Lecture # 02

NLP is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages.

⇒ Identify the structure and meaning of words, sentences, texts and conversations.

Why NLP is hard?

- Ambiguity - Word sense ambiguity
- Propositional Phrase (PP) - attachment ambiguity
- Language is not static
- Language is compositional

Key NLP components

→ Parts of speech tagging
→ Syntactic Parsing
→ Semantic Analysis

Parsing → means taking an input and producing some sort of linguistics structure for it.

- Morphological
- Syntactic
- Semantic
- Discourse
- In the form of a
- Tree or a network
- string

Morphology → Study of way words are built up from smaller meaning-bearing units, morphemes.

fox → fox (single morpheme) cats → (cat, s) → two morphemes

⇒ Two broad classes - Stem or Root and Affixes

Roots or Stem → The central morphemes in words, which carry the main meaning.

cats → cat + s
cat → root
s → affix

Affix → add additional meaning of various kinds
Prefixes → preceed the stem (unbuckle)
suffixes → follow the stem (cats)
circumfixes → do both (enlighten)
Infixes → inside the stem (hingi, humingi)
↳ Tagalog word

→ A word can have more than one affixes.
↳ unbelievably → -un, -able, -ly

Turkish → 9 or 10 affixes coucatenatid. English does not have this.

Agglutinative → Languages that tend to string affixes together like Turkish does are called agglutinative.

Word Creation using Morphemes

1) Inflection → Combination of a word stem with grammatical morpheme, usually resulting in a word of the same class as the original stem, and usually filling some syntactic function like agreement.

-s, -ed → inflectional morphemes

2) Derivation → Combination of word stem with grammatical morpheme, usually resulting in a word of different class, often with a meaning hard to predict exactly.

computerize → computerization
-ation

3) Compounding → combination of multiple word stems together
↳ doghouse → dog + house

4) Cliticization → Combination of word stem with a clitic.
Clitic → morpheme that acts syntactically like a word, but is reduced in form and attached (phonologically or sometimes orthographically) to another word.
→ 've , 'nt etc

Irregularity

Sometimes, inflectional marking differs depending on root/base.
walk: walked, walked :: sing: sang: sung

Semantic irregularity / unpredictability
→ Same derivational morpheme may have different meaning/functions depending on base it attaches to.
→ a kind-ly old man
→ a slow-ly old man

Inflection morphology makes instances of the same word appear to be different words.

Morphology in NLP → largely solved
∴ rule based method: finite state methods
Other solutions
→ supervised, Seq-to-Seq, Unsupervised

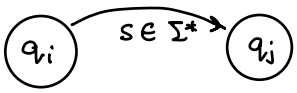
Morphological analysis → map from orthographic form to lexical form vice versa is called morphological generation

Morphological Parser

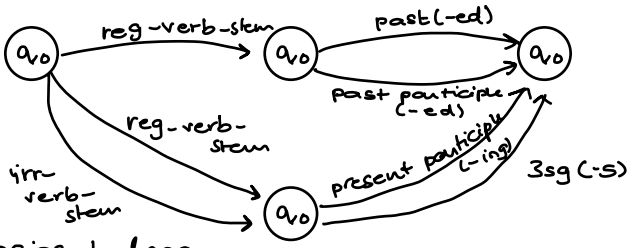
- Lexicon → The list of stem and affixes, together with basic information about them (stem is N or V)
- Morphotactics → the model of morpheme ordering that explains which classes of morphemes can follow other class of morpheme inside a word.
- Orthographic Rules → These spelling rules are used to model the changes that occur in a word

Finite State Automata (FSA)

Q: set of finite states
$q_0 \in Q$: a special start state
$F \subseteq Q$: a set of final states
Σ: a finite alphabet
Transitions:



Example

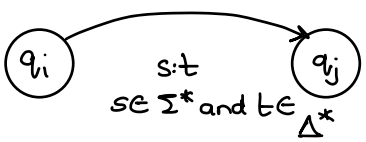


→ Most natural language morphologies belong to regular language.

