NLP is hard due to ambiguity, variability,scarce data,robustness, context dependence,unknown representation,language diversity. Probabalistic modelling is useful.

Num of parses with CFG and Syntactic ambiguity is the catalan num. Catalan:

Zipf’s law (reason for sparse data):

Accuracy: Proportion model gets right:

P Precision (proportion of model’s answers that are right):

R Recall (proportion of test data that model gets right):

F1 F1-score: Harmonic mean of precision and recall. For isolating performance on a particular label in multi-label tasks,or for chunking,phrase structure parsing, or anything where word-by-word accuracy isn’t appropriate.

Significance Test

Parametric when the underlying distribution is normal / Gaussian (∗): – t-test, z-test Non-Parametric otherwise.-Usually do need non-parametric tests:The Gaussian distribution is often not applicable-Can use McNemar’s test or variants of it- Stochastic/pemutation tests are a convenient alternative(esp. with

complex predictions, such as parse trees)

Effect size δ(x):

a random variable X^ ranging over all test sets. H0 = Null Hypothesis

this p-value is the prob that we would see δ(x) assuming A is not better than B. It is common to use values like .05 or .01 as the thresholds. A value of .01 means that if the p-value (the prob of observing the δ we saw assuming H0 is true) is less than .01, we reject the null hypothesis and assume that A is indeed better than B. In NLP we usually use non-parametric tests based on sampling: we artificially create many versions of the experimental setup. For example, if we had lots of different test sets x’ we could just measure all the δ(x’) for all the x’. That gives us a distribution. Now we set a threshold (like .01) and if we see in this distribution that 99% or more of those deltas are smaller than the delta we observed, i.e., that p-value(x)—the prob of seeing a δ(x) as big as the one we saw—is less than .01, then we can reject the null hypothesis and agree that δ(x) was a sufficiently surprising difference and A is really a better algorithm than B.

Bootstrap testing: Calculate δ(x). Perform trials by creating test sets sampled randomly with replacement and seeing the results of the models at the end of a trail by calculating δ(x)i. In a simple case, the p-value would be where δ(x)i ≥ 2 δ(x) summed over all trails & then divided by the num of trials.

Common p-values: 0.05, 0.01, 0.001

0.05 -> 5% of the time

MLE:

C(x) = count of x/event x in dataset

Problem w/ MLE: can assign prob of 0 on grammatical correct sentences – sparse data problem.

N-gram model: Reduces sparse data problem.

Remember: P(X,Y) = P(Y|X)P(X)

Generic case with m being the num of words checked before the cur word:

Unigram:

Trigram:

To capture behaviour at beginning/end of sequences, add <s> to start of and </s> to end of .

We typically use negative log probs. Range from 0 to ∞. Lower cost = higher prob. Instead of multiplying, add neg log probs.

Evaluation

Extrinsic: measure performance on a downstream application. The most reliable evaluation,but can be time-consuming.

Intrinsic: design a measure that is inherent to the current task. Can be much quicker/easier during development cycle. But not always easy to figure out what the right measure is: ideally, one that correlates well with extrinsic measures.Entropy: a measure of uncertainty/disorder. Gets smaller as an event becomes more likely over all others. Average number of bits needed to encode X ≥ entropy of X. The expected value of -log2P(X) -->

Cross Entropy: Used when actual prob not known. Measures how close is to true P Cross Entropy ≥ Entropy. Lower cross-entropy ⇒ model is better at predicting next word. Cross-entropy depends on both the model and the corpus.

Estimating Cross Entropy: For w1, … wn with large n, per-word cross-entropy is well approximated by:

This is the avg negative log prob our model assigns to each word in the sequence. (i.e., normalized for sequence length).

Perplexity: LM performance is often reported as perplexity rather than cross-entropy. The average branching factor at each decision point, if our distribution were

uniform. 2cross-entropy

Smoothing: reduces sparse data problem. Smoothing methods below assign equal prob to all unseen events if interpolation and back-off are not used.

Add-One (Laplace) Smoothing: pretend we saw everything one more time than we did and adjust the denominator accordingly. Bad for sets with large vocab size. Assumes we know the vocab size – just use UNK. Below, v = vocab size.

Add-α (Lidstone) Smoothing: α < 1 in

Train model (estimate probs) on training set with different values of

α. Choose the α that minimizes cross-entropy on development set. Assumes we know the vocab size – just use UNK.

