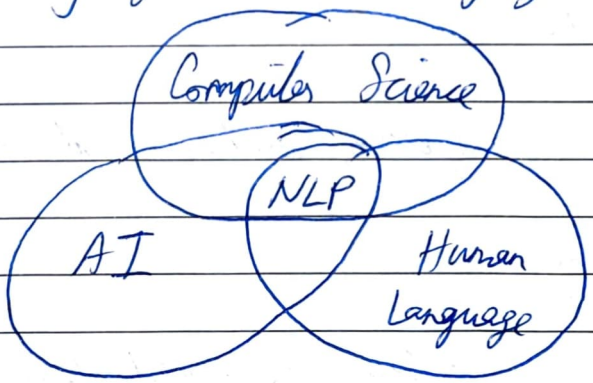
NLP

- Study of human languages and how they can be represented computationally and analyzed and generated algorithmically

Ex- The cat is on the mat ⇒ on(mat, cat)

- aka Computational Linguistics (CL), Human Language Technology (HLT), Natural Language Engineering (NLE), Speech and Text Processing.

- Natural Language Processing (NLP) is the branch of AI that gives machines the ability to read, understand and derive meaning from human languages.



- **Applications**

- Machine Translation between two different languages
- Information Retrieval (Web Search)
- Report Generation, Grammar & Spell Checking.
- Query Answering / Dialogues (ChatGPT)

__/__/__

- Parts of Speech Tagging, Sentiment Analysis

★ NLP Pipeline

Text → Segmentation → Tokenization → Stemming

Parsed Text ← Named Entity Recognition ← Parts of Speech Tagging ← Lemmatization

- Segmentation

- Process of dividing a sentence/paragraph into component sentences, along punctuation marks.

- Tokenization

- Process of splitting sentences into their constituent words

- Stemming

- Process of obtain the word stem/root of a word. using which new words can be formed upon adding affixes

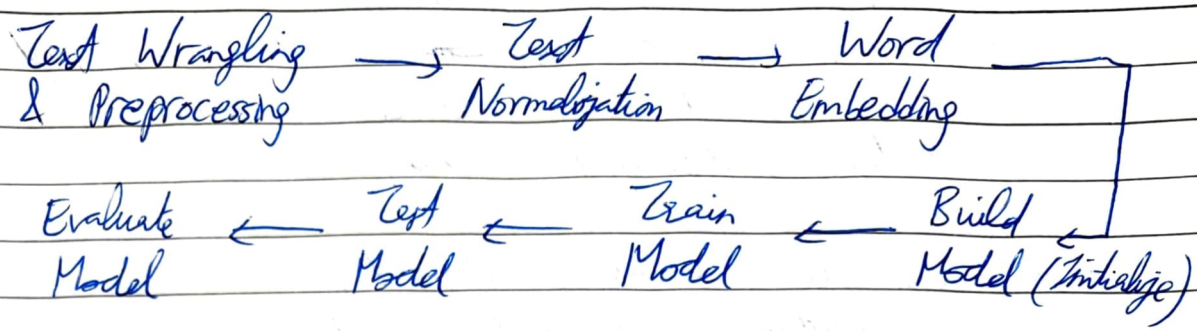
- Lemmatization

- Process of obtaining root stem of a word, atleast one that has meaning.

- Parts of Speech Tagging

- Tags a word as a noun, adjective, verb, etc.

__/__/__



Forms of Natural Language

- Input/output of an NLP system can be written text (mostly) or speech
- To process written text, we need lexical, syntactic and semantic knowledge about the language and discourse information, real world knowledge.

Knowledge of Language

- Phonology: Concerns how words are related to the sounds that realize them.

Ex: tap (clear t sound), butter (flap sound)

- Morphology: Concerns how words are constructed from basic meaning units (morphems)

Ex: child, children

- Syntax: Concerns how words can be put together to form correct sentences and what structural role each word plays in a sentence or what phrases are subparts of other phrases

Ex: Im sorry Dave, I'm afraid I can't do that

__/__/__

- Semantics: Concerns what words mean and how when combined form sentence meaning. The study of context-independent meaning.

Ex: Lexical (meaning of all words)
Compositional (knowledge about relationship of words)

- Pragmatics: Concerns how sentences are used in different situations and how they affect the interpretation.

Ex: Request (open the door)
Statement (door is open)
Information Question (is the door open?)

- Discourse: Concerns how preceding sentences affect the interpretation of the next sentence.

Ex: How many students graduated that year
Here, "that year" may be when covid-19 hit or when first batch graduate.

- World Knowledge: Every language uses must know about the other's beliefs and goals.

~~Ex~~

Natural Language Generation

- Producing output in the natural language from some internal representation, through deep planning and then syntactic generation.

__/__/____

- A sentence / word / phrase is said to ambiguous when it's meaning cannot be interpreted to certain exactness (multiple meanings) thus creating problems

- Ambiguity can be at different levels:

① Lexical (different meaning of words)
② Syntactic (different ways to ~~pase~~ parse / understand sentence)
③ Contextual info (context may affect meaning)
④ Interpreting partial info (pronouns)

- Ambiguity can be resolved through:

① Parts of Speech Tagging (verb / noun)
② Word-Sense Disambiguation (meaning that make sense)
③ Lexical Disambiguation (① + ②)
④ Syntactic ambiguity (by probabilistic parsing)

— • Linguistic Knowledge Representation

- State Machines: FSA, FST (morphological analysis), HMM (Hidden Markov Model) (probabilistic models used for POS tagging)
ATN, RTN

- Formal Rule Systems: CFG (define rules for constructing valid sentences), Probabilistic CFG (to resolve syntactic ambiguity)

- Logic-based Formalisms: First-order predicate logic (captures relationships), higher order logic (semantics)

- Models of Uncertainty: Bayesian ~~probability~~ probability theory

__/__/__

- Algorithms used to manipulate linguistic knowledge include transducers, parsers, state space search, dynamic programming.

☆ Morphology

- Study of the way words are built up from smaller meaning-bearing units (morphemes)

Ex: fox (1 morpheme fox)
cats (cat, cats (morpheme -s))

- Stems are the main morpheme of the word supplying main meaning, while affixes add additional meaning.

- Affixes include prefixes (undo → un-do)
  suffixes (playing → play-ing)
  infixes (abso-*ucking-lutely)
  circumfixes (used in goofy ahh languages)

- Some ways to form words from morphemes:

① Inflection (stem + grammatical morpheme) (same class)

Ex: Verb tense (walk → walked)
Noun count (dog → dogs)
Subject identity ("I run → he runs)

② Derivation (stem + grammatical morpheme) (diff class) (diff meaning)
(nominalization)

Ex: happy → unhappy (negation)
beauty → beautiful (noun → adjective)
teach → teacher (verb → noun)

Finite State Morphological Parsing

- Converting surface word forms into structured representations that include the stem and the morphological features

Ex:-
- cats → cat + N + PL (noun, plural)
- cat → cat + N + SG
- geese → goose + N + PL
- goose → (goose + N + SG) or (goose + V)
- gooses → goose + V + 3SG (third person singular)
- merging → merge + V + PRES-PART
- caught → catch + V + PAST-PART

① Lexicon: List of stems and affixes, with basic info about them (noun stem or verb stem)

② Morphotactics: Rules that determine how morphemes can be combined to form valid words.

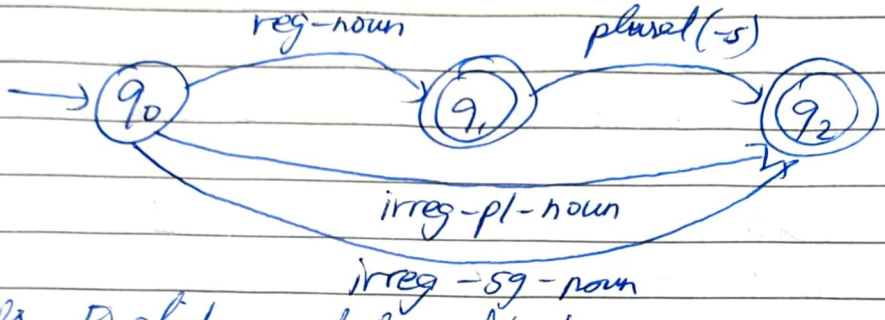
③ Orthographic Rules: Spelling rules used to model the changes that occur in a word, usually when two morphemes combine (city + -s → cities)

- Most common way of modelling morphotactics is finite state automaton.

