Why is NLP hard?

i) Lexical Ambiguity:

ex: A) Will Will will Will's will?
B) Rose rose to put rose rose on her rows of roses.
C) Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo

ii) Language Ambiguity: Structural
→ diff interpretations of the same sentence

ex: A) The man saw the boy with the binoculars
B) Flying planes can be dangerous
C) Hole found in the room wall; police are looking into it.

iii) Language imprecision & vagueness

ex: A) it is very warm here [what is considered warm in Chennai is diff from what is considered warm in US]

B) Did your mom call your aunt last night?
A: I'm sure she must have. [basically saying idk]

Catalan Number: as the no of phrases in a sentence ↑ the no. of interpretations in which they can be connected in a sentence ↑

Why is Language Ambiguous?

→ goal in the production & comprehension of natural lang is efficient communication
→ allowing resolvable ambiguity
- permits shorter linguistic expressions
- avoids language being overly complex

→ Language relies on ppl's ability to use their knowledge & inference abilities to properly resolve ambiguities.

Natural Language vs Computer Language

1. CL doesn't have any ambiguity. They mean only one thing.
2. Formal programming lang are designed to be unambiguous
↳ can be defined by a grammar that produces a unique parse for each sentence in the lang.
3. Programming Languages are also designed for efficient (deterministic) parsing.

Why else is NLP Hard?

→ idioms
- dark horse
- ball in your court
- burn the midnight oil

→ neologisms (used in social media as verbs)
- unfriend
- retweet
- Google / Skype / photoshop

Empirical Laws

- Function words: have little lexical meaning but serve as important elements in the structure of sentences
- to make the sentence more grammatical
- ex: words like 'a', 'the', pronouns etc

- Content words are various nouns & verbs that convey what are the important concepts in the sentence.
- mainly used for determining the structure of a sentence.

Function words are closed-class words
→ there are very specific grammatical categories these called function words
→ they are close-class words in the sense that we dont have newer & newer function words coming.
ex: prepositions, pronouns, auxiliary verbs, conjunctions, grammatical articles, particles etc, determiners

Type-Token Distinction:

type-token distinction is a distinction that separates a concept from the objects which are particular instances of the concept.

concept → Type
instance of concept → Tokens

ex: Will Will
Type = 1 → unique words
Token = 2 → number of words (not unique)

Type-Token Ratio:

- TTR = types (no of diff words) / tokens (no. of running words) in a text or corpus
- this index indicates how often, on average, a new 'word form' appears in the text or corpus.
- High TTR ⇒ lots of new words coming
Low TTR ⇒ words in the corpus keep getting repeated

TTR by itself is not a valid measure of 'text complexity'
→ the value varies with the size of text
→ for a valid measure, a running avg is computed on consecutive 1000-word chunks by text.

Zipf's Law:

→ Count the freq of each word type
→ List the word types in decreasing

- Zipf's Law gives the rel" b/w the & its position in the list (its ra

$$f \propto \frac{1}{r}$$

$$f \cdot r = \text{constant}$$

i.e, the 50th most common word 3 times the frequency of the 150th

most common / rank	freq
50	f_1
150	f_2

$$f_1 = 3 f_2$$

Another way to denote this,
let P_r = denote the probability of
N = denote the total no. of

$$P_r = \frac{f}{N} \quad \text{(by definition)}$$

$$f \propto \frac{1}{r}$$

$$\Rightarrow \frac{f}{N} = \frac{A}{r}$$

$$f \cdot r = A N \} \; K = A \cdot N$$
$$f \cdot r = K \quad A \approx 0.$$

Zipf's Other Laws:

Correlation: Number of meani
The no. of meanings m of a wo

$$m \propto \sqrt{f}$$

given the first law,

$$m \propto \frac{1}{\sqrt{r}}$$

~~Combining~~

Correlation: Word Length & word frequency is inversely p their length

Zipf's Law:

→ Count the freq of each word type in a large corpus
→ List the word types in decreasing order of their freq

• Zipf's Law gives the rel" b/w the freq of a word (f) & its position in the list (its rank r) as:

$$f \propto \frac{1}{r}$$

$$f \cdot r = \text{constant}$$

i.e, the 50th most common word should occur with 3 times the frequency of the 150th most common word.

most common/rank	freq
50	f_1
150	f_2

$$f_1 = 3f_2$$

Another way to denote this,
Let P_r = denote the probability of word of rank r
N = denote the total no. of word occurrences

$$P_r = \frac{f}{N} \quad \text{(by definition)}$$

$$f \propto \frac{1}{r}$$

$$\Rightarrow \frac{f}{N} = \frac{A}{r}$$

$$f \cdot r = A \cdot N \} \; K = A \cdot N$$
$$f \cdot r = K \quad A \approx 0.1$$

Zipf's Other Laws:

Correlation: Number of meanings & word freq
The no. of meanings m of a word obeys the law:

$$m \propto \sqrt{f}$$

given the first law,

$$m \propto \frac{1}{\sqrt{r}}$$

} Says Law

~~Combining~~

Correlation: Word Length & Word Frequency
word frequency is inversely proportional to their length

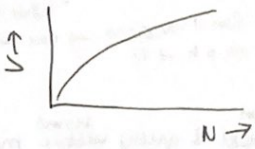
Heaps Law:

• gives relationship b/w size of vocab V and no of tokens N

$$|V| = KN^{\beta}$$

$K \approx 10-100$
$\beta = 0.4-0.6$

no of unique tokens ↑ with corpus size



Text Processing : Basics

1) Tokenization : split string into char of words

2) Sentence Segmentation
   - challenge → where does sentence end?
     - ↳ use binary classifier
     - ↳ use decision tree

3) Decision Tree
   - ↳ feature can be separated into classes
   - ↳ issues
     - A) how to tokenize "Finland's"
     - B) "What're"
     - C) San-Francisco → one token or 2 tokens
     - D) m.p.h → ??

