approaches
2. ML based approaches
   - Supervised
   - Semi supervised
   - unsupervised
3. Hybrid

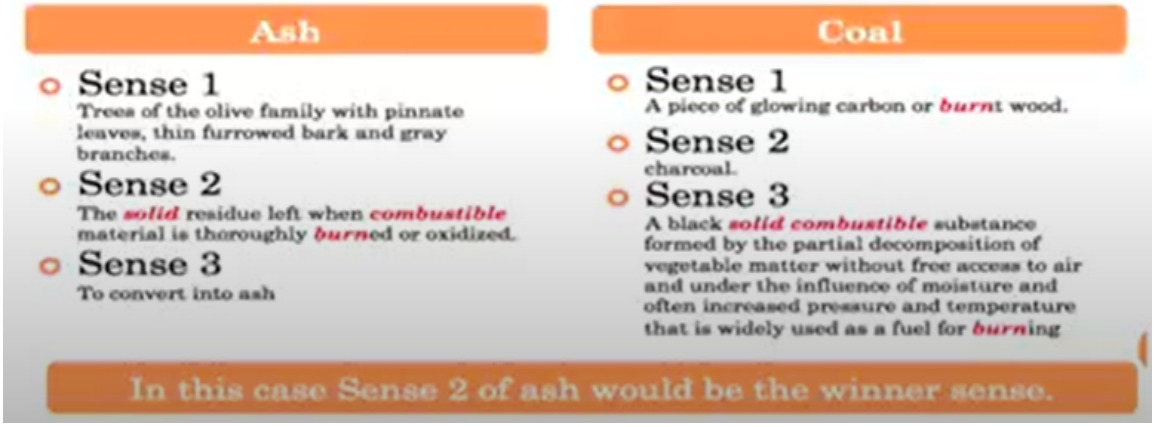
Knowledge Based Approaches
(Overlap based approaches)
- Require a machine readable dictionary (MRD)
- Find the overlap between the features of different senses of an ambiguous word (This set of features for a particular sense is called sense bag) and the features of

the words in its context (context bag). Choose the sense whose sense bag has the maximum overlap with the context bag.

- Features can be definitions, example sentences, hypernyms etc.

Example: using **Lesk's algorithm**

1. Sense bag is created and contains the words in the definition of the sense of that ambiguous world
2. Context bag is created and contains the words in the definition of each of the senses of each context word.
   e.g. On burning coal, we get **ash**. (ambiguous word is ash; coal is the context word we are using here)
3. The overlap is found using the lesk algorithm (adding N^2 for each common n-gram)
   Only coal has been considered as the context word here.



Sense 2 wins

Another approach (Walker's algorithm)

Thesaurus based approach

Each sense belongs to a thesaurus category. (Finance/Location etc)

Algorithm

1. Find the thesaurus category for each sense of the ambiguous word.
2. Calculate the score of each sense by using the context words.
   If the category of a sense and a context word matches, 1 is added to the score of that sense.
   e.g. The money in this *bank* fetches an interest of 8% per annum.
   Bank here has two senses: location and finance.

Context word ↓ / Sense →	Finance	Location
money	+1	0
fetches	0	0
interest	+1	0
annum	+1	0
TOTAL	3	0

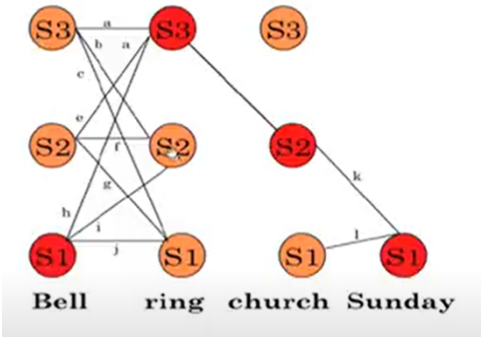
Here we are disambiguating a single word at a time, and others' multiple senses would be being considered. (disambiguating the sense of a particular word but not disambiguating the sense of other words; for example if money would have different senses too)

How to disambiguate all the ambiguous words present in the sentence together?
There would be some connection between the actual senses of each ambiguous word in the sentence. We need to find that connection.
Hence, we use a graph based approach.

Random Walk Algorithm

Example: the church bells no longer rung on sundays.
Church, bells and ring have 3 senses each, and sunday has 1.

1. Draw the corresponding graph.
2. Add edge weights using lesk algorithm

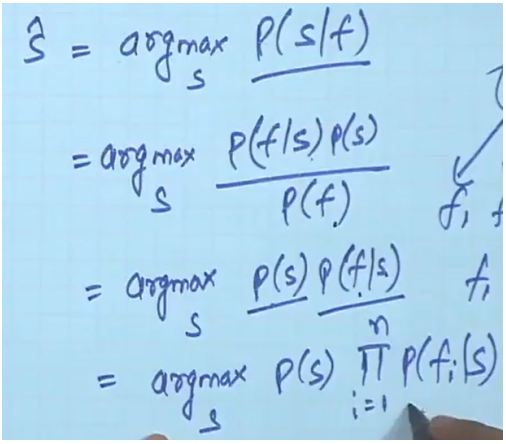


3. Find the best path (one node in each column, and the path should have maximum similarity), the overall combination whose similarity is the highest. A pagerank algorithm is employed for this. A pagerank score is found for each node. Start from an initial random probability distribution for all nodes, then iterate till convergence.

ML Based

Supervised

Naive Bayes: classification task: find the appropriate sense for the ambiguous words given the features of the sentence (of the context).