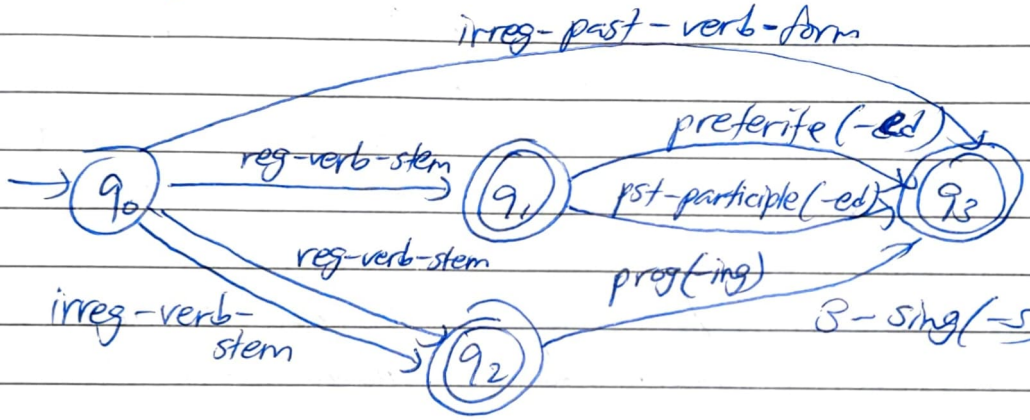
- Regular nouns follow a simple, predictable pattern when forming plural (cat → cats
  box → boxes)
- Irregular nouns break the rules, their plural forms change in unique ways (man → men
  mouse → mice
  child → children)

__/__/__

- FSA for English nominal inflection:



- FSA for English verbal inflection:



- Preterite indicates that action was completed at a specific time in the past. (I walked to the store)

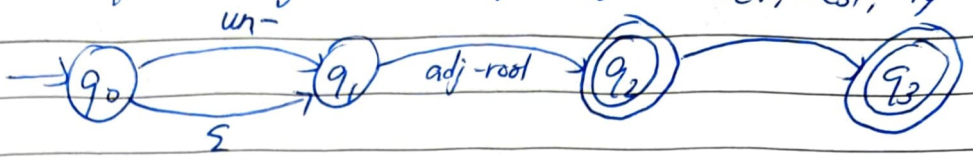
- Past-participle, on the other hand, used in perfect tenses and passive voice (uses has/have/be/been)

Ex:

	Stem	Past	Past-Participle	Present-Participle	3-sing
Regular =>	walk	walked	walked	walking	walks
	fry	fried	fried	frying	fries
Irregular =>	speak	spoke	spoken	speaking	speaks
	eat	ate	eaten	eating	eats
	sing	sang	sung	singing	sings

__/__/__

- FSA for English adjective morphology:



Ex: big, bigger, biggest
unhappy, unhappies, unhappiest, unhappily
clear, clearer, clearest, clearly, unclear, unclearly

Lexical: |c|a|t|+N|+PL| (simple concat of morphemes)

Surface: |c|a|t|s| (actual spelling of final word)

Note: Word → standalone unit made of morphemes (books, unhappy)
Morpheme → Smallest unit of meaning (book, run → free
-s, un- → bound)
Lexicon → collection of all known words and morphemes in a language.

- Morphological parsing involves mapping rules that map letter sequences on the surface level to morpheme and feature sequence on the lexical level, using finite state transducer (FST)

- While FSA defines a formal language, FST defines relation between sets of strings.

- FST, as a recognizer, accepts / rejects pairs of strings by checking if input-output pair is valid (run → run)
(pretty much like FSA)
- FST, as a generator, produces outputs strings from input strings (morphological generation) (walk → walked)

- FST as a transducer, transforms one string into another (cats → cat + N + PL)
where it segments and tags the word morphologically.

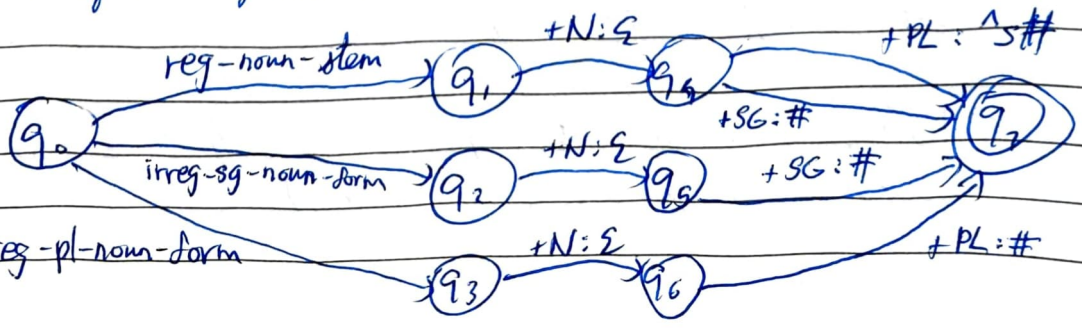
- FST as a Set relates, defines relation between two sets of strings

- Q : finite set of N states.
Σ : finite set of corresponding to input alphabet
Δ : finite set of corresponding to output alphabet
$q_0 \in Q$: start state
$F \subseteq Q$: final states set
$\delta(q,w)$: transition function $(Q \times \Sigma^* \rightarrow 2^Q)$
$\sigma(q,w)$: output function giving set of possible output strings for each state and input
$Q \times \Sigma^* \rightarrow 2^{\Delta^*}$

- Inversion: If T maps from $I \rightarrow O$
T^{-1} maps from $O \rightarrow I$
- Composition: If $T_1 : I_1 \rightarrow O_1$
$T_2 : I_2 \rightarrow O_2$
$T_1 \circ T_2 : I_1 \rightarrow O_2$

- ^ → morpheme boundary
→ word boundary.

- FST for English nominal number inflection T_{num} :



→ Orthographic Rules

~~• Constant doubling~~

① • Consonant doubling (1-letter consonant doubled before -ing/-ed)
(beg → begging)

② E deletion (silent e dropped before -ing/-ed)
(make → making)

③ E insertion (e added after -s, -z, -x, -ch, -sh, before -s)
(watch → watches)

④ Y replacement (-y changes to -ie before -s, -i before -ed)
(try → tries)

⑤ K-insertion (add -k for verbs ending with vowel + -c)
(panic → panicked)

• These rules are applied in the intermediate level before surface.

• Parsing can be slightly more complicated than generation due to ambiguity (for example, foxes is either fox + V + 3Sg (or) fox + N + PL)

• Since the transducer is not capable of deciding, disambiguating requires some external evidence such as ~~external~~ surrounding words

Ex: I saw two foxes yesterday (noun)
That trickster foxes me every time! (verb)

• Intersection is an operation that combines two FSTs such that resulting FST only accepts I/O pairs accepted by both original FSTs

• Useful for combining constraints from diff linguistic levels (phonology + morphology)

★ Porter Stemmer (Lexicon-free FST)

- Based on series of cascaded rewrite rules, uses suffix-stripping rules without reading lexicon lookup

- Error of commission: When stemming algorithm incorrectly modifies a word, resulting in a stem that does not accurately represent the word's root form

Ex: university → univers

- Error of omission: When stemming algorithm fails to reduce a word to it's root form

Ex: runner → runner

★ Spelling Error Detection & Correction

① Non-Word Error Detection

- Detecting words that don't exist in the language by lexicon lookup or edit distance.

Ex: graffe

② Isolated-Word Error Detection

- Correcting non-words without context, based on word itself

Ex: graffe → giraffe

③ Context-dependent Error Detection + Correction

- Correcting real-word errors using surrounding context

Ex: I ate desert → I ate dessert

__/__/__

Minimum Edit Distance

- Minimum number of editing operations (insertion, deletion, substitution) needed to transform one into the other.

- For Levenshtein minimum edit distance, insertion, deletion and substitution have a cost of 1. On the other hand, for weighted minimum edit distance, substitution costs 2.

Ex: I N T E * N T I O N
D S S I I S I I I I — Cost = 5 (Lev)
* E X E C U T I O N — = 7 (weighted)

- Searching for a path (sequence of edits) from start string to final string.
  Initial state: Word we are transforming
  Operators: Insert, delete, substitute
  Goal state: Word we are trying to get to
  Path cost: No. of edits we want to minimize.

- For two strings X of length n, Y of length m. we define D(i,j) as the edit distance of first i characters of X and first j characters of Y. Hence edit distance between X and Y is D(n,m)

- Using bottom-up dynamic programming approach, we compute D(i,j) for small i,j and compute D(i,j) for all i ($0<i<n$) and j ($0<j<m$) based on previously computed smaller values.

→ $D(i,0) = i$
$D(0,j) = j$ ~~$D(0,j) = 0$~~

For each $i = 1 \ldots M$
  For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + 2 \text{ if } X(i) \neq Y(j) \\ D(i-1,j-1) + 0 \text{ if } X(i) = Y(j) \end{cases}$$

$D(N,M)$ is distance

Ex: K I T T E N → S I T T I N G Cost = 3

	N	6	6	5	4	3	3	2	3
E	5	5	5	4	3	2	2	3	4
T	4	4	4	3	2	1	2	3	4
T	3	3	3	2	1	2	3	4	5
I	2	2	2	1	2	3	4	5	6
K	1	1	2	2	3	4	5	6	7
#	0	1	2	3	4	5	6	7	
	#	S	I	T	T	I	N	G	

- So basically set the backtag columns with numbers from 0. Next check if selected cell has matching row and column characters. If yes, just set it as the South West value. If not, find the smallest number in the surrounding L values and add 1. Repeat this until we reach the top-right (result)

__/__/__

☆ Regular Expressions Usecases

– Extracting Word Pieces

- re.findall() methods finds all matches of given RE.