4) Normalization
   - ↳ indexed text & query ~~vectors~~ terms must have the same form
     - ex: U.S.A & USA should be matched.

5) Lemmatization
   - ↳ reduce inflections or variant forms to base form
     - am, are, is → be
     - car, cars, car's, cars' → car

Spelling Correction : Edit Distance

- data obtained is noisy (with spelling errors)

How to use Edit Distance for Spelling correction?

Situation: There is a sentence
"I am writing this email on behy of..."
Spelling of behy is wrong.

You list down the words closest to the misspelled word:

→ behy
→ behive ... etc.

Now, you have to choose 1 word from the given list of candidate words.

Isolated word error correction: trying to correct a word that is incorrect but not trying to use the context around it.
∴ we want to correct "behy" w/o seeing the words before & after it.

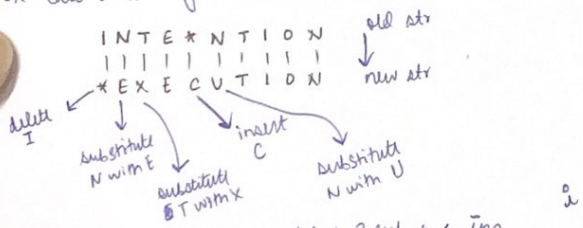
∴ we use only candidate keys & the misspelled word.

How do we find out which candidate word is closest to the misspelled word?
→ using a distance metric

Edit Distance : The simplest metric

→ minimum no. of edit operations needed to go from one string to another
→ various edit operations : insertion, deletion & substitution

ex: edit distance from 'intention' to 'execution'



⇒ 5 diff operations : 1 del + 3 sub + 1 ins

→ 2 diff variations to use :
(i) Each operation has a cost of 1 [Levenshtein]
↳ edit distance = 5 op × cost of 1 = 5

(ii) Substitution has a cost of 2
↳ edit distance = (1 del * 1) + (1 ins * 1) + (3 sub * 2)
= 1 + 1 + 6
= 8

How to find the minimum edit distance?
→ search for a path (sequence of edits) from the start string to the final string
i) initial state : word we are transforming (intention)
ii) operators : ins, del, sub
iii) goal state : final string (execution)
iv) path cost : we want to minimize : the no. of edits

2 strings : X of length n
Y of length m

We define D(i,j):
→ the edit distance b/w X[1...i] and Y[1...j]
i.e, the first i char of X and the first j char of Y

The edit dist^n b/w X and Y is D(n,m)

Dynamic Prog Approach :

→ tabular computation of D(n,m)
→ solving prob by combining sol^n to subproblems
→ Bottom-Up :
i) compute D(i,j) for small i,j
ii) compute larger D(i,j) based on prev computed smaller values
iii) compute D(i,j) for all i & j values till you get to D(n,m)

Algorithm:

D(i,0) = i
D(0,j) = j

For each i = 1...M
For each j = 1...N

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 & \text{(del)} \\ D(i,j-1)+1 & \text{(ins)} \\ D(i-1,j-1)+2; & x(i) \neq y(j) \\ \text{(sub)} \quad +0; & x(i) = y(j) \end{cases}$$

Termination:
D(N,M) is distance

										edit distance
N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	5	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

D(i-1,j) = 1 + 1 = 2
D(i,j-1) = 1 + 1 = 2
D(i-1,j-1) = 0 + 2 = 2

D(i,j-1) = 2 + 1 = 3
D(i-1,j-1) = 1 + 2 = 3
D(i-1,j) = 2 + 1 = 3

$$ptr(i,j) = \begin{cases} \text{LEFT} & \text{(ins)} \\ \text{DOWN} & \text{(del)} \\ \text{DIAG} & \text{(sub)} \end{cases}$$

Non-word spelling error detection

- any word not in a dict is an error
- the larger the dict the better

Non-word spelling error correction

i) generate candidates: real words that are similar to the error word
ii) choose the best one
  - ↳ shortest weighted edit dist
  - ↳ highest noisy channel prob

Noisy Channel Model for Spelling Correction

we have an observation X, the misspelt word
we have the correct word w

now, we wanted to use w, but while going through a channel, we end up with a word x.

$$w \xrightarrow[\text{channel}]{\text{noisy}} x$$

Now, how do we find w, given x, the misspelt word.