The naive bayes assumption: features are independent of each other.

$$\hat{s} = \arg\max_{s \in S} P(s) \prod_{j=1}^{n} P(f_j|s)$$

fi comes from the context. This can involve PoS tags of context words, co-occurrence vector etc.

$$P(s_i) = \frac{count(s_i, w_j)}{count(w_j)}$$

$$P(f_j|s_i) = \frac{count(f_j, s_i)}{count(s_i)}$$

Decision List Algorithm

- ‘One sense per collocation’ property (collocation: nearby words)
  E.g. ‘cricket bat’ suggests one sense of the bat only.
  Check P(sense1|collocation) and P(sense2|collocation)
  A collocation is good if one probability is significantly higher than the other.
  Then the log likelihood ratio is calculated.

$$log(\frac{P(Sense-A|Collocation_i)}{P(Sense-B|Collocation_i)})$$

The more it is, the more predictive evidence is provided by that collocation.
Collocations are ordered in a decision list with most predictive collocations ranked highest. (the absolute values, the sign will represent which sense to be assumed)

Resultant Decision List

Final decision list for *plant* (abbreviated)

LogL	Collocation	Sense
10.12	*plant* growth	⇒ A
9.68	car (within ±*k* words)	⇒ B
9.64	*plant* height	⇒ A
9.61	union (within ±*k* words)	⇒ B
9.54	equipment (within ±*k* words)	⇒ B
9.51	assembly *plant*	⇒ B
9.50	nuclear *plant*	⇒ B
9.31	flower (within ±*k* words)	⇒ A
9.24	job (within ±*k* words)	⇒ B
9.03	fruit (within ±*k* words)	⇒ A
9.02	*plant* species	⇒ A
...	...	

Now this table can be used as a decision tree classifier. First check for collocation plant growth if present or not, if yes assign sense A else move ahead to the next collocation.

Smoothing required in case denominator probability is zero.

Semantic Roles

A level of representation that captures the commonalities between different forms of a sentence or different sentences. Semantic Roles/Thematic Roles express the role that the argument of the predicate takes in the event. Deep roles are specific to each event.

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
CONTENT	The proposition or content of a propositional event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

Figure 19.1 Some commonly used thematic roles with their definitions.

Thematic Role	Example
AGENT	*The waiter* spilled the soup.
EXPERIENCER	*John* has a headache.
FORCE	*The wind* blows debris from the mall into our yards.
THEME	Only after Benjamin Franklin broke *the ice*...
RESULT	The city built a *regulation-size baseball diamond*...
CONTENT	Mona asked *“You met Mary Ann at a supermarket?”*
INSTRUMENT	He poached catfish, stunning them *with a shocking device*...
BENEFICIARY	Whenever Ann Callahan makes hotel reservations *for her boss*...
SOURCE	I flew in *from Boston*.
GOAL	I drove *to Portland*.

Figure 19.2 Some prototypical examples of various thematic roles.

Selection Restriction
A selectional restriction is a semantic type constraint that a verb imposes on the kind of concepts that are allowed to fill its argument roles.
Example: I want to eat someplace nearby. Vs I want to eat Indian food.
We reject the first sentence: the THEME of EATING events tends to be something that is edible. This restriction placed by the verb 'eat' on the filler of its THEME argument is a selectional restriction.
Selection restriction is associated with senses and not entire lexemes.

Selectional restrictions vary widely in their specificity.
The verb imagine imposes strict requirements on its AGENT role (restricting it to humans and other animate entities) but places very few semantic requirements on its THEME role.
A verb like diagonalize, on the other hand, places a very specific constraint on the filler of its THEME role: it has to be a matrix.
The arguments of the adjectives odourless are restricted to concepts that could possess an odour.

How to impose these restrictions?
In first order logic, just AND the restriction.
Example: FOL for eat verb: ∃ e, x, y Eating(e) ∧ Agent(e, x) ∧ Theme(e, y)
Imposing restriction: ∃ e, x, y Eating(e) ∧ Agent(e, x) ∧ T heme(e, y) ∧ EdibleThing(y)

But using FOL for imposing restrictions isn't preferred.
- It requires high computational cost.
- It assumes a large logical knowledge base of concepts that make up selection restrictions.

Hence, rather than representing the restrictions in terms of logical concepts, wordnet synsets are preferred. Each predicate simply specifies a WordNet synset as the selectional restriction on each of its arguments. The filler should be a hyponym of that synset.
Example: 'ate a hamburger', here the THEME role for eat verb can be assigned the synset {food, nutrient}. Hamburger is a hyponym of this synset and hence valid.

It was later realized that the selection restrictions should be better considered as preferences rather than strict constraints. Because for example, restriction violations often occur in theme role for eat. E.g. people realized you can't eat gold for lunch if you're hungry.

For this, there's a selection association model by Resnik. Resnik defines the idea of selectional preference strength as the general amount of information that a predicate tells us about the semantic class of its arguments. Example: eat tells a lot about edible things as theme.
The selection preference strength is defined by difference of: probability of the direct object falling into class c: P(c), and probability of the direct object of the specific verb falling into the class c: P(c|v). The greater the difference, the more information the verb provides. The difference can be quantified by relative entropy or KL divergence (D(P||Q)).

$$D(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

$$S_R(v) = D(P(c|v)||P(c)) = \sum_c P(c|v) \log \frac{P(c|v)}{P(c)}$$

….