Ex: word = 'johann'
re.findall(r'[aeiou]', word)
→ ['o', 'a']

– Finding Word Stems

- For some language processing tasks like ~~web~~ searching, we want to ignore word endings and just deal with the word stems.

Ex: re.findall(r'^.*(ing|ly|ed|ious|ies|ive|es|s|ment)$', 'processing')
(Here ^ → start of string, . → ~~end~~ any character,
* → zero or more of preceding character)
$ → end of string)
→ ['ing']

– Searching Tokenized Text

- "<a> <man>" finds all instances of 'a man' in the text.
<> → used to mark token boundaries

- <.*> → used to match any single token
Enclose in parenthesis to only obtain matched words not matched phrases.

__/__/__

Ex: from nltk.corpus import gutenberg, nps_chat
moby = nltk.Text(gutenberg.words('melville-moby_dick.txt'))

moby.findall(r"<a>(<.*>)<man>")

→ nervous, dangerous, white, ...

chat = nltk.Text(nps_chat.words())
chat.findall(r"<l.*>{3,}")
↳ atleast 3 occurrences

→ lol lol lol, lmao lol lol lol (starting with l)

— Normalizing Text (text.lower())

- Case normalization is converting all text to lowercase or uppercase to standardize the text.
- Helps eliminate case sensitivity, making text data consistent and easier to process. However it can lead to loss of information (nouns)

— Punctuation Removal

- Process of remove special characters and punctuation marks

★ - Stopword Removal

- Process of removing common words with little meaning

Ex: The quick brown fox jumps over the lazy dog
quick brown fox jumps over lazy dog

__/__/__

Exa
```
from nltk.corpus import stopwords
text = " "
stop_words = set(stopwords.words("english"))
words = text.split()
filtered_words = [word for word in words if word not
                  in stop_words]
text = " ".join(filtered_words)
print(text)
```

— Stemming

Exa
```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
text = " "
words = text.split(" ")
stemmed_words = [stemmer.stem(word) for word in words]
text = " ".join(stemmed_words)
print(text)
```

— Lemmatization (WordNetLemmatizer)

Exa
```
lemmatized_words = [lemmatizer.lemmatize(word) for
                    word in words]
```

— Tokenization

- `re.split(r'[ |t|n]+', text)`
  ↳ one or more occurrences
- ```
  text = " "
  pattern = r"""(?x)
            ([A-Z]\.)+
          | \$?\d+(\.\d+)?%?
  """
  ```
  (enable verbose mode) (abbreviations) (currency/percentage)

```
tokens = nltk.regexp_tokenize(text, pattern)
```
```

Segmentation

- Dividing text into individual sentences.

Ex:

```
sent_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
sents = sent_tokenizer.tokenize(text)
```

N-GRAMS

- For a given sentence, we assign a probability to each possible next word in that sentence.
- This can be helpful for machine translation, spelling correction, speech recognition and summarization.
- Word prediction can be used to suggest likely words for the menu
- If a person is physically unable to speak, but can use eye gaze or other specific movements to select words from a menu to be spoken by the AAC system (Augmentative and Alternative Communication)

☆ Language Model

- A probabilistic statistical model that determines probability of given sequence of words occurring in a sentence based on previous words.
- (I/P) → Training set of example sentences
  (O/P) → Probability Distribution over sequence of words
- $P(W)$ or $P(W_n / W_1, W_2, W_3, \ldots, W_{n-1})$
  (W → sentence)
- Transforms qualitative information about text into quantitative information that machines can understand.
- An n-gram is a sequence of n words

Ex- Bigrams: "please follow", "follow the", "the rules"

- Corpus is a significant collection of texts written in everyday language, used for training ML models, language understanding, etc.
- Utterances are spoken equivalent of sentences which often contains disfluencies like fragments (broken-off words) and fillers / filled pauses (uh, um, er). These are however, useful in speech recognition since they tell about natural pauses and hesitations.
- Lemma is a set of lexical forms have same stem, same major parts-of-speech and same word sense
- Wordforms include full inflected (same class) or derived form (diff class)
- Chain Rule implies that:

$$P(W_1, W_2, \ldots W_n) = P(W_1)\, P(W_2/W_1)\, P(W_3/W_1, W_2) \ldots \ldots P(W_n/W_1, W_2 \ldots W_{n-1})$$

$$P(A/B) = \frac{P(A \cap B)}{P(B)} \quad \text{(Bayes)}$$

$$P(A, B, C, D) = P(A)\, P(B/A)\, P(C/A, B)\, P(D/A, B, C)$$

- The intuition of N-gram model is that instead of computing next word based on its entire history, we can approximate the history by last few words

$$P(W_n / W_1^{n-1}) \simeq P(W_n / W_{n-1}) \quad \text{(bigrams)}$$

(whole sequence of (n-1) words) (immediate preceding word)

- In general, $P(W_n / W_1^{n-1}) \simeq P(W_n / W_{n-N+1}^{n-1})$

Markov Assumption

- Assumption that probability of a word depends only on the previous word, without looking too far into the past.

Maximum Likelihood Estimation

- To compute a particular bigram probability of a word W_n given a previous word W_{n-1}, we compute the count of the bigram $C(W_{n-1}, W_n)$ and normalize by sum of all bigrams that share same first word W_{n-1}

$$P(W_n / W_{n-1}) = \frac{C(W_{n-1} W_n)}{\sum C(W_{n-1})}$$

- For n-grams,

$$P(W_n / W_{n-N-1}^{n-1}) = \frac{C(W_{n-N+1}^{n-1} W_n)}{C(W_{n-N+1}^{n-1})}$$

Ex- (Corpus) <s> I am Sam </s> ← end tag
<s> Sam I am </s>
start tag → <s> I do not like green eggs and ham </s>

$$P(am / I) = \frac{Count(I\ am)}{Count(I)} = \frac{2}{3} = 0.67$$

Note - As no. of previous state (history) increases, it is very difficult to match the set of words in corpus. Probabilities of larger collection of word is minimum. To overcome this problem, bi-gram model is used.

__/__/____

Ex: Count(about five minutes from) = P(about / <s>) *
P(five / about) * P(minutes / five) *
P(from / minutes)

(Bi-gram) Count(about five minutes from college) = P(about / <s>)
* P(five / about) * P(minutes / five)
* P(from / minutes) * P(college / from)

P(college / about five minutes from) = P(about five minutes from college) / P(about five minutes from)

= P(college / from)

Ex: Count(about five minutes from) = P(five / <s>, about)
* P(minutes / about, five) * P(from / five minutes)

(Trigram) Count(about five minutes from college) = P(five / <s>, about)
* P(minutes / about, five) * P(from / five minutes)
* P(college / minutes, from)

P(college / about five minutes from) = P(college / minutes, from)

Ex: <s> I am Henry </s>
<s> I like college </s>
<s> Do Henry like college </s>
<s> Henry I am </s>
<s> Do I like Henry </s>
<s> Do I like college </s>
<s> I do like Henry </s>

If W_{i-1} = Henry in the sentence = <s> I like Henry ...

Word	Frequency
<s>	7
</s>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

$P(</s> / Henry) = \frac{C(Henry, </s>)}{C(Henry)} = \frac{3}{5}$ ✓

$P(<I> / Henry) = \frac{1}{5}$

$P(am / Henry) = 0$ $\quad P(college / Henry) = 0$

$P(Henry / Henry) = 0$

$P(like / Henry) = \frac{1}{5}$ $\quad P(do / Henry) = 0$

∴ Hence, $</s>$ is more probable after Henry.

(Which sentence is more probable) (using bigrams)

$<s>$ I like college $</s>$

$P(I / <s>) * P(like / I) * P(college / like) * P(</s> / college)$

$= \frac{3}{7} * \frac{3}{6} * \frac{3}{5} * \frac{3}{3} = 0.13$ ✓

$<s>$ Do I like Henry $</s>$

$P(Do / <s>) * P(I / Do) * P(like / I) * P(Henry / like)$

$* P(</s> / Henry) = \frac{3}{7} * \frac{2}{4} * \frac{3}{6} * \frac{2}{5} * \frac{3}{5}$

$= 0.0257$

Note To generate N-grams practically,

```
from textblob import TextBlob
text = "I am learning NLP"