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}}\ P(w|x) = \underset{w \in V}{\operatorname{argmax}}\ \frac{P(x|w)P(w)}{P(x)}$$

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}}\ P(x|w)P(w)$$

words with similar spelling } small edit distance to error

words with similar pronunciation } small edit dist of pronunciation to error

Damerau-Levenshtein edit distance

minimum edit distance, where edits are:
ins, del, sub & transposition of 2 adj letters

Error	Candidate Correction	Correct Letter	Error Letter	Type	P(x|w)
acress	actress	t	–	del	c|ct = 0.00017 ← highest
acress	acress	–	a	ins	a|# = 0.000001
acress	caress	ca	ac	trans	ac = 0.000016
acress	access	c	r	sub	r|c = 0.000002 09 ← 2nd highest
acress	across	o	e	sub	e|o = 0.0000093
acress	acres	–	s	ins	es|e = 0.0000321
acress	acres	–	s	ins	ss|e = 0.0000342

$$P(x|w) = \begin{cases} \dfrac{del(w_{i-1}, w_i)}{count(w_{i-1} \cdot w_i)} & ; del \\ \dfrac{ins(w_{i-1}, w_i)}{count(w_{i-1} \cdot w_i)} & ; ins \\ \dfrac{sub(w_{i-1}, w_i)}{count(w_{i-1} \cdot w_i)} & ; sub \\ \dfrac{trans(w_{i-1}, w_i)}{count(w_{i-1} \cdot w_i)} & ; trans \end{cases}$$

N-Gram Language Models

Applications :
i) Speech Recognition
ii) Machine Translation
iii) Context-Sensitive Spelling Correction
iv) Natural Language Generation
v) Sentence Completion / Next word Prediction

Probabilistic Language Modelling

Goal: compute the prob of a sentence or seq of words

$P(W) = P(w_1, w_2, w_3 \dots w_n)$ → model that computes either of these is a language model

related task: prob of an upcoming word

$P(w_4 \mid w_1, w_2, w_3)$ → model that computes either of these is a language model

How to compute joint probability :

P(about, fifteen, minutes, from)

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

$\Rightarrow P(A,B,C) = P(A) \cdot P(B|A) \cdot P(C|A,B)$

$P(A,B,C,D) = P(A) \cdot P(B|A) \cdot P(C|A,B) \cdot P(D|A,B,C)$

P("about fifteen minutes from") = P(about) · P(fifteen/about) · P(minutes | about fifteen) · P(from | about fifteen minutes)

$P(\text{office} \mid \text{about fifteen mins from}) = \frac{\text{count(about fifteen mins from office)}}{\text{count(about fifteen mins from)}}$

simplify this assumption :

$P(\text{office} \mid \text{about 15 mins from}) \approx P(\text{office} \mid \text{from})$

(office = current word; from = prev word to current word)

K^{th} ordr markov model :

$P(w_1, w_2 \dots w_n) = \prod_i P(w_i \mid w_{i-K} \dots w_{i-1})$

$\Rightarrow P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-K} \dots w_{i-1})$

→ An N-gram model uses only N-1 words of prior context

Unigram: P(office)
Bigram: P(office | from)
Trigram: P(office | minutes from)

An N-gram model is an (N-1) ordr Markov model

Estimating N-gram probabilities

(i) Maximum Likelihood Estimate

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

Example :

<s> I am here </s>
<s> who am I </s>
<s> I would like to know </s>

P(I | <s>) = 2/3
P (am | I) = 1/1
P (here | am) = 1/2
P (who | <s>) = 1/1
P (am | who) = 1/1
P (I | am) = 1/2
P (would | I) = 1/3
⋮
and so on

Example 2 :

Train sentences :

<s> Thru friends Amar, Akbar and Antony are reading book </s>
<s> Amar is reading Malgudi days </s>
<s> Antony is reading a book by Rahman </s>
<s> Akbar is reading a detective book </s>

	count
Thru	1
friends	1
Amar	2
Akbar	2
Antony	2
and	1
are	1
reading	4
a	2
book	3
is	3
Malgudi	1
Days	1
Detective	1
by	1
Rahman	1

<s> = 4
</s> = 4

Test sentence

<s> Amar is reading a book </s>

P(Amar | <s>) * P(is | Amar) * P(reading | is) * P(a | reading) * P(book | a) * P(</s> | book)

$$= \frac{\text{count}(\text{Amar}, <S>)}{\text{count}(<S>)} * \frac{\text{Count}(\text{is}, \text{Amar})}{\text{count}(\text{Amar})} * \frac{\text{Count}(\text{Reading}, \text{is})}{\text{Count}(\text{is})} *$$

$$\frac{\text{count}(a, \text{reading})}{\text{count}(\text{reading})} * \frac{\text{count}(\text{book}, a)}{\text{count}(a)} * \frac{\text{count}(</s>, \text{book})}{\text{count}(\text{book})}$$

$$= \frac{1}{4} * \frac{1}{2} * \frac{3}{8} * \frac{2}{4} * \frac{1}{2} * \frac{2}{3}$$

$$= \frac{1}{4} * \frac{1}{2} * \frac{2}{4} * \frac{1}{2} * \frac{2}{3}$$

$$= \frac{2}{24} = \frac{1}{12} = 0.083$$

Practical issues.