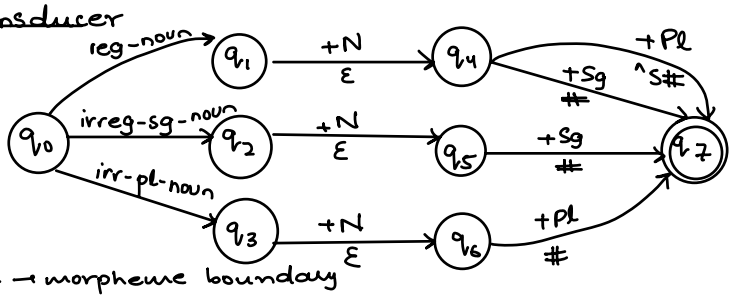
Finite State Transducers

→ Type of finite automaton that maps between two sets of symbols.

Q: a finite set of states
$q_0 \in Q$: a special start state
$F \subseteq Q$: a set of final states
Σ, Δ: two finite alphabets



Transducer



^ → morpheme boundary
# → word boundary

Morphological analyzer → divided into two stages
→ Morphotactics → Allomorphic/Orthographic rules

- Morphotactics:
  - Maps between fox+N+Pl and fox^s#
  - Concats basic forms of morphemes together.
  - Lemmas concatenated with affixes
- Allomorphic/Orthographic rules:
  - Maps b/w output of morphotactics & surface rep.
  - ⇒ fox^s# → foxes

Operations on FSTs
- Composition, concatenation, Kleene closure, Union, Intersection (FSTs are not closed under intersection)

Rules

Name	Description	Example
Consonant doubling	1 letter consonant doubled before -ing/-ed	beg / begging
E deletion	Silent E dropped before -ing & -ed	make / making
E insertion	e added after -s, -z, -x, -ch, -sh before -s	watch / watches
Y replace	-y → -ie before -s, -i/-ed	try / tries
K insert	verbs ending with vowel + c	panic / panicked

Stemming

→ applies to many affixes other than plurals
⇒ Given a word, returns the stem of the word.

Lecture #05

I do uh main- mainly business data processing

- The broken-off word main- is called a fragment
- words like -uh and -um are called fillers

Conditional Prob.

$$P(x|y) = \frac{P(x,y)}{P(y)}$$

Product Rule

$$P(x,y) = P(x|y)P(y)$$

Chain rule

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\dots$$
$$= \prod_{i=1}^{n} P(x_i | x_1, \dots, x_{i-1})$$

X, Y independent if ? if $\forall x,y : P(x,y) = P(x)P(y)$

X, Y conditionally independent given Z $X \perp\!\!\!\perp Y | Z$

$$\forall x,y,z : P(x,y|z) = P(x|z)P(y|z)$$

Probability of word

Our goal is to compute the probability of a word w given some history h or $P(w|h)$

$$P(x_1, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1^2)\dots P(x_n|x_1^{n-1})$$
$$= \prod_{k=1}^{n} P(x_k|x_1^{k-1})$$

Apply to words

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

w_1^n ↓ sequence of n words

Unigram → no history

$$P(w_i^n) = \frac{c(w_i^n)}{\sum_w c(\tilde{w})}$$

Bigram → approximates the probability by using only the conditional probability of preceding word $P(w_n|w_{n-1})$

Markov Models

In bigram, we are making the following assumption:

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

This is known as the Markov assumption

↳ class of probabilistic models that assume that we can predict the prob. of some future unit without looking too far into the past.

N-gram

$$P(w_n|w_1^{n-1}) = P(w_n|w_{n-N+1}^{n-1})$$

Maximum Likelihood Estimation (MLE)

MLE estimates the parameters of N grams model by counts from a corpus normalizing so that they lie b/w 0 & 1.

To compute the prob. of a word y given x, we compute count of bigram C(xy) and normalize by sum of all bigrams that share the same first word x:

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

To simplify,

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

MLE N-gram Estimation

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$ → relative frequency

Ex.

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs & jam </s>

$P(I|<s>) = 2/3 = 0.67$
$P(Sam|<s>) = 1/3 = 0.33$
$P(am|I) = 2/3 = 0.67$

To compute prob.

Ex.