TextBlob(text).ngrams(1)    # unigrams
TextBlob(text).ngrams(2)    # bigrams
```

__/__/__

★ Evaluation & Perplexity

- Assign higher probability to real and frequently-observed sentences than ungrammatical or rarely-observed sentences. (ideal model)

- Extrinsic Evaluation: Put each model in a wellness test (translation accuracy, spelling correction, etc.) and compare accuracies. However, it is time-consuming.

- Intrinsic Evaluation: Internal quality check (perplexity) to direct measure model prediction quality.

— Perplexity

- Measure of how well a probabilistic model predicts a given sequence of words / tokens (confidence)

- Lower perplexity value indicate better performance
Perplexity = 1 means model perfectly predicts test data.

$$PP = P(W_1^N)^{\frac{-1}{N}}$$

- Suppose : <s> I want food </s>
uses bi-gram model :

$$P(I/<s>) * P(want/I) * P(food/want) * P(</s>/food)$$
$$= 0.0001$$

N = length/size of vocabulary (excluding <s>)
$N = 4$ (I, want, food, </s>)

Here $PP = (10^{-4})^{\frac{-1}{4}} = 10$
(model has to choose among 10 equally likely words at each step)

- Generalization is the ability of a language model to make reasonable predictions on unseen / out-of-sample data.

- N-grams often works well for word predictions if test corpus looks like the training corpus.

Ex: If the model has seen the bigram "hot coffee" frequently, then it can generalize to predict "iced tea" even if it hasn't seen the specific combo in the training set before.

- For elements not present in training set but occur in test set, probability of test set = 0 (perplexity cannot be computed) (which is not ideal)

— Smoothing

- Technique used in N-gram LM's to address the problem of zero probabilities for n-grams not observed in training data.

- Add-1 estimation (Laplace smoothing)
(pretend we saw each word more than once than we did)

$$P^*_{\text{Add-1}}(w_i / w_{i-1}) = \frac{c(w_{i-1} w_i) + 1}{c(w_{i-1}) + V}$$

V ← vocabulary size

— Discounting

- Reserving some probability mass for unseen word sequences by lowering the probabilities of seen sequences

Original count = c
Discounted count = c^*

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

$(1 - d_c)$ probability mass distributed

- By doing so, we can reconstitute the counts in the count matrix

$$C^*(W_{n-1}W_n) = \frac{[C(W_{n-1}W_n)+1] * C(W_{n-1})}{C(W_{n-1}) + V}$$

Ex: For the previous example,

- <s> like college </s>

$P(\text{like}/\langle s\rangle) * P(\text{college}/\text{like}) * P(\langle/s\rangle/\text{college})$

$= \frac{0}{7} * \frac{3}{5} * \frac{3}{3} = 0$ (Zero probability)

- <s> Do I like Henry </s>

$P(\text{do}/\langle s\rangle) * P(\text{I}/\text{Do}) * P(\text{like}/\text{I}) * P(\text{Henry}/\text{like}) * P(\langle/s\rangle/\text{Henry}) = \frac{3}{7} * \frac{2}{4} * \frac{3}{6} * \frac{2}{5} * \frac{3}{5} =$

$= 0.0257$

(Even though the second statement seems more probable, let's try to smoothen the probabilities)

<s> like college </s> ($V = 7$ excluding <s>)

$P = \frac{0+1}{7+7} * \frac{3+1}{5+7} * \frac{3+1}{3+7} = 0.0095$

<s> Do I like Henry </s>

$P = \frac{3+1}{7+7} * \frac{2+1}{4+7} * \frac{3+1}{6+7} * \frac{2+1}{5+7} * \frac{3+1}{5+7} =$

$= 0.0020$

(After smoothing, first statement is now more probable)

Good Turing Algorithm

- Statistical method used to estimate the probability of unseen events in a sample.
- Based on computing N_c, number of N-grams that occur c times (frequency of frequency c)

Eg: N_0 → no. of bigrams with count 0 (never seen) (zero count)
N_1 → no. of bigrams with count 1

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$ (adjusted count)

For $c=0$, $c^* = \frac{1 \cdot N_1}{N_0}$ (probability mass of unseen n-grams from no. of singletons (seen once))

$$P^*(w_i/w_{i-1}) = \frac{c^*}{N}$$ → total no. of N-grams tokens in training corpus

- Unlike Laplace smoothing, where constant is simply added, Good-Turing redistributes the probability based on no. of rare events in data.

Eg: While fishing, we caught 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

$N_1 = 3$, $N_2 = 1$, $N_3 = 1$, $N_{10} = 1$

Without smoothing, $P(\text{trout}) = \frac{1}{18}$ (0.055) $P(\text{unseen}) = \frac{0}{18}$

$$P^*_{GT}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = \frac{1}{6} = 0.167$$

(N_0 = unknown / ∞ so ignore)

$$c^* = \frac{(c+1)N_{c+1}}{N_c} \qquad \text{(for fraud, } c = 1\text{)}$$

$$= \frac{2 \times 1}{3} = \frac{2}{3}$$

$$P^*_{GT}(\text{fraud}) = \frac{c^*}{N} = \frac{2}{3+27} = \frac{1}{27} = 0.037$$

∴ Good-Turing smoothes reduced fraud's probability from 0.055 → 0.037, for distributing to unseen events.

— **Backoff**

- Based on the idea that if a higher-order n-gram (sequence of n words) has low or zero probability you can back-off to a lower-order n-gram to get a non-zero probability estimate.

Trigram → Bigram → Unigram

— **Interpolation**

- Combine probabilities of all orders together (trigram + bigram + unigram)

$$\hat{P}(w_n \mid w_{n-2} w_{n-1}) = \lambda_1 P(w_n \mid w_{n-2} w_{n-1}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n)$$

where $(\lambda_1 + \lambda_2 + \lambda_3 = 1)$ → weights.

- While trigrams give more context, they may be unreliable if data is sparse.
Bi-gram gives less context, but more reliable if count is high
Unigram gives less information but never zero.

__/__/____

- Split data in training data (to compute raw n-gram probabilities), held-out data (to tune λ's) and test data (for final evaluation)
- Choose λ's that make held-out data more probable using some optimization technique.

PARTS OF SPEECH TAGGING

- Assigning a grammatical label (word class) to each word in a sentence, based on its syntactic role within that sentence.

- POS is a category of words that have similar grammatical properties.

- Provides valuable linguistic information about a text. Helps in understanding the structure and meaning of sentences.

(Cons) - Ambiguity: Many words can have multiple POS tags depending on context.

Ex: Lead → verb ("he leads the team")
Lead → noun ("pencil has a lead tip")

- Handling languages with free word order like in Latin or Finnish can be particularly challenging.

- Common POS categories include:
  - Noun (NN) (Names a person, place or thing)
  - Verbs (VB) (expresses an action/state of being)
  - Adjectives (A)(JJ) (describes a noun)
  - Adverbs (RB) (shows manner, time, degree)
  - Pronouns (PRON)(PRP) (replaces a noun, he/she)
  - Conjunctions (CONJ)(CC) (connects words, and, but)
  - PREPOSITIONS (PREP)(IN) (in, on, at, under)
  - Determiners (DET)(DT) (the, a, this, many)
  - INTERJECTIONS (INTJ)(UH) (Wow!! Oh!!) (sudden emotion)

- POS tagging can be performed using:

(a) Rule-based taggers (use dictionary/lexicon to tag each word)

(b) Statistical models (HMM, Maximum Entropy Markov Model)

(c) Deep Learning methods (RNN (recurrent neural networks), BERT (transformer-based))

- A large annotated corpus (collection of text with labelled POS) is needed to train a POS tagger, which would then learn the relationships between words and their corresponding POS tags.

(Applications)
- Information extraction (key info from text)
- Parsing (understanding syntactic structure of sentences)
- Sentiment analysis
- Machine translation (between languages)

- Open class words (nouns, verbs, adjectives, adverbs)