→ multiplication of many prob values
↳ do log as it avoids underflow & adding is faster than multiplying

$$\log(p1 * p2 * p3) = \log p1 + \log p2 + \log p3$$

→ handling zeros
↳ use smoothing

Evaluation of Language Models, Basic Smoothing

1. One way to evaluate is by assigning high prob to real sentences (freq obs) over ungrammatical ones (rarely obs)
   ↳ but this has to be done manually
   ↳ can it be done automatically?
   ↳ by dividing dataset into training & testing

Extrinsic Evaluation:

- learn 2 models A & B.
- use each model for one or more tasks such as spelling checker, speech recognizer etc
- compare performance of A & B
- whichever gives better perf, that model is preferable

(task-based evaluation)

Intrinsic Evaluation : Perplexity

Shannon Game (based on intuition)
↳ how well one can predict the next word

- we have 2 models.
- we apply them here & the models will give diff words to predict the next word in a sentence.
- whichever model predicts better is preferable

→ in this case, unigram models are not preferable
↳ don't use context around a word

→ in this case, a better model will be one that assigns a high prob to the actual word.

Perplexity:

→ we want a model that best predicts an unseen test data

Perplexity is the inverse prob of the test data, normalized by the no. of words.

→ Lower the perplexity, better the model

$$PP(w) = P(w_1 w_2 \dots w_N)^{-1/N}$$

where does the language model come into all this?

$$PP(w) = \left(\prod \frac{1}{P(w_i \mid w_1 \dots w_{i-1})}\right)^{1/N}$$ [applying chain rule]

→ in the form of a language model.
so in we can use how many ever context words we want to predict the next word.
Here, we can put in any lang model & test it.

ex: For bigram model,

$$PP(w) = \left[\prod \frac{1}{P(w_i \mid w_{i-1})}\right]^{\frac{1}{N}}$$

ex: consider a sentence consisting of N random digits
The model assigns a prob of $P = \frac{1}{10}$ to each digit
What is the perplexity of this sentence?

$$PP(w) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$
$$= P\left(\frac{1}{10} \times \frac{1}{10} \times \dots \frac{1}{10}\right)^{-\frac{1}{N}}$$
$$= \left(\left(\frac{1}{10}\right)^N\right)^{-1/N}$$
$$PP(w) = 10$$

ex 2: Training : 38 M words
Testing : 1.5 M words

Perplexity of Unigram model trained on this } 962
PP(w) of Bigram : 170
PP(w) of Trigram : 109

What 962 means is that whenever my model is trying to assign a word, it is as though it has to choose among 962 diff possibilities at each individual choice point
Therefore, the model is very perplexed (confused)

∴ when perplexity is high, model is confused (BAD)
when perplexity is low model is fine (GOOD)

Once we build a lang model, we can use it for other tasks like Shannon Visualization Method
↳ which means, can I use this model to visualize or generate sentences?

How to apply this?

Suppose I have to generate a sentence. What will I do?

→ Choose a random bigram $(\langle s \rangle, w)$ as per its prob, to start a sent.

$\langle s \rangle, w_1 \rightarrow$
$\langle s \rangle, w_2 \rightarrow$ [probs]
$\langle s \rangle, w_N \rightarrow$

choose w_i based on max prob

This is called sampling a word from multinomial dist"

Next choose a random bigram (w, x) as per its prob to continue the sent

$(w_2, w_1) \rightarrow$
$(w_2, w_2) \rightarrow$ [probs]
$(w_2, w_3) \rightarrow$
$(w_2, w_N) \rightarrow$

again choose a word based on this prob.
then again continue the process with (next word, candidate possibilities)

Now, when do we stop? How to say wh the sentence is complete?
→ when you sample a word that is end of a sentence $(w_i, \langle /s \rangle)$

Now, in a scenario, (for example) if we take a corpus, we will find then are 844 M bigrams. Out of them only 300 K bigrams are real occurring.
↳
This means that 300K bigrams have a while rest have prob = 0.

Now if you take a ~~train~~ testing d try to apply this model that was on the above corpus, then a bigra is present in testing data but not data will get assigned a prob = 0.

If you try to find perplexity of t pplexity is multiplication of all di multiplication o/p = 0.
We want the perplexity to have p greater than zero.
↓ How to tackle this issue of
By Smoothing
↳ assigning diff prob va

ex: Training set:
- ...denied the allegations
- ....denied the reports
- ....denied the claims
- ....denied the requests

Due to testing data, not being
P(offer | denied the) = 0
due to prob = 0, denominato
⇒ card con

Next choose a random bigram (w, x) as per its prob to continue the sent.

$(w_2, w_1) \rightarrow$
$(w_2, w_2) \rightarrow$ [p r o b]
$(w_2, w_3) \rightarrow$
$(w_2, w_N) \rightarrow$

again choose a word based on this prob.

then again continue the process with (next word, candidate possibilities)

Now, when do we stop? How to say when the sentence is complete?

→ when you sample a word that is the end of a sentence $(w_i, </s>)$

Now, in a scenario, (for example)
if we take a corpus, we will find that there are 844M bigrams. Out of all of them only 300K bigrams are real or frq occuring.
↳
This means that 300K bigrams have a prob > 0 while rest have prob = 0.

Now if you take a ~~train~~ testing data & try to apply this model that was trained on the above corpus, then a bigram which is present in testing data but not in training data will get assigned a prob = 0.

If you try to find perplexity of testing data, perplexity is multiplication of all diff prob, then multiplication o/p = 0.
We want the perplexity to have preferably value greater than zero.

↓How to tackle this issue of zeroes

By Smoothing
↳ assigning diff prob values to get around zeroes.