Good-Turing Smoothing: adjust the count on the numerator. c = actual count c\* = adjusted count.

we only know N0 if we actually know what’s missing and we can’t always estimate what words are missing from a corpus. But for bigrams, we often assume N0 = V2−N, where V is the different (observed) words in the corpus. The sum of all discounts is Assumes we know the vocab size – just use UNK. Doesn’t allow “holes” in the counts (if Ni >0, Ni+1 >0) can estimate using linear regression. Applies discounts to high-frequency items

Good-Turing justification: If word not seen before, aka , divide that prob equally amongst all unseen events.

if word already has 1 count, aka  
, Divide that prob equally amongst all 1-count events.

Kneser-Ney Smoothing: takes diversity of histories into account. Diversity of histories being when n-gram models, excluding a unigram, become biased towards a specific set of words that often follow on from one another. As the unigram is unaffected, words will have a much higher prob compared to,say, the bigram probs. This smoothing aims to reduce this problem. Until recently the best smoothing method for word n-grams.

Count of distinct histories for a word:

Interpolation: Combine higher and lower order N-gram models, since they have different strengths and weaknesses:

– high-order N-grams are sensitive to more context, but have sparse counts

– low-order N-grams have limited context, but robust counts

If PN is N-gram estimate (from MLE, GT, etc; N = 1 − 3), i.e is a unigram, use:

λis must sum to 1

interpolation parameters are also called mixture weights as any weighted combination of distributions is called a mixture model. The values of the λis are chosen to optimize perplexity on a held-out data set.

Katz Back-Off: Good-Turing discount the

observed counts, but instead of distributing that mass uniformly over unseen items, use it for backoff estimate. Discount the trigram-based prob estimates. This leaves some prob mass to share among the estimates from the lower-order model(s).

Back-Off Formulae: Where is the adjusted prediction model and are the backoff weights.

Word similarity: Early version: class-based language models. Recent version: distributed language models

Distributed: words represented as high-dim vectors with similar words having similar vectors. Use ML methods like neural networks that iteratively improve embeddings. Can be time-consuming (hours-days). Learned embeddings capture semantic and syntactic similarity.

Noisy channel model: Imagine input has some noise to it. Aim is to predict what the input was supposed to say before, for example, spelling mistakes.

A picture containing text, font, screenshot, number

Description automatically generated

In spelling correction, P(Y) is what the user intended to type, P(X|Y) is the dist describing what user is likely to type, given what they mean in P(Y) (including typos due to key position or common spelling errors). P(X) is the dist of what we see. We want to recover the most probable y given observation x.

Noisy channel as probabilistic inference:

As P(x) is fixed ^

If we can train P(X|Y), why can’t we just train P(Y|X)? Who needs Bayes’

Rule? - training P(X|Y ) or P(Y |X) requires input/output pairs, which are often limited: Misspelled words with their corrections; transcribed speech; translated text. But LMs can be trained on huge unannotated corpora: a better model. Can help improve overall performance.

Basic spelling correction system: correct each non-word xi by generating a list of all words yi that differ by 1 character from xi

and computing for each and returning the with highest value. A simple model could have substitution(o ->e), deletion(t-> -),insertion( - ->u). Assume that the typed character xi depends only on intended character yi then  
Example: P(no|not) = P(n|n)P(o|o)P(-|t)To estimate , count how many times each character (including empty character for del/ins) was used in place of each other character -> confusion matrix. Use MLE or smoothing to estimate probs

Alignments and edit distance: find the optimal character alignment between two words. Possible alignments: A picture containing font, text, number, line

Description automatically generated

Minimum Edit Distance:

|  |  |
| --- | --- |
| Oper | Cost |
| Delete | 1 |
| Insert | 1 |
| Subst | 2 |
| None | 0 |

A picture containing text, handwriting, font, line

Description automatically generated Using dynamic programming, Strings of length n and m require O(mn) time and O(mn) space to find optimal alignment using edit distance.

Default costs of MED. Usually use noise model costs.^

A picture containing text, line, number, font

Description automatically generated

Table above stores 2 things.

-D(stall[0..i],table[0..j]): the MED of substrings of length i, j

- Backpointer(s): which sub-alignment(s) used to create this one.

You sum the vals. First column is all deletions, first row is all insertions. Pick the minimum cost for each box. If there is a tie for an operation, you can list them all. Costs can change to suit the problem!

Likelihood Function: θ = params of model. For our spelling error model, θ is the set of all character rewrite probs P(xi|yi). For any value of θ, we can compute the prob of our dataset P(data|θ).

This is the likelihood. If our data includes hand-annotated character alignments, then P(data|θ) =. If the alignments a are latent, sum over possible alignments:

P(data|θ) =. The likelihood P(data|θ) is a function of θ, and can have multiple local optimaA picture containing line, plot, diagram, font

Description automatically generated

EM will converge to one of these; hard EM might not. Neither is guaranteed to find the global optimum!

Text Classification: For categorizing the author or content of the text. N-gram models can be used for classification. Word order matters less for many tasks, so we can just consider the document as a bag of words. On the other hand, N-gram models miss other features like part of speech tags. Instead, consider Naïve Bayes and Max Entropy.

Features: Words, binary values for fi: did this word occur in d or not, Use only a subset of the vocabulary for F (i.e ignore stopwords or choose a small task-relevant set), or even use more complex features!

Bag of words: A paper bag with text on it

Description automatically generated with low confidence

Naïve Bayes: A generative model. Can both discriminate correct vs incorrect + generate data by sampling a class from P(c), then sampling

words from P(x|c). Features are directly observed (e.g., words in doc): no difference between features and data. Given document d and set of categories C (say, spam/not-spam), we want to

assign d to the most probable category .P(d) is fixed - > can be removed from equation. Just as in spelling correction, we need to define P(d|c) and P(c). To find P(d|c), define a set of features that might help classify docs. We then represent each document d as the set of features (words) it contains: f1, f2, . . . fn. So . As in LMs, we can’t accurately estimate P(f1, f2, . . . fn|c) due to sparse data.

Naïve Bayes Assumption: To solve sparse data problem, assume features are conditionally independent given the class. -->

That is, the prob. of a word occurring depends only on the class. Not on which words occurred before or after (as in N-grams) or even which other words occurred at all. Effectively, we only care about the count of each feature in each document.

Naive Bayes classifier: Given a doc with features f1 (as a recap, features can be WORDS!), f2, . . . fn and set of categories C, choose the class where  
where P(c) is the prior prob of class c before observing any data and P(fi|c) is the prob of seeing feature fi in class c. P(c) normally estimated with MLE - where Nc = the num of training docs in class c and N = the total num of training docs. = proportion of training docs belonging to class c. P(fi|c) normally estimated with simple smoothing:

Where count(fi,c) = the number of times fi occurs in class c, F = the set of possible features, α = the smoothing parameter, optimized on held-out data. Thus when classifying a doc, you need to calculate the prob of the doc belonging to each class.

NAÏVE BAYES with Costs and Linearity: Naive Bayes is called a linear classifierTo avoid multiplying small probs which would cause problems, Naïve Bayes uses costs – costs are negative log probs that must be summed rather than multiplied and then look for the lowest cost overall. Using costs, our Naive Bayes equation looks like this:

if classes are fairly well-balanced, accuracy is fine to use. Report the % of correct classification decisions. However, if (say) 95% of documents belong to class A, it’s easy (but not

useful) to get 95% accuracy by always guessing A.

Advantages of Naïve Bayes:

Easy to implement, fast to train, works reasonably well, doesn’t need a large training set. Should be baseline method for any classification task.

Disadvantages of Naïve Bayes: Doesn’t work as well on features that are actually dependent given the category due to its assumption. Features are not usually independent given the class. Adding multiple feature types (e.g., words and morphemes) often leads to even stronger correlations between features. Accuracy of classifier can sometimes still be ok, but it will be highly overconfident in its decisions as 5 features that all point to class 1 will be treated as 5 different pieces of evidence.

Self-supervised learning: annotated texts are scarce compared to unannotated ones so we should try to train a model without needing to annotate things ourselves.

“self training” :

1. Train NB on labeled data alone

2. Predict labels on on unlabelled data

3. Re-estimate NB (in the usual way), but now also using self-labelled data

EM for Semi-supervised Learning: Self-training for NB is known as “hard EM”

1. Train NB on labeled data alone

2. Make soft prediction on on unlabelled data (”E-step”)

3. Recompute NB parameters using the soft counts

Max Entropy: A discriminative model. Not all discriminative models are probabalistic but this is probabalistic. It can ONLY be trained to discriminate correct vs. incorrect values of c, given input x. Does not make a conditional independence assumption and therefore can handle features that are not independent given the class unlike Naïve Bayes. Also known as multinomial logistic regression. Like naïve bayes, goal: however Max Ent doesn’t apply Bayes Rule – directly models P(c|d). We will use to represent the observed

data. Features are functions that depend on both observations and class c. Each feature fi has a real-valued weight wi (learned in training). If we have three classes, our features will always come in groups of three. For example, we could have three binary features:

f1 : contains(‘ski’) & c = 1

f2 : contains(‘ski’) & c = 2

f3 : contains(‘ski’) & c = 3

– training docs from class 1 that contain ski will have f1 active; – training docs from class 2 that contain ski will have f2 active…

Classification with MaxEnt: Choose class w/ highest prob using:

Where normalisation constant is

Inside brackets is dot product. . Choose class where dot product is highest. is a monotonic function of this dot product. In practice, features are usually defined using templates

contains(w) & c

header contains(w) & c

header contains(w) & link in header & c

Instantiate with all possible words w and classes c then you can filter out features occurring very few time.

Training Max Ent: given items x(1) . . . x(N) with labels c(1) . . . c(N), choose called conditional maximum likelihood estimation (CMLE). CMLE can overfit so must regularise.

Gradient Descent:

Where Random is a random initialisation from, say, Gaussian dist. Where η = learning rate, a scalar

regulating how much you update on every example.

Mini-batch gradient descent: Like grad desc but does not sum over all N on each iteration, only a batch B that is a subset of N.

Gradient of Max Ent: If the classifier is already confident, gradient is close to 0 and no learning is happening. Check Max Ent for equations of and Z. Let us consider a single component of the gradient, corresponding to a feature

Where and

Where

Disadvantages of Max Ent Models: Supervised CMLE in MaxEnt model not as east as supervised MLE in generative models. Requires multiple iterations over the data to gradually improve weights and each iteration computes for all j, and all possible which can be time consuming given a large num of classes and/or features to extract from each example.

Less robust than NB if the test data is different from the training data as it depends on the presence of other predictive features vs NB that relies on all features.

A picture containing text, diagram, font, line

Description automatically generatedMorphology: study of the structure of words. English has relatively impoverished morphology vs say, Turkish that is an agglutinative language: tends to construct complex words by concatenating morphemes, each expressing some simple concept. Small parts of words make more complex words that might be expressed using multiple in English. In English, ‘Whole’ words constructed by combining Stems (house, combine, eat, walk…) , the ‘dictionary’ bit, and Affixes (prefixes, suffixes, infixes and circumfixes), grammar parts. The type of affix is determined by where it goes with respect to the stem. Prefix – before the stem, Suffix – after the stem, Infix – middle of stem, Circumfix – “reduction” of stem. Morphology can be concatenative or non-concatenative (e.g. templatic morphology as in Arabic). English has concatenative morphology. Combining stem + affix: Inflection (stem+grammar affix): no change to grammatical category (walk→walking), Derivation (stem+grammar affix): change to grammatical category (combine→ combination), Compounding (stems together): doghouse, Cliticization: I’ve, we’re, he’s

Inflection: In English, nouns are inflected for number; verbs for person and tense: book(1)/ books(>1), you read (2nd pers. present or past),she reads (3rd pers. present),she read(3rd pers. past). In German, nouns inflected for number and case.

Agglutination and compounding: Example: ostoskeskuksessa

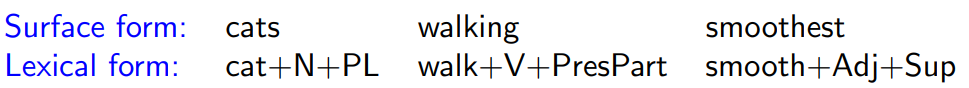
ostos#keskus+N+Sg+Loc:in shopping#center+N+Sg+Loc:in

‘in the shopping center’ (Finnish)

Compounding def: when two or more words are combined together to form a new word.

Morphological parsing: problem of extracting the lexical form from the surface form. Any NLP tasks involving grammatical parsing will usually involve morphology

parsing as a prerequisite. Why not just list all derived forms separately in our wordlist? Not good for morphologically rich languages where there are 1000s of forms for some verbs. Morphological parsing makes adding/learning new words easier (and it makes POS tagging easier) and in speech processing, word breaks aren’t known in advance. As English has concatenative morphology, words can be made up of a main stem plus one or more affixes carrying grammatical information. Surface form: word, Lexical form: word broken down into its more basic components.

 For morphological parsing, we should take account of: Irregular forms, Systematic rules (e.g. ‘e’ inserted before suffix ‘s’ after s,x,z,ch,sh:fox → foxes), things that look like affixes but aren’t (proactive vs. protect), blocking: semi-productivity of morphological rules: N+ful 7→ Adj (graceful, pityful . . . ), but \*intelligenceful (intelligent), \*gloryful (glorious)...

Finite-state machines are a good way to model morphology as although morphemes are tacked together in a rather ‘regular’ way, there are no long range dependencies (in contrast to sentential syntax).

Parsing: going from the surface to the lexical form.

Generation: going from the lexical to the surface form. Often useful to proceed via an intermediate form, corresponding to an analysis in terms of morphemes (= minimal meaningful units) before orthographic rules are applied. A picture containing text, font, screenshot, line

Description automatically generated

Nondeterministic Finite State Automatons (NFAs): finite state machine where a state can have more than one outgoing arc. Below captures (0|1)∗1(0|1)2:

A picture containing diagram, circle, font, design

Description automatically generatedAn ε-NFA allows an input (in parsing) or output (in generation) defined by an arc to be the empty string. ε-NFAs over an input alphabet Σ can capture transitions that (optionally)

A picture containing diagram, sketch, drawing, circle

Description automatically generatedproduce output symbols (over a possibly different alphabet Π). Side: input alphabet Σ = {a, b}, output alphabet Π = {0, 1}. Such a thing is called a finite state transducer. In effect, it specifies a (possibly multi-valued) translation from one regular language to another: – abba 7→ 00, aababa 7→ 0010 . . . More than one output can be possible (e.g., an arc labelled a : 0, 1)Formal definition of finite state transducer: A finite state transducer T with inputs from Σ and outputs from Π consists of: – states Q, S (for start), F (for ‘finish’) – a transition relation ∆ ⊆ Q × (Σ∪{ε}) × (Π∪{ε}) × Q. This defines a many-step transition relation ∆ˆ ⊆ Q×Σ\*× Π\*×Q

– (q, x, y, q’) ∈ ∆ˆ means “starting from state q, the input string x can be translated into the output string y, ending up in state q”. A finite state transducer can be run in either direction! From T you can obtain another transducer T by swapping inputs and outputs. FSTs can be cascaded: output from one can be input to another. For generation, deterministic. For parsing, frequently non-deterministic. Converting from lexical to intermediate form: Use a transducer with a specific form that can handle the different lexical add-ons for regular nouns copied to output, irregular nouns copied to output, and irregular nouns replaced by plural.

Converting from intermediate to surface form:

A picture containing text, font, screenshot, line

Description automatically generatedTo convert a sequence of morphemes to surface form, we apply a number of orthographic rules: – E-insertion: Insert e after s,z,x,ch,sh before a word-final morpheme -s. (fox → foxes) – E-deletion: Delete e before a suffix beginning with e,i. (love → loving) – Consonant doubling: Single consonants b,s,g,k,l, m,n,p,r,s,t,v are doubled before suffix -ed or -ing. (beg → begged). We shall consider a simplified form of E-insertion, ignoring ch,sh. This rule is oblivious to whether -s is a plural noun suffix or a 3rd person verb suffix! Here ? may stand for any symbol except z,s,x,ˆ,#. (Treat # as a ‘visible space character’.)

A picture containing diagram, drawing, font, line

Description automatically generated

Porter Stemmer: a lexicon-free method for getting the stem of a given word: – ATIONAL → ATE (e.g., relational → relate) ING → ε if stem comtains a vowel (e.g. motoring → motor) SSES → SS (e.g., grasses → grass) • Makes errors: – organization → organ (computerization→ computer) policy → police (juicy → juice)

Part-of-speech Tagging: Given a string, identify parts of speech (syntactic categories): This/DET is/VB a/DET simple/ADJ sentence/NOUN. POS tagging is a first step towards syntactic analysis and has simpler models that are often faster than full parsing, but sometimes enough to

be useful. POS tags can be useful features in text classification or word sense disambiguation. Corpus annotators decide the num of parts of speech to look for whether linguistic or practical. Commonly used tagsets for English usually have 40-100 tags. Morphologically rich languages often have compound morphosyntactic tags (Noun+A3sg+P2sg+Nom) predicting these requires more complex methods as there are loads of combos. Hard due to ambiguity and sparse data. The problem of finding the best tag sequence for a sentence is sometimes called

decoding because, like spell correction etc, HMM can also be viewed as a noisy channel model.

Named entity recognition: label words as belonging to persons, organizations, locations, or none of the above.

Information field segmentation: Given specific type of text (classified advert, bibiography entry),identify which words belong to which “fields” (price/location,author/title/year)

Sequence labelling: In all of the above POS tasks, deciding the correct label depends on the word to be labeled and the labels of surrounding words. HMM combines these sources of information probabilistically.

Open class words (content words): nouns, verbs, adverbs, adjectives. Mostly content-bearing: they refer to objects, actions, and features in the world. Open class: new ones are added all the time. Closed class words (function words):pronouns, determiners, prepositions, connectives.

Limited num of these. Mostly functional: to tie the concepts of a sentence together.

Probabilistic model for tagging: The model assumes that

each tag depends only on previous tag: a bigram tag model. Words are independent given tags. To generate sentence of length n:

Let t0 =<s>

For i = 1 to n : Choose tag conditioned on prev tag: P(ti|ti-1), Choose word conditioned on its tag: P(wi|ti)

A picture containing sketch, drawing, circle, line art

Description automatically generatedProbabilistic finite-state machine: sentences are generated by walking through states in a graph. Each state represents a tag. Prob of moving from state s to s’ (transition prob): P(ti = s’|ti-1 = s). Example: Proper nouns (NNP) often begin sentences: P(NNP|<s>) ≈ 0.28 – Modal verbs (MD) nearly always followed by bare verbs (VB).–Adj (JJ) are often followed by nouns (NN). When passing through each state, emit a word. Prob of emitting w from state s (emission or output prob):

P(wi = w|ti = s)

Joint prob (In this context): Suppose we have sentence S = w1...wn and its tags T = t1...tn, what is the prob that our probabilistic FSM would generate exactly that sequence of words and tags, if we stepped through at random? First, add <s> and </s> then compute with probs from corpus

Hidden Markov Model (HMM): find the best tag sequence for an untagged sentence. Markov: because of Markov independence assumption (each tag/state only depends on fixed number of previous tags/states—here, just one).

Hidden: because at test time we only see the words/emissions; the tags/states are hidden (or latent) variables. Elements of an HMM: a set of states (here: the tags), a set of output symbols (here: words), intitial state (here: beginning of sentence), state transition probs (here: p(ti|ti-1)), symbol emission probs (here: p(wi|ti)). We usually assume hidden variables are observed during training-- annotated data.

Formalizing the tagging problem: Find the best tag sequence T for an untagged sentence S:

Now we need to compute P(S|T) and P(T) (actually, their product P(S|T)P(T) = P(S, T)).

P(T) is the state transition sequence:

P(S|T) are the emission probs:The Viterbi algorithm: Enumeration to find best tag sequence won’t work as for c tags & n words, there are c

n possible tag sequences. Thus use this algo as it is more efficent. Dynamic programming.

Decoding: general term in NLP for inferring the hidden variables in a test instance.

Named Entity Recognition: [PER Bill Clinton] embarrassed [PER Chelsea] at her wedding at [LOC Astor Courts] Gesture Recognition: Given a sequence of frames in a video annotate each frame with a gesture type. It is hard to predict gestures for each frame in isolation, again need to exploit interaction between gestures in different frames

Generative Hidden Markov Models: Model: Introduce a parameterized model of how both words and tags are

A picture containing text, circle, diagram, font

Description automatically generatedgenerated P(x, y|θ) **(We do not want to specify the transition system (associated probs) but learn it**

**from the data)** Learning: use a labeled training set to estimate the most likely parameters of the model Decoding: where y are tags and x are words. Below: A simplistic state diagram for noun phrases: N = tags, M= vocabulary size. States correspond to POS tags, words are emitted independently from each POS tag

Parameters (to be estimated from the training set): Transition probs: : [N X N] Matrix - Stationarity assumption: this prob^ doesn’t depend on the position in the sequence t

Emission probs: : [N X M] Matrix

A picture containing text, font, line, white

Description automatically generatedA picture containing text, circle, diagram, font

Description automatically generated<- 1st order Hidden Markov Model using Transition and Emission probs specified above.

HMM Estimation: N = the number tags, M = vocab size. It is convenient to think that the output sequences are padded with START and STOP. Therefore y0=START and y|x|+1=STOP.

Params(to be estimated from the training set):

Transition probs: aij = , A - [NxN] matrix

Emission probs: bik = , B - [NxM] matrix

Training corpus:x(1) = (In, an, Oct., 19, review, of, …), y(1)=(IN,DT,NNP, CD,NN, IN,…) where  
 is #times word k is emitted by tag

is #times tag i is followed by tag j

Higher order HMMS: The higher the order, the more

zeros, the more important smoothing of the corresponding

distribution is becoming. Emission probs do not change. Transition prob matrix becomes size of N x num of Order for example, 2nd order HMM: : [NxNxN]

HMMs Decoding: Predict PoS-tags for a sentence. Corresponds to maximization:

Brute forcing takes O(N|x|) time. Viterbi takes O(|x|N2).

HMM VITERBI:

Initialisation:

Recomputation:

Final:

In log-space form (in a more generalized form):

A picture containing circle

Description automatically generatedThe score is associated

w/ a fragment responsible for one transition and one word

generation

🡨

Initialization:

Recomputation:

Final:

Computing the likelihood POS tagging FORWARD ALGORITHM: But there are an exponential number of sequences y so use the forward algorithm, an algorithm similar to Viterbi:

A picture containing text, diagram, line, triangle

Description automatically generatedInitialisation:

Recomputation:

Final:

Expectation-maximization (EM): As in spelling correction, we can use EM to bootstrap, iteratively updating the params and hidden variables.

A picture containing diagram, line, origami

Description automatically generatedInitialise parameters, A(0) and B(0)

At each iteration k:

E-step: Compute expected counts using A(k-1) and B(k-1)

M-step: set A(k-1) and B(k-1) using MLE on the expected

counts

Repeat until doesn’t converge (a stopping criteria)

Counting transitions from (yt-1= i, yt=j ):

Real Counts: count 1 each time we see (yt-1= i, yt =j ) in true tag sequence.

Expected counts: With current A and B, compute probs of all possible tag sequences. If sequence y has prob p, count p for each (yt-1= i, yt=j) in y. Add up these fractional counts across all possible sequences.

EM is guaranteed to find a local maximum of the likelihood.

Not guaranteed to find global maximum. Practical issues: initialization, random restarts, early stopping. Fact is, it doesn’t work well for learning POS taggers!

Forward-Backward algorithm: As usual, avoid enumerating all possible sequences.Forward-Backward (Baum-Welch) algorithm computes expected counts (using forward probs and backward probs) P(yt-1= i, yt=j|x). EM idea is much more general: can use for many latent variable models.

Long-range dependencies: The form of one word often depends on (agrees with) another, even when arbitrarily long material intervenes. We want models that can capture these dependencies.A picture containing text, font, screenshot, line

Description automatically generated

A picture containing font, text, white

Description automatically generatedPhrasal categories: We may want to capture substitutability at the phrasal level. POS categories indicate which words are substitutable. For example, substituting adjectives. Phrasal categories indicate which phrases are substitutable. For <example, substituting noun phrase

Constituency tests:

Coordination- Only constituents (of the same type) can be coordinatedusing conjunction words like and, or, and but

Pass: Her friends from Peru went to the show.

Mary *and* her friends from Peru went to the show.

Fail: We peeled the potatoes.

\*We peeled the *and* washed the potatoes

Clefting- Only a constituent can appear in the frame “ \_\_\_\_ is/are who/what/where/when/why/how …”

Pass: They put the boxes in the basement.

In the basement *is where* they put the boxes.

Fail: They put the boxes in the basement.

\*Put the boxes *is what they did* in the basement.

Theories of syntax: A theory of syntax should explain which sentences are well-formed (grammatical) and which are not. Well-formed is distinct from meaningful. We care mainly for interpreting meaning. Constituency (aka phrase) structures and Dependency structures are two such theories. These can be viewed as different models of language behaviour.

A picture containing diagram, line

Description automatically generatedConstituent trees: Internal nodes correspond to phrases, Nodes immediately above words are PoS tags.

A picture containing font, typography, text, handwriting

Description automatically generatedA picture containing font, typography, text, graphics

Description automatically generatedContext-Free Grammar(CFG): What can be a