(Mostly content-bearing, refer to objects, actions and features in the world) (new words added all the time)

- Closed class words (pronouns, determiners, prepositions, conjunctions)
(Mostly functional, tie concepts of sentence together, fixed)
(limited in number)

→ - Morphological Forms (word changes by tense/aspect)

eat/VB (base) (I want to eat)
eat/VBP (present tense, plural subject) (They eat lunch)
eats/VBZ (present tense, 3rd person singular) (He eats an apple)

__/__/__

ate /VBD (past tense) (She ate yesterday)
eaten /VBN (past participle) (The apple was eaten)
eating /VBG (present participle) (I am eating)

Note: Articles can be definite (the) and indefinite (a, an)
Conjunctions can be coordinate (and, but) and subordinate (that, because, unless)
Pronouns can be personal (I, he, she), possessive (my, his) or wh (who, whom)
Auxillary verbs include be, do, have, can, will, shall.

- Penn Treebank contains a hand-annotated corpus of Wall Street Journal having 1M words, 45-46 tags.

Ex: The grand jury commented on a number of other topics
The /DT grand /JJ jury /NN commented /VBD on /IN a /DT number /NN of /IN other /JJ topics /NNS (plural)

- A tagger for English that simply chooses the most likely tag for each word can achieve good performance (unigram approach)

— Rule-Based Tagging

- Assign each word in the input a list of potential POS tags
- Then winnow down this list to a single tag using hand-written rules.

Ex: Sentence: I can book a ticket

can, could, may, might, should, will.
(attitude towards action)
(modal)

Baseline tagging: I/PRP can/MD book/NN a/DT ticket/NN

After rule application: Rule: Change NN → VB if previous word is MD (modal)
I/PRP can/MD book/VB a/DT ticket/NN

— Transformation-based Tagging

- Uses a set of rules learned from tagging training data to change tags that are applied to words inside a text, based on contextual information.

- Provides high accuracy when dealing with complex grammatical structures. However, to achieve this, it might require a large dataset and additional computer power.

★ Probabilistic Tagging

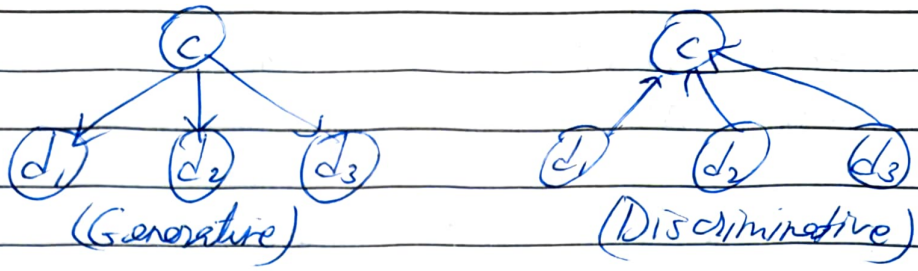
- We have some data $\{(d, c)\}$ of paired observations d and hidden classes c.

- POS tagging: Words are observed, tags are hidden
  Text classification: Sentences/documents are observed, category is hidden (positive/negative sentiment)

(i) • Generative Models: learn how data is produced

$P(d, c) = P(d/c) * P(c)$ (model both how class produces data and how common the class is) (joint probability)

Ex:- Naive Bayes Classifier,
HMM



(Generative) (Discriminative)

(ii) Discriminative / Conditional Models : Learn how to label data

$P(c/d)$ (Probability over hidden structure given the data)

Ex:- Logistic Regression, Maximum Entropy Markov Model

(i)(a) Naive-Bayes Approach

Given a sentence:

$W = w_1, w_2, w_3, \ldots, w_n$ (observed words)

$T = t_1, t_2, t_3, \ldots, t_n$ (hidden POS tags)

$$P(T/W) = \frac{P(W/T) * P(T)}{P(W)}$$ (same for all tag sequences)

$$\hat{T} = \text{argmax}\,(P(T/W))$$

$$\left(\hat{T} = \text{argmax}\, P(W/T) * P(T)\right)$$

(most probable tag sequence given the words)

Here $P(W/T)$ → likelihood (how likely is this sequence of words if these were the tags)

$P(T)$ → prior (how likely is this tag sequence)

Ex:- DT → NN is very common (the boy)

NN → VBZ is common (John eats)

VBZ → JJ might be rare.

Now since determining the full probability requires every word depending on all previous words and tags, and every tag depending on entire history, making it computationally expensive; HMM makes two assumptions:

(1) Each word depends ONLY on it's own tag

$$P(W/T) \approx \prod P(w_i / t_i)$$

(2) Tag depends only on previous tag (bigram assumption)

$$P(T) \approx \prod P(t_i / t_{i-1})$$

$$\therefore \hat{T} = \underset{T}{\operatorname{argmax}} \prod P(w_i / t_i) \times P(t_i / t_{i-1})$$

- Transitions Probalities: $P(t_i / t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$
- Word Likelihood: $P(w_i / t_i) = \frac{C(t_i, w_i)}{C(t_i)}$

— Hidden Markov Model

- Extension of finite automaton, which is defined by a set of states, and a set of transitions between states taken based on input observations.
- A weighted FSA is an augmentation of FSA where each arc is associated with a probability indicating how likely that path is to be taken.
- Probability of all arcs leaving that node must sum up to 1.

__/__/__

- Markov chain is a special case of weighted automata in which input sequence uniquely determines which states the automaton will go through, granted the sequence is unambiguous.

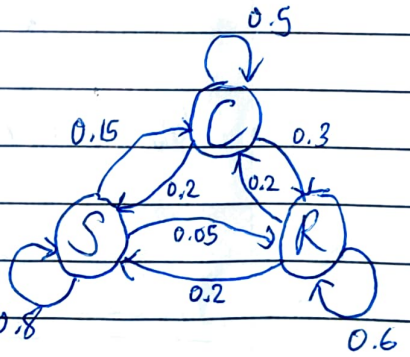
- HMM model comprises of a set of states (tags), an output alphabet (words), initial state (beginning of sentence), state transition probabilities $P(t_i / t_{i-1})$ and word likelihood (emission) probabilities $P(w_i / t_i)$

- In a Markov model, next state only depends on the present state, not on sequence of events preceding it.

Ex:- Sunny → Cloudy → Rainy → ? → ?

Tomorrow → / (Today) ↓	Sunny	Rainy	Cloudy
Sunny	0.8	0.05	0.15
Rainy	0.2	0.6	0.2
Cloudy	0.2	0.3	0.5

(Transition Probability Table)



Given that today weather is sunny, what is the probability that tomorrow is sunny and day after is rainy?

$$P(q_2 = \text{sunny}, q_3 = \text{rainy} / q_1 = \text{sunny})$$
$$= P(q_3 = \text{rainy} / q_2 = \text{sunny}) \cdot P(q_2 = \text{sunny} / q_1 = \text{sunny})$$
$$= 0.05 \times 0.8 = 0.04$$

- In Hidden Markov Model, state is only partially observable

- Since in POS tagging, words are visible but tags are

not (true states), simple Markov model fails here. However HMMs would work by letting us model POS tags as hidden states that generate the observed words, and it uses transition + emission probabilities to compute the most likely tag sequence even if tags are not visible.

Ex: Say we observe the sequence of weather outcomes:
Dry → Damp → Soggy
We want to predict the hidden sequence of weather states (Sunny, Rainy) that most likely caused this using HMM.

Hidden States ⇒ Sunny, Rainy
Observations ⇒ Dry, Damp, Soggy
Transition Probabilities (probability of moving from one state to another)

From / To	Sunny	Rainy
Sunny	0.8	0.2
Rainy	0.4	0.6

Emission Probabilities (probability of observing a certain outcome)

Weather	Dry	Damp	Soggy
Sunny	0.6	0.3	0.1
Rainy	0.1	0.4	0.5

Initial Probabilities (probability of weather on first day)

P(Sunny) = 0.7, P(Rainy) = 0.3

Dry → Damp → Soggy

① $P(Sunny) \times P(Dry/Sunny) = 0.7 \times 0.6 = 0.42$
$P(Rainy) \times P(Dry/Rainy) = 0.3 \times 0.1 = 0.03$

② $P(Sunny/Damp) = [P(Sunny)^{(before)} \times P(Sunny \to Sunny) + P(Rainy) \times P(Rainy \to Sunny)] \times P(Damp/Sunny)$
$= [(0.42 \times 0.8) + (0.03 \times 0.4)] \times 0.3 = 0.1068$

$P(Rainy/Damp) = [P(Sunny)^{(before)} \times P(Sunny \to Rainy) + P(Rainy) \times P(Rainy \to Rainy)] \times P(Damp/Rainy)$
$= [(0.42 \times 0.2) + (0.03 \times 0.6)] \times 0.4 = 0.0384$

③ $P(Sunny/Soggy) = [P(Sunny) \times P(Sunny \to Sunny) + P(Rainy) \times P(Rainy \to Sunny)] \times P(Soggy/Sunny)$
$= [(0.1068 \times 0.8) + (0.0384 \times 0.4)] \times 0.1$
$= 0.0098$

$P(Rainy/Soggy) = [P(Sunny) \times P(Sunny \to Rainy) + P(Rainy) \times P(Rainy \to Rainy)] \times P(Soggy/Rainy)$
$= [(0.1068 \times 0.2) + (0.0384 \times 0.6)] \times 0.5$
$= 0.0249$

∴ Sunny → Sunny → Rainy