P(<s> I want English food </s>)
P(I|<s>) P(want|I) P(English|want) P(food|English) P(</s>|food)

N-gram model is very dependent on training corpus.
⇒ substitute unknown word <UNK> in training & solve.

Perplexity (PP)

↳ most common intrinsic evaluation metric for N-gram language models.

Intuition → better one is the one that predicts the details of test data better.

PP of a language model on test set is the function of the probability that language model assigns to that test set.

For a test set $W = w_1, \ldots, w_N$, the PP is the probability of the test set, normalized by number of words.

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

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 \cdots w_N)}}$$

Using Chain rule,

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

→ The higher the conditional prob, the lower the perplexity.
→ Thus minimizing perplexity is equivalent to maximizing the test set probability.

Issue with MLE

Sparse data: If test set has N-gram that was never in the training set, the MLE prob for this N-gram is zero and also 0 for whole test set.

⇒ So, modify MLE to assign non-zero probability to N-gram not observed in training.

Smoothing — Method to assign non-zero probability to N-gram that was not observed in training.

Laplace Smoothing
→ Add to all counts

Unsmoothed MLE : $P(w_i) = \frac{c_i}{N}$

Smoothed MLE : $P(w_i) = \frac{c_i+1}{N+V}$

} unigram probabilities

The adjusted count:

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

Lecture # 6

Part-of-Speech Tagging

Input: a sequence of word tokens w
Output: a sequence of part-of-speech tags t, one per word

PoS has two broad categories:

closed class: Closed classes are those that fixed membership. For example, prepositions are a closed class because there is a fixed set of them in English; new prepositions are rarely coined in English.

open class: By contrast noun and verbs are open classes because new nouns and verbs are continually coined or borrowed from other languages.

Hidden Markov Model PoS Tagging

- HMM is a sequence model
- A sequence or sequence classifier is a model whose job is to assign a label or class to each unit in a sequence.

HMM is a probabilistic sequence model: given a sequence of units (words, letters, morphemes, sentences), it computes a probability distribution over possible sequence of labels and chooses the best label sequence.

Markov Chain

A model that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set.

Set → words, tags, etc

Assumption: If we want to predict the future in the sequence, all that matters is the current state.

A sequence of state variables: $q_1, q_2, \dots, q_i$

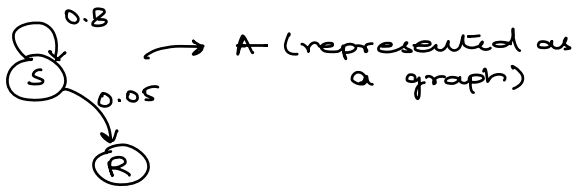
$$P(q_i \mid q_1, \dots, q_{i-1}) = P(q_i \mid q_{i-1})$$

Formally, a markov chain is defined by following:

- ✓ $Q = q_1 q_2 q_3 \dots q_n$ set of N states
- ✓ $A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$ A transition prob. matrix. Each a_{ij} represents prob. of moving from one state i to another j
  $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
- ✓ $\pi = \pi_1 \pi_2 \dots \pi_n$ An initial prob. distribution over states. π_i is the probability that Markov chain will start in state i, some states $\pi_j = 0$ meaning it cannot be start state.
  $\sum_{i=1}^{n} \pi_i = 1$

Ex. Sunny, next Sunny & Rainy?

$$P(q_2, q_3 \mid q_1) = P(q_2 \mid q_1) P(q_3 \mid q_2)$$
$$= P(S \mid S) P(R \mid S)$$
$$= (0.8)(0.05)$$
$$= 0.04$$



A Markov chain is useful when we need to compute prob. for a seq. of observable events.

- In many cases, events we are interested in are hidden:
  → PoS tags not in text
- A HMM allows us to talk about both observed events and hidden events.