Ex: Training set:
- ....denied the allegations
- ....denied the reports
- ....denied the claims
- ....denied the requests

Testing set:
- .... denied the offer
- .... denied the loan

Due to testing data, not being part of training data
P(offer | denied the) = 0
due to prob = 0, denominator of perplexity = 0
⇒ cant compute perplexity

Idea of Smoothing:

P(w | denied the)
where w based on training ~~& testing set~~

word	count	prob
allegations	3	3/7
reports	2	2/7
claims	1	1/7
request	1	1/7
	7	

what abt the prob of other trigram words like offer & loan?

→ First, assign them a prob of zeroes
→ Second, idea of smoothing is can I take some prob mass from each of the 4 training set words & assign them to the ~~train~~ testing set words that can be said as unseen data.

So, if you remove 0.5 prob mass from the 4 words, you get a value of 2, which is prob 2/7, which you distribute among all other words in the corpus.
This distⁿ can be done in multiple ways

Laplace Smoothing

- also called "add-one estimation"
- here, pretend as if we saw every N-gram one more time than we actually did.
  So, for those N-grams that you have not seen at all, you pretend you've seen them once
  ↳ just add 1 to all the counts

MLE estimate for bigrams: $P_{MLE}(w_i/w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$

For laplace smoothing this,
we add 1 to numerator (for each possible bigram)
$\Rightarrow c(w_{i-1}, w_i) + 1$

but to ensure that the prob sums upto 1, we have to modify the denominator such that they are normalized, so what to add to the denominator.

↳ How many diff bigrams will be there for which we will be adding 1?
↳ $w_i = V$ (no of words in the vocab)
So, we will add one 'V' times.

So, to normalize, we'll have to add a V to the denominator

∴ Add-one estimate: $P_{add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$

Variant of Add-1: Add-K:

$P_{add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + K}{c(w_{i-1}) + KV}$

NOTE:
if $KV = m$
then $K = m(\frac{1}{V})$

Unigram Prior Smoothing:

$P_{\text{unigram prior}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$

Computational Morphology:

- the internal structure of words, how words are built up from smaller meaning units are called morphemes.

ex: dogs = dog + S ← plural marker on nouns

unladylike = un + lady + like ← having the char of
(not; lady = well behaved women)

Allomorphs:

- given a word, using morphemes, you convert the word to its opposite
- variants of the same morpheme, but can't be replaced by one another ⇒ means can't use un for ir-rational
- ex: un-happy
  ir- rational
  in- comprehensible
  im- possible
  ~~ir-~~

Bound Morphemes:

- can't appear as a word by itself
- -s (dog-s)
  -ly (quick-ly)
  -ed (walk-ed)

Free Morphemes:

- can appear as a word by itself
- often can combine with other morphemes too
- (house-s)
- walk (walk-ed)
- of
- the
- or

Stems & Affixes.

Stems (roots): the core meaning bearing units
Affixes: bits & pieces adhering to stems to change their meanings & grammatical functions

Types of Affixes:

① Prefix: before word
ex: un-, anti-, pre-

② Suffix: after word
ex: -ity, -ation, -ly, -ing

③ Infix: inbetween word
ex: in Sanskrit, vi-n-dati

Content Morphemes:

- carry some semantic content
ex: con, -able, un-

Functional Morphemes:

- provide grammatical info
ex: -s (plural),

Inflectional Morphology:

ex: Walk + ing → Walking

- makes changes in grammar such as number, tense, gender, case
- create new forms of same word → brings

Derivational Morphology:

ex: Drive + er → Driver
(verb → noun)

- create new words by changing
ex: logic, logical, illogical, illogically

Applications:

① text to speech synthesis
② search & informational retrieval
③ machine translation
④ grammar correction

Intro to POS Tagging

- given a sentence, identify the POS of each word

How many?

① Open class words (content words)
- nouns, verbs, adjectives, adverbs
- mostly content-bearing: they refer to objects, actions, & features in the world
- open class, since new words are added all the time

② Closed class words
- pronouns, determiners, prepositions, connectives
- limited no. of these
- mostly functional: to tie the concepts of a sentence together

POS examples

• N	Noun	chair, bandwidth, pacing
• V	Verb	study, debate, munch
• Adj	Adjective	purple, tall
• Adv	Adverb	unfortunately, slowly
• P	Preposition	of, by, to
• Pro	Pronoun	I, me, mine
• Det	Determinant	the, a, that, those

→ to do POS tagging, a standard set needs to be chosen
↳ Upenn Treebank tagset (45 tags)

ex: The grand jury commented on a no. of other topics
DT JJ NN VBD IN DT NN IN JJ NNS

Why is POS tagging hard?
- words often have more than 1 POS

ex: The back door : JJ
On my back : NN
Win the voters back : RB
Promised to back the bill : VB

POS Tagging: 2 Approaches

① Rule-Based Approach
- 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

② Statistical Tagging
- Get a training corpus of tagged text, learn the transformation rules from the most frequent tags (TBL tagger)
- Probabilistic: Find the most likely sequence of tags T for a sequence of words W

Probabilistic Tagging

Problem: we have some data $\{(d,c)\}$ of paired observations d and hidden classes c

Different instances of d and c
- Parts-of-Speech Tagging
  → words are observed and tags are hidden
- Text Classification
  → sentences/documents are observed & the category is hidden
  → categories can be positive/negative for sentiments
  → sport, politics, business for documents

Generative Models (Joint)

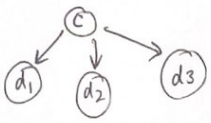
- Generate the observed data from hidden structure
  i.e, put a prob over the observations given the class
  ↳ $P(d,c)$ in terms of $P(d|c)$

ex: Naive Bayes
Hidden Markov Models

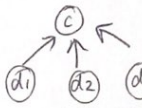
Discriminative (Conditional) Models

- Take the data as given, and put a prob over hidden structure given the data
  ↳ $P(c|d)$

ex: Logistic regression
maximum entropy models
conditional random fields



Naive Bayes



Logistic Regression

→ A joint model gives prob $P(d,c)$ and tries to maximize this joint likelihood

→ A conditional model gives prob $P(c|d)$ taking the data as given and modeling only the conditional probs of the class.

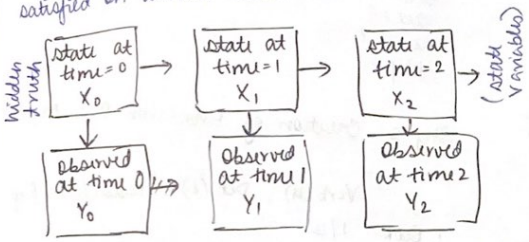
Hidden Markov Model (for POS Tagging)

HMM is a probabilistic model to infer unobserved data / info from observed data

ex:
in mobile, sometimes you mistype the char next to what you wanted to.
→ the char you mistyped is the observed data
→ one you intended to type is unobserved data

similarly,
there are many cases where what is observed is not the truth & the HMM is one way to give us the hidden truth.

NOTE:
there are some assumptions that need to be satisfied on which HMM works.



for each observed data, there's a hidden truth (hidden truth = state variable)

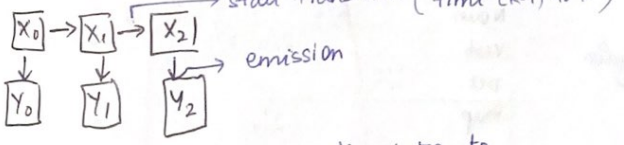
there are a sequence of states (hidden truths) linked with each other.
arrow = dependency
↳ any state depends ONLY on some of its previous state(s).

a state that doesn't depend on any prev state } = independent seq

a state depends on its immediate prev state } = simple markov chain

a state depends on its 'n' prev states } = n-order markov chain

State Transition, Emission & Initial State



i) assume there are M possible states to choose from $\{1, 2, \ldots M\}$

ii) transition prob from state i to j is a_{ij}

iii) NOTE:
transition can happen from any one state to another out of M possible states,
so there are M × M possibilities which can be arranged in a state transition matrix A

$$\text{From} \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0m} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1m} \\ \vdots & & & & \\ a_{m0} & a_{m1} & a_{m2} & & a_{mm} \end{bmatrix} \quad \text{(to)}$$

→ all a_{ij} b/w 0 & 1
→ sum of row = 1

for a stationary markov chain, matrix A is constant over time else, its diff for each timestep

$$a_{ij} = P(X_k = j \mid X_{k-1} = i)$$

$$S_k^T = S_{k-1}^T A \quad \text{where } S_k = \begin{bmatrix} P(X_k=1) \\ P(X_k=2) \\ \vdots \\ P(X_k=m) \end{bmatrix}$$

↓ multiplying the prob dist of time (k-1) by A updates the prob dist to next time step k.

emission matrix B for storing the prob of a state, resulting in an observed value j
observable is assumed to have K possible values, meaning it could take any one of $\{1, 2, \ldots K\}$

$$\text{From} \begin{bmatrix} b_{00} & b_{01} & \cdots & b_{0K} \\ b_{10} & b_{11} & \cdots & b_{1K} \\ \vdots & & & \vdots \\ b_{m0} & b_{m1} & \cdots & b_{mK} \end{bmatrix} \quad \text{(To)}$$

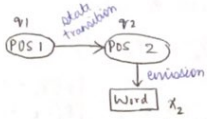
b_{ij} represents the prob of state i to emit observable j.

$$b_{ij} = P(Y_k = j \mid Y_K = i)$$

$$O_k = S_k * B \quad \text{where } O_k = \begin{bmatrix} P(Y_k=1) \\ P(Y_k=2) \\ \vdots \\ P(Y_k=m) \end{bmatrix}$$

POS Tagging, HMM, Viterbi Algo

Generative Model : HMM



here, we want to determine the POS tag for word x_2

q_1 = prev state
q_2 = current state

we want to determine to which category the word emitted by POS 2 belongs to?

HMM is a combo of ST & SE
For ST, we use a bigram [can go for tri/n-gram too]
For SE prob, we are using the bayes rule.

Formula of HMM

$$P[POS_2 \mid (word, POS_1)] = P(word \mid POS_2) * P(POS_2 \mid POS_1)$$

$P(word \mid POS_2)$ ← bayes rule; data is obtained from emission prob matrix

$P(POS_2 \mid POS_1)$ bigram; data obtained from state transition matrix

So, we are determining the POS tag for x_2 based on POS 1

Emission Probability Matrix :

i) From a given corpus, identify unique tokens. Also identify total POS tags present in the corpus.
ii) Assign correct POS tag for every token
iii) Arrange tokens in horizontal & POS tags in vertical fashions. (We can change this order also)
iv) This will provide data for Bayes rule emission prob
v) For N-tokens, & parts of speech tags, size of the emission matrix will be N×M.

State Transition Prob:

i) Use POS tags found from emission prob matrix
ii) Create matrix size of M×M
iii) Use bigram concept & fill the entries in a table
iv) This will provide data for state trans" prob

Need for Viterbi Decoding Algo :

i) its based on DP
ii) DP breakdowns the prob into sub-parts
iii) it saves the result for future purposes
∴ no need to compute the result again

Training Corpus with correct POS tag :

<s> Book a car </s>
<s> park the car </s>
<s> The book is in the car </s>
<s> The car is in a park </s>

Step 1 : Assign correct POS tag to training corpus

<s>	Book	a	car	</s>		
	verb	det	noun			
<s>	Park	the	car	</s>		
	verb	det	noun			
<s>	The	book	is	in	the	car
	det	noun	verb	prep	det	noun
<s>	The	car	is	in	a	park
	det	noun	verb	prep	det	noun

total
verb = 4
det = 6
noun = 6
prep = 2

Step 2 : Creation of Emission Prob Matrix

	Verb (4)	Det (6)	Noun (6)	Prep (2)
1. Book	1/4		1/6	
2. a		2/6		
3. car			4/6	
4. Park	1/4		1/6	
5. the		4/6		
6. is	2/4			
7. in				2/2
Σ	4	6	6	2

$$P(word \mid class) = \frac{P(class, word)}{P(class)}$$

Step 3 : State Trans" Prob Matrix

	Noun	Verb	Det	Prep	</s>	total of how many times count
<s>		2/4	2/4			4
Noun		2/6			4/6	6
Verb			2/4	2/4		4
Det	6/6					6
Prep			2/2			2

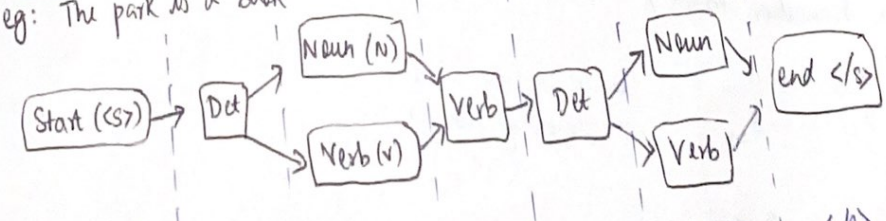
Verb follows <s> twice

</s> occurs after verb 4 times

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

Test Data

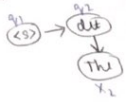
eg: The park is a book



total no. of possibilities = (1 × 2 × 1 × 1 × 1 × 2) = 4

⇒ Viterbi algo is used to find the efficient path

I) The First Word : "The"



$P(q_2 \mid x_2) = P(x_2 \mid q_2) * P(q_2 \mid q_1)$

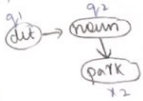
$P[det \mid (the, <s>)] = P(the \mid det) * P(det \mid <s>)$
$= \frac{1}{4} * \frac{2}{4} = \frac{1}{8}$

emission prob; state trans"

II) The Second Word : "Park"

previous prob = $\frac{1}{8}$

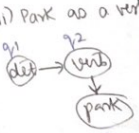
(i) Park as a noun



$P[noun \mid (park, det)] = P(park \mid noun) * P(noun \mid det)$
$= \frac{1}{6} * \frac{4}{6} = \frac{1}{9}$

$P[noun \mid (park, det)] = P($
$=$

(ii) Park as a verb



$P[verb \mid park\ det] = P[$
$=$

sometimes a path 2 states, values we do arithmetic

∴ we multiply

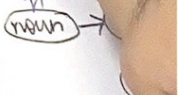
∴ $P(verb \mid park, det) = \frac{1}{4} *$

now, compare $\frac{1}{1200}$ & $\frac{1}{18}$

$\frac{1}{18} > \frac{1}{1200}$ } ∴ park is current

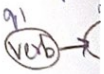
III) The

Prev prob



IV) The fou

Prev prob



I) The First Word: "The"

$P(q_2|x_2) = P(x_2|q_2) * P(q_2|q_1)$

$P[det|(the, <s>)] = P(the|det) * P(det|<s>)$

$= \frac{1}{4} * \frac{2}{4} = \frac{1}{3}$

(emission prob, state trans.)

II) The Second Word: "Park"

previous prob = $\frac{1}{3}$

(i) Park as a noun

$P[noun|(park, det)] = P(park|noun) * P(noun|det)$

$= \frac{1}{6} * \frac{1}{4} = \frac{1}{6}$

$P[noun|(park, det)] = P(noun|park, det) * prev\ prob$

$= \frac{1}{6} * \frac{1}{3} = \frac{1}{18}$

(ii) Park as a verb

$P[verb|park, det] = P[park|verb] * P[verb|det] * prev\ prob$

$= \frac{1}{4} * 0 * \frac{1}{3} = 0$

sometimes a path is efficient but in b/w 2 states, value becomes zero. so that time we do smoothing.

∴ we multiply by (1/100)

$\therefore P(verb|park, det) = \frac{1}{4} * \frac{1}{100} * \frac{1}{3} = \frac{1}{1200}$

Now, compare $\frac{1}{1200}$ & $\frac{1}{18}$

$\frac{1}{18} > \frac{1}{1200}$

noun > verb

∴ park is a noun

current prob = 1/18

for next step, it becomes prev prob

III) The third word: "is"

Prev prob = 1/18

$P[verb|(is, noun)]$

$= P(is|verb) * P(verb|noun) * prev\ prob$

$= \frac{2}{4} * \frac{2}{6} * \frac{1}{18} = \frac{1}{108}$

IV) The fourth word: "a"

Prev prob = 1/108

$P[det|a, verb]$

$= P(a|det) * P(det|verb) * prev\ prob$

$= \frac{2}{6} * \frac{2}{4} * \frac{1}{108} = \frac{1}{648}$

V) The Fifth Word: "book"

Prev prob = 1/648

i) BOOK as a noun

$P(noun|book, det)$

$= P(book|noun) * P(noun|det) * prev\ prob$

$= \frac{1}{6} * \frac{6}{6} * \frac{1}{648} = \frac{1}{3888}$

ii) Book as a verb

$P[verb|book, det]$

$= P(book|verb) * P(verb|det) * prev\ prob$

$= \frac{1}{4} * 0 * \frac{1}{648} = 0$

perform smoothing

$= \frac{1}{4} * \frac{1}{100} * \frac{1}{648} = \frac{1}{259200}$

$\frac{1}{3888} > \frac{1}{259200}$

noun > verb

∴ book is a noun

∴ result of POS tagging after applying HMM

<s>	The	park	is	a	book	<s>
	det	noun	verb	det	noun	

Learning the parameters:

Scenario 1:

- A labelled dataset is available, with the POS category of individual words in a corpus

How to do → parameters can be directly estimated using maximum likelihood estimate from the labelled dataset.

Scenario 2:

- Only the corpus is available, but not labelled with the POS categories

How to do → Baum-Welch algorithm is used to estimate the parameters of the hidden markov model.

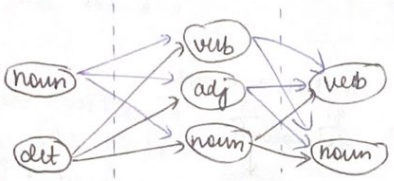
"The light book"

P(the | det) = 0.3
P(the | noun) = 0.1
P(light | noun) = 0.003
P(light | adj) = 0.002
P(light | verb) = 0.06
P(book | noun) = 0.003
P(book | verb) = 0.01

P(word | tag) = symbol emission

P(verb | det) = 0.00001
P(noun | det) = 0.5
P(adj | det) = 0.3
P(noun | noun) = 0.2
P(adj | noun) = 0.002
P(noun | adj) = 0.2
P(noun | verb) = 0.3
P(verb | noun) = 0.3
P(verb | adj) = 0.001
P(verb | verb) = 0.1

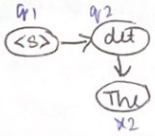
$P(tag_1 | tag_2)$ = state transition



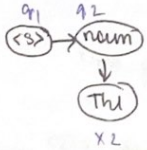
The | light | book

total tags = 4
tag prob = 1/4

I) The First Word : "The"



P[det | The, <S>]
= P(The | det) * P(det | <S>)
= 0.3 * ~~1111~~ 1/4 (prob of det)
= 0.0075

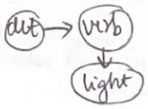


P[noun | The, <S>]
= P(The | noun) * P(noun | <S>)
= 0.1 * ~~1111~~ 1/4 (prob of noun)
= 0.025

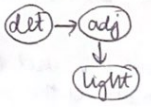
0.0075 > 0.025
det > noun
∴ The = det

II) The second word : "light"

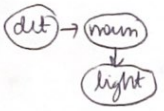
prev prob = 0.0075



P[verb | light, det]
= P(light | verb) * P(verb | det) * prev prob
= 0.01 * 0.00001 * 0.0075
= 0.00000045



= P[adj | light, det]
= P(light | adj) * P(adj | det) * prev prob
= 0.002 * 0.3 * 0.0075
= 0.0000045

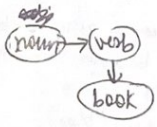


= P[noun | light, det]
= P(light | noun) * P(noun | det) * prev prob
= 0.003 * 0.5 * 0.0075
= 0.00001125

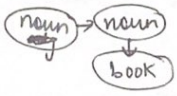
0.00001125 > 0.0000045 > 0.00000045
noun > adj > verb
∴ light = ~~adj~~ noun

III) The third word : "book"

prev prob = 0.00001125



P[verb | book, noun]
= P(book | verb) * P(verb | noun) * prev prob
= 0.01 * 0.3 * 0.00001125
= 0.00000034



P[noun | book, noun]
= P(book | noun) * P(noun | noun) * prev prob
= 0.003 * 0.2 * 0.00001125
= 0.00000007

0.00000034 > 0.00000007
∴ verb > noun

The	light	book
↓	↓	↓
det	~~noun~~ adj	verb