__/__/__

Viterbi Algorithm

- Dynamic Programming algorithm used for finding the most likely sequence of hidden states (Viterbi Path) that results in a series of observed events.

- **Input:** State space $(S) = \{S_1, S_2, \ldots, S_{|T|}\}$
  Observed space $(O) = \{O_1, O_2, \ldots, O_{|V|}\}$
  Transition matrix (T) of size $|T * T|$
  Emission matrix (E) of size $|V * T|$
  Initial probabilities matrix (I) of size $|1 * T|$
  Sequence of observations (Y) of length N.
  Output: Most likely hidden state sequence $(X) = T_1, T_2, \ldots T_N$

- def Viterbi$(O, S, I, T, E, Y) \rightarrow X$:
  for each state s from 1 to $|T|$ do:
  Viterbi$[s, 1] = I(s) \times E[s, 1]$
  (probability that first word is tagged as s)
  $BP[s, 1] = 0$ (no backpointer, no previous tag for start of sequence)

  for each step t from 2 to N do:
  for each state s from 1 to $|T|$ do:
  Viterbi$[s, t] = \max$ over k (Viterbi$[k, t-1] *$ $T[k, s] * O[s, t]$)
  $BP[s, t] =$ argmax over k (Viterbi$[k, t-1] *$ $T[k, s] * O[s, t]$)
  (for each word after the first word, for each tag s, look at every previous tag k of word $t-1$ and for each k compute the probability of best path to tag k at previous word $\times$ probability of moving from tag $k \rightarrow s$ $\times$ probability that tag s emits current word and choose the maximum among these probabilities)
  (Also store which previous tag k gave the maximum)

Z_N = argmax[S in 1 to N](Viterbi[S, N])
X_N = S[Z_N] (for final word, pick the tag with highest Viterbi score)

for i in 1,..., 2 do:
  Z_{i-1} = BP[Z_i, i] (Backtracking to recover full tag sequence)
  X_{i-1} = S[Z_{i-1}]
return X

Time Complexity = $O(N^2 |T|^2)$
Space Complexity = $O(N * |T|)$

(1) Initialization: Start by calculating the probability of the first word in the sentence having each possible tag (Initial × Emission)

(2) For each word in the sentence (after first word), for each possible tag of current word, calculate (transition k→s, × emission of current word with s)

(3) Termination: Once final word is processed, find highest probability among all possible tags for last word.

(4) Backtracking: Backtrack through previous words to find most likely sequence of tags.

Ex: Doctor diagnoses fever by asking patients how they feel each day

Hidden States: {Healthy (H), Fever (F)}
Observations: {cold (c), dizzy (d), normal (n)}

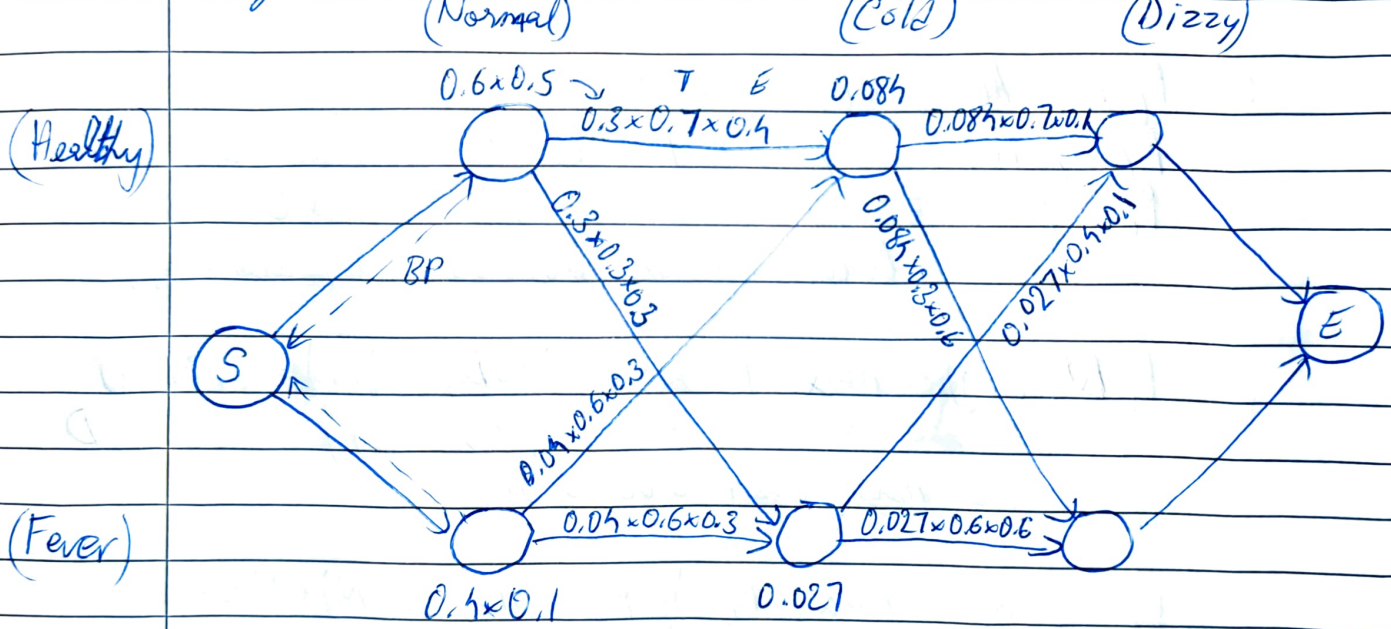
P(H) = 0.6 P(F) = 0.4

Find diagnosis of patient that reports for 3 days:
Normal → Cold → Dizzy.

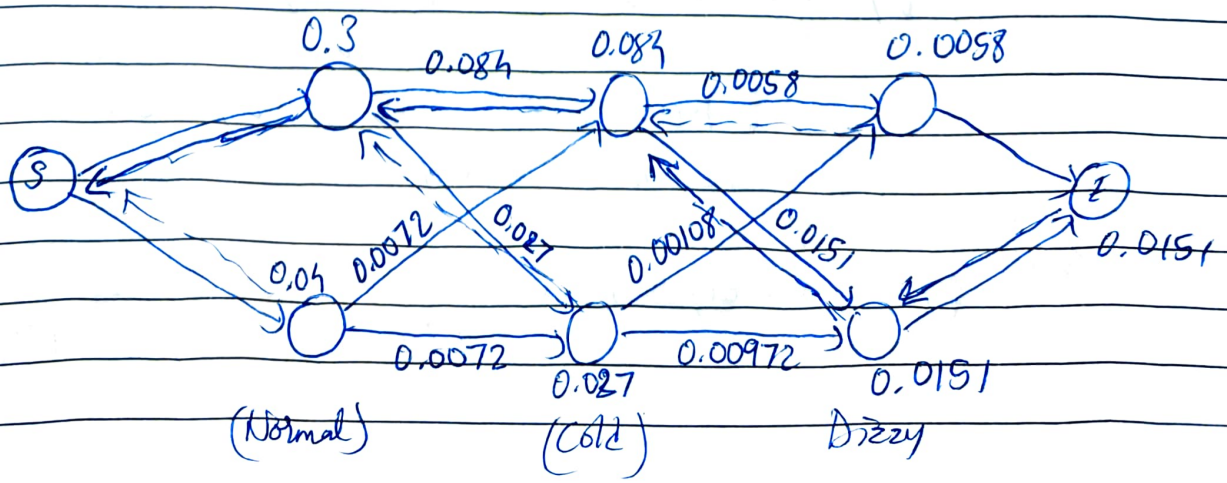
T	Healthy	Fever
Healthy	0.7	0.3
Fever	0.4	0.6

E	Healthy	Fever
Normal	0.5	0.1
Cold	0.4	0.3
Dizzy	0.1	0.6

We can represent how probabilities evolve over time across possible states using a Trellis diagram.



- Here BP (backpointer) tells which previous state (tag) gave the maximum probability path.



Eg: Consider the sentence: "The dog barked"

I: $P(\text{Determiner}) = P(D) = 0.6$ (how likely each tag is at start of sentence)
$P(\text{Noun}) = P(N) = 0.3$
$P(\text{Verb}) = P(V) = 0.1$

E:

	D	N	V
The	0.9	0.1	0.0
dog	0.0	0.8	0.2
barked	0.0	0.1	0.9

T:

→	D	N	V
D	0.5	0.5	0.1
N	0.2	0.2	0.6
V	0.1	0.3	0.6

① $P(D/\text{The}) = P(D) \times P(\text{The}/D) = 0.6 \times 0.9 = 0.54$
$P(N/\text{The}) = P(N) \times P(\text{The}/N) = 0.3 \times 0.1 = 0.03$
$P(V/\text{The}) = P(V) \times P(\text{The}/V) = 0.1 \times 0 = 0$
(Most likely tag for 'the' is D (determiner)

② $P(D/\text{dog}) = \max(P(D) \times P(D \to D), P(N) \times P(N \to D), P(V) \times P(V \to D)) \times P(\text{dog}/D)$
$= \max(0.24, 0.06, 0.01) \times 0 = 0$

$P(N/\text{dog}) = \max(P(D) \times P(D \to N), P(N) \times P(N \to N), P(V) \times P(V \to N)) \times P(\text{dog}/N)$
$= \max(0.3, 0.06, 0.03) \times 0.8 = 0.24$

$P(V/\text{dog}) = \max(P(D) \times P(D \to V), P(N) \times P(N \to V), P(V) \times P(V \to V)) \times P(\text{dog}/V)$
$= \max(0.06, 0.18, 0.06) \times 0.2 = 0.036$
(Dog is most likely N (noun))

③ $P(D/\text{barked}) = \max(P(D) \times P(D \to D), P(N) \times P(N \to D), P(V) \times P(V \to D)) \times P(\text{barked}/D)$
$= 0$

__/__/____

$P(N/barked) = \max(P(D) \times P(D \to N), P(N) \times P(N \to N), P(V) \times P(V \to N)) \times P(barked/N)$

$= 0.3 \times 0.1 = 0.03$

$P(V/barked) = 0.18 \times 0.9 = 0.162$

(barked is most likely a V (verb))

By backtracking the highest probabilities for each word, most likely sequence of tags for "the dog barked" is "D N V".

- To measure how well a POS tagger performs, we compare it's output against a human-tagged reference called a Gold Standard (Brown Corpus)

- To train the model, we need to split the corpus into training sets (to compute frequencies, transition and emission counts) and testing sets (to compute accuracy of tagger, never used during training).

$\%\ correct = \frac{No.\ of\ words\ tagged\ correctly\ in\ test\ set}{Total\ no.\ of\ words\ in\ test\ set}$

- Error Analysis is the process of identifying and understanding why a POS tagger makes mistakes. Helps improve tagger performance and understand model weaknesses:

(Errors) (i) Ambiguity (words have multiple POS tags depending on context)

(ii) Unknown Words: (Out-Of-Vocabulary) (words not seen in training data)

(iii) Contextual Errors (tagger fails to consider surrounding context)

(iv) Model bias / training data limitations

- Some techniques for error analysis include:
  - Confusion Matrix (which tags model commonly confused with)
  - Manual Inspection
  - Error Categorization.

☆ Noisy Channel Model

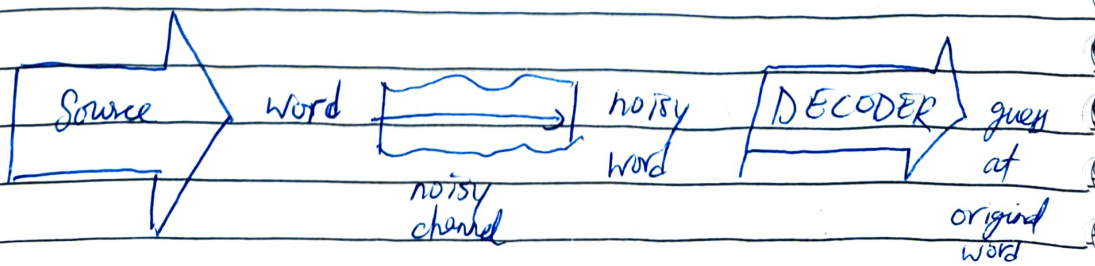
- Probabilistic framework used to recover an intended (original) message from an observed but possibly distorted input (correcting spelling errors essentially)

- Non-word errors ⇒ graffe → giraffe
  (any word not in the dictionary is an error)

- Real-word errors (detect spelling errors) (contextual spell checking)
  ⇒ three → there (typological errs)
  piece → peace (homophones)
  too → two

- For correction, generate candidates (real words that are similar to errs) and choose one which has either shortest weighted edit distance (or) highest noisy channel probability.

- Special case of Bayesian inference



There exists a true, intended word w, a noisy channel corrupts it and produces the observed word x.

$$\hat{w} = \underset{w \in V \text{ (vocabulary)}}{\operatorname{argmax}} P(w/x)$$ (choose the word w that most likely generated misspelling x)

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(x/w)\,P(w)$$ (using Bayes rule, since $P(x)$ is constant for all words)

Here $P(x/w)$ → likelihood (how likely is it to misspell w as x)

$P(w)$ → prior (how common that word is in English) (derive from unigram/bigram counts)

- Error patterns:

(a) Insertion ↘ (b) Deletion ↗ (c) Substitution ↔

(d) Transposition (swap positions of two diff characters)

Ex Noisy Channel Probability for "acress":

~~Candidate Correction Correct letter Error letter~~

	Candidate Correction	Correct Letter	Error Letter	x/w	$P(x/w)$	$P(w)$	$10^9 \times P(x/w)P(w)$
~~I~~	actress	t	–	c/ct	0.000117	.0000231	2.7
~~D~~	cress	–	a	a/#	0.0000144	.00000274	0.00078
T	caress	ca	ac	ac/ca	0.00000164	.0000170	0.0028
S	access	c	r	r/c	0.000000209	.0000916	0.019
S	across	o	e	e/o	0.0000093	.000299	2.8 ✓
D	acres	–	s	es/e	0.0000321	.0000318	1.0
D	acres	–	s	ss/s	0.0000342	.0000318	1.0

__/__/____

Case Study (Python)

```
import nltk
nltk.download('average_perceptron_tagger')
text = nltk.word_tokenize("My name is Johann")
nltk.pos_tag(text)
```

Output: `[('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('Johann', 'NNP')]`

```
nltk.similar('word')        (same context used)
```

```
tagged_token = nltk.tag.str2tuple('fly/NN')
```

(Automatic Tagging)

```
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
```

```
nltk.DefaultTagger('NN').tag(tokens)
```
(assigns same tag to all tokens in corpus)

```
tagger.evaluate(brown_tagged_sents)
```

(Regular Expression Tagger)

```
pattern = [(r'.*ing$', 'VBG'),          (present participle)
          (r'.*ed$', 'VBD'),            (past tense)
          (r'.*es$', 'VBZ'),            (present tense)
          (r'.*ould$', 'MD'),           (modal verbs)
          (r'.*\'s$', 'NN$'),           (possessive noun)
          (r'.*s$', 'NNS'),             (plural noun)
          (r'.*', 'NN')                 (noun)
]
```

→ regexp_tagger = nltk.RegexpTagger(pattern)
regexp_tagger.tag(brown_sents)

→ (N-gram Tagging)
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
size = int(len(brown_tagged_sents) * 0.9)

train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]
tagger = nltk.UnigramTagger(train_sents) (Train)
tagger.evaluate(test_sents)

tagger.tag(brown_sents[2007])

→ For bigram tagging, use nltk.BigramTagger()

(Context = current word + POS tags of n-1 preceding tokens)

__/__/__

FORMAL GRAMMARS OF ENGLISH

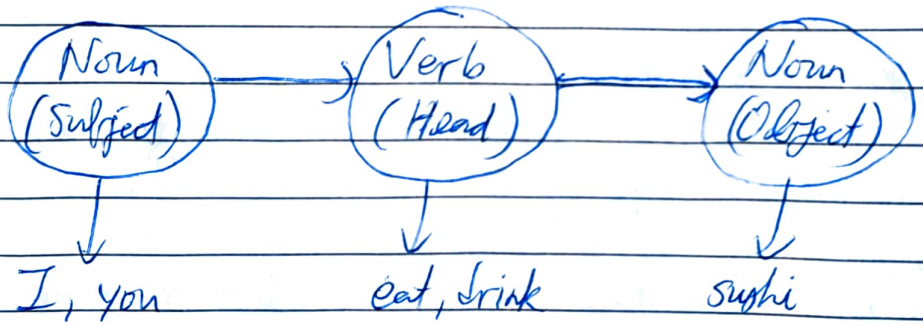
- Grammar can be defined as a system of rules that define how words combine into sentences.

- Grammar formalisms are abstract frameworks (like programming languages for linguists) that describe sentence structure using data structures and operations.

Ex:- CFGs, dependency grammars, FSA, probabilistic models.

- Specific grammars are concrete implementations of formalisms for a particular language.

FSA :-



HMM

- Constituents are groups of words that function as a single unit within a sentence (words, phrases, clauses)

Ex:- The cat ⇒ Noun Phrase (NP)
is on ⇒ Verb Phrase (VP)

- In formal grammars, grammatical relations refer to the syntactic roles that words and phrases play in a sentence (define how they function within a sentence's structure and how they relate to each other.)

- Sentence structure is a hierarchical setup consisting of words, which form phrases / constituents.

- Subcategorization (Valency)

• A verb comes with a template about what arguments it needs to form a complete sentence. This template is known as subcategorization frame.

Ex- Consider the verb 'eat', it subcategorizes for two arguments, a subject (S) and a direct object (O)
Frame for eat : [S DO]
In the sentence "She eats pizza", "She" is the subject S and "pizza" is the direct object DO.

- Dependency Relations

• Represent syntactic relationships between words in a sentence, on a dependency tree.

• Each word in a sentence depends on another word represented as labelled direct links.

• Head (root): Each word in the sentence depends on it. (main verb)

• Dependent: Word that depends on the head

• Head governs the dependent, defining the grammatical relation between the two.

• Dependency relations are labelled to indicate the specific syntactic role of the dependent in relation to the head.

(i) nsubj (nominal subject) (subject of a clause)
(She eats)
↖ nsubj

dobj

(ii) dobj (direct object) (object of a verb) (She eats an apple)

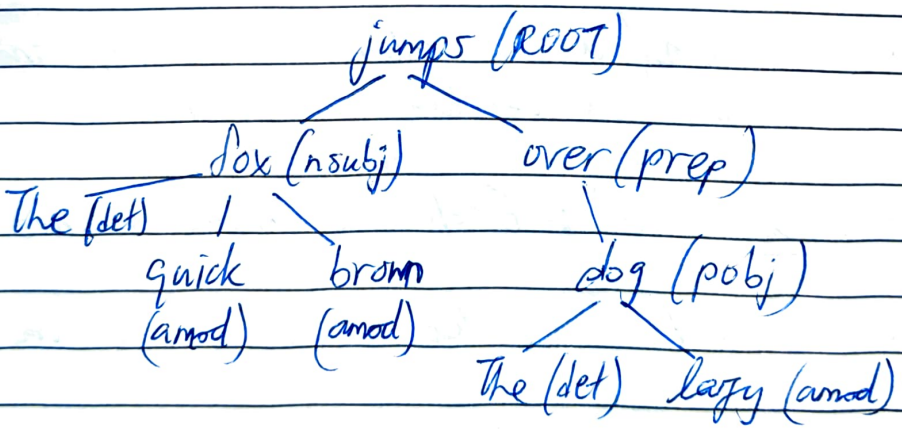
(iii) amod (adjectival modifier) (adjectives modifying the noun)

(iv) prep (prepositional modifier)

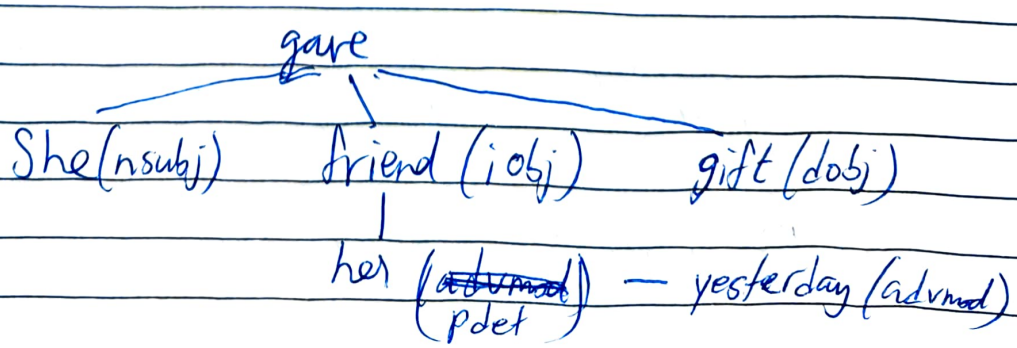
Ex: The red apple on the table
amod ↑ prep

- Projective dependency trees allow no edges to cross
- Non-projective trees occur when dependencies involve crossing edges, often seen in free-word-order languages (complex like Czech, Russian)

Ex: "The quick brown fox jumps over the lazy dog"

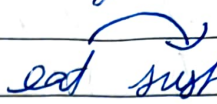


Ex: "She gave her friend a gift yesterday."



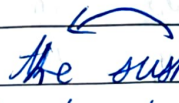
- Different kinds of dependencies are:

(a) Head-argument: Arguments are required by head to complete the meaning



(b) Head-modifier: Modifiers add detail about head but not required.

(c) Head-specifier: Function words that specify the head (determiner, prepositions)



(d) Coordination: Unclear where the head is [sushi and sashimi]

Note: Few examples of dependencies with cross-branches include extraposition (The guy is coming who is wearing a hat) topicalization (Cheeseburgers, I thought he likes)

— Context-Free Grammar

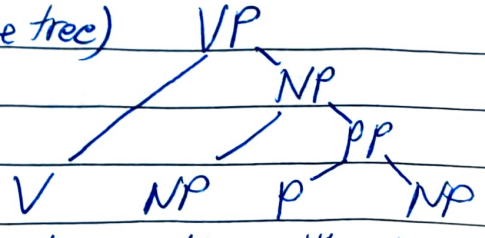
- Consists of a set of rules or productions, each which expresses the way symbols of the language can be grouped and ordered together, and a lexicon of words and symbols

- A CFG is a 4-tuple $\langle N, \Sigma, R, S \rangle$ consisting of a set of non-terminals (N) (tags/POS), set of terminals Σ (words), set of rules R and a start symbol $S \in N$.

Exa:

$N \rightarrow$ {sushi, tuna}
$P \rightarrow$ {with}
$V \rightarrow$ {eat}
$NP \rightarrow N$, $PP \rightarrow P\ NP$
$NP \rightarrow NP\ PP$, $VP \rightarrow V\ NP$
(prepositional phrase)

(Parse tree)



__/__/____

- Some Grammar Rules in English include:

① Declarative sentences
S → NP VP (I prefer morning flight)
(pronoun) (verb) (det) (noun)

② Imperative sentences (no subject)
S → VP (Show the latest fares)

③ Yes - No Questions
S → Aux NP VP
S → Aux NP VP
→ Does Det N PP VP
→ Does any of these flights VP
→ Does any of these flights Verb NP
→ Does any of these flights have stops.

④ Wh - structures (subject)(question)
(when, where, what, which, why)

S → Wh - NP VP (What airlines from BLR to DEL)

⑤ Wh - non subject question structure

S → Wh - NP Aux NP VP (+ subject)
(What flights do you have from Tanzania to India)

→ x

PARSING WITH CONTEXT-FREE GRAMMARS

- Parsing is the process of analyzing the grammatical structure of sentences in natural language.
- Shows how a sentence is built using phrases, based on constituents
- CFG is a set of rules used to construct parse trees.
- In syntactic parsing, parsers can be viewed as searching through the space of all possible parse trees to find the correct parse tree for the sentence. (just like FSA)
- Since a grammar can generate many possible trees, the parser must search through those possibilities to find valid one.
- Search space = all trees the grammar can generate.
- There are two constraints that guide the search:
  (a) trees must produce exact same input words (data constraint)
  (b) tree must have root S and use only allowed grammatical rules (grammatical constraints)

Top-Down Search (Goal-Directed)

- Start from the root S, expand downward using grammar rules to predict what could generate the input.

(Pros)
- Never wastes time with trees that do not lead to S

(Cons)
- But wastes time predicting trees that do not match input, generates structure even before seeing input

__/__/__

— **Bottom-Up Search (Data-Directed)**

- Starts from words
- Build upward by applying grammar rules
- Tries every combo that can form larger constituents.

(Pros) • Everything generated is grounded in actual input

(Cons) May build subtrees that may never lead to S, thus wasting time.

Note: In top-down parser, trees were expended by applying rules when LHS matched unexpanded non-terminal. In bottom-up parser, trees are extended by looking for rules where RHS might fit parse-in-progress.

☆ • **Ambiguity**

- Happens when a sentence can have more than one parse tree (structural ambiguity)

- Attachment ambiguity happens when a constituent can attach at more than one location in the parse tree

Exi- "I shot an elephant in my pajamas"
Here "in my pajamas" can attach to NP (elephant in my pajamas) or VP (shot ... in my pajamas)

- Coordination ambiguity happens when it is unclear which phrases are being ~~control~~ joined by 'and'.

Exi- old men and women (both are old)
oldmen and women (only men are old)

__/__/__

- Syntactic Disambiguation happens when NLP systems need to choose the right parse from many possible parses. (many of which are incorrect and unreasonable)

- Even if whole sentence has only one parse, parts of it may be ambiguous temporarily (local ambiguity). making parsing slow and inefficient.

☆ - CKY Algorithm

- Dynamic programming algorithm used for parsing sentences in formal grammar, particularly CFG.

- Determine if given sentence can be generated by a CFG and if so, generates a parse tree for the sentence.

- Uses a chart to store intermediate results, enabling efficient parsing of sentences.

- Input: A sequence of words $W = w_1, w_2, w_3, \ldots, w_n$
  CFG: A set of rules R in CNF

- Output: Parse chart (2D array) where each cell $[i,j]$ contains a set of non-terminals that can generate the substring $W[i,j]$ in the sentence.

① Create an empty parse chart with dimensions $n \times n$ where $n \rightarrow$ no. of words in the sentence.
② Fill diagonal cells with non-terminals that can derive the individual words in the sentence.
③ Traverse the chart diagonally from bottom to top (increasing substring length)

④ For each cell $[i,j]$ where $i<j$, calculate the non-terminals that can generate the substring $w[i,j]$

⑤ Finally, the parse chart will contain the non-terminals that generate the entire sentence at cell $[0, n-1]$

Ex:- "The cat chased the mouse"
1 2 3 4 5

(Grammar)

$S \rightarrow NP\ VP$ — $Det \rightarrow$ "The"

$NP \rightarrow Det\ N$ — $N \rightarrow$ "cat" / "mouse"

$NP \rightarrow N$ — $V \rightarrow$ "chased" (terminal rules)

$VP \rightarrow V\ NP$ (binary rules)

	1	2	3	4	5
1	Det	NP	ϕ	ϕ	S
2		N, NP	ϕ	ϕ	S
3			V	ϕ	VP
4				Det	NP
5					N, NP

For cell (1,2), between (1,1) and (2,2),
Det + N → NP (matches rule NP → Det N)

For cell (2,3), NP + V → ϕ (no matching rules)
N + V → ϕ

For cell (3,3) V + Det → ϕ

For cell (4,5), Det + N → NP
Det + NP → ϕ

For cell (1,3), (1,1) + (2,3) ⇒ Det + ϕ ⇒ ϕ
(1,2) + (3,3) ⇒ NP + V → ϕ (order matters)

For cell (3,5), V + NP → VP
ϕ + N → ϕ

__/__/__

For cell (2,5), NP + VP → S

For cell (1,5), Det + S → Q
NP + VP → S

Note: For every cell (i,j):
Set i → start row index
Set j → end column index
Try all k from i to j-1:
Combine left subtree (i,k) with right subtree (k+1,j)
Use grammar rules A → BC to decide (i,j) content.

Note: Rules should be Chomsky Normal Form (CNF) such that A → BC or A → w (A,B,C → non-terminal, w → terminal, actual word)

For the rule: VP → Verb-NP-PP
VP → X2 NP
X2 → Verb PP