HMM is explained by following:

- ✓ $Q = q_1 q_2 q_3 \dots q_n$ set of N states
- ✓ $A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$ A transition prob. matrix. Each a_{ij} represents prob. of moving from one state i to another j
  $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
- ✓ $\pi = \pi_1 \pi_2 \dots \pi_n$ An initial prob. distribution over states. π_i is the probability that Markov chain will start in state i, some states $\pi_j = 0$ meaning it cannot be start state.
  $\sum_{i=1}^{n} \pi_i = 1$
- ✓ $O = o_1 o_2 \dots o_T$ A sequence of T observations each drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
- ✓ $B = b_i(o_t)$ A sequence of observation likelihood also called emission probabilities, each expressing prob. of an observation o_t being generated from state q_i

2 assumptions

- The probability of a particular state depends only on previous state
  $$P(q_i \mid q_1 \dots q_{i-1}) = P(q_i \mid q_{i-1})$$

- The prob. of an output observation o_i depends only on state that produced observation q_i and not on any other state or any other observation.

$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$$

Emission probabilities → Prob. of making certain observations given a particular state.

Transition probabilities → Prob. of transitioning to another state given a state

HMMs for PoS Tagging

To model any problem as HMM, we need a set of observations and a set of possible states (hidden).

→ In PoS, observations are words
→ States are PoS tags
→ Transitional prob ⇒ P(VP|NP) etc.
→ Emission prob ⇒ P(John | NP)

The matrix A contains the tag transitional prob. $P(t_i | t_{i-1})$

MLE ⇒ $P(t_i | t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$

$P(w_i | t_i) = \frac{C(t_i, w_i)}{C(t_i)}$

Likelihood Problem

→ Given an HMM $\lambda = (A, B)$ and observation sequence O, determine likelihood $P(O|\lambda)$.

Decoding Problem

→ Given an HMM $\lambda = (A, B)$ and observation sequence O, determine the best hidden state sequence Q.

Lecture # 7

Input: a sequence of word tokens x
Output: a sequence of PoS tags y, one per word

HMM Sol: Find the most likely tag sequence, given the word sequence

Decoding

For any model, containing hidden variables, the task of determining the hidden variable sequence corresponding to the sequence of observations is called decoding.

Decoding: Given $\lambda = (A, B)$ and sequence of observations $O = o_1, o_2, \dots, o_T$, find the most probable seq. of states $Q = q_1, q_2, \dots, q_T$

For PoS, decoding goal is to choose tag sequence $\hat{t}_1^n$ given words w_1^n:

$$\hat{t}_1^n = \text{argmax}\, P(t_1^n | w_1^n)$$

HMM uses Bayes rule:

$$\hat{t}_1^n = \text{argmax} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$$

Dropping the denominator,

$$\hat{t}_1^n = \text{argmax}\ P(w_1^n | t_1^n) P(t_1^n)$$

Two assumptions:

- prob. of word depends only on its tag:

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

- Prob. of next tag is dependent only on prev tag

$$P(t_1^n) = \prod_{i=1}^{n} P(t_i | t_{i-1}) \quad \text{emmission} \quad \text{transition}$$

$$\hat{t}_1^n = \text{argmax}\ P(t_1^n | w_1^n) = \text{argmax} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$$

Extending HMM to Trigrams

In practice, we use more of history:

$$\Rightarrow P(t_1^n) \simeq \prod_{i=1}^{n} P(t_i | t_{i-1}, t_{i-2})$$

→ small increase in performance, significant change in Viterbi algo.

The tagger knows the end of sentence by adding dependence on end-of-sentence marker for t_{n+1}.

$$\hat{t}_1^n = \text{argmax}\ P(t_1^n | w_1^n) = \underset{t_1^n}{\text{argmax}} \left[\prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1}, t_{i-2}) \right] P(t_{n+1} | t_n)$$

MLE can be compuiid as:

$$\hat{P}(t_i | t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})} \rightarrow \text{trigram}$$

$$\hat{P}(t_i | t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \rightarrow \text{Bigram}$$

$$\hat{P}(t_i) = \frac{C(t_i)}{N} \rightarrow \text{unigram}$$

$$\hat{P}(t_i | t_{i-1}, t_{i-2}) = \lambda_3 \hat{P}(t_i | t_{i-1}, t_{i-2}) + \lambda_2 \hat{P}(t_i | t_{i-1}) + \lambda_1 \hat{P}(t_i)$$

$\lambda_3 + \lambda_2 + \lambda_1 = 1$ → to ensure prob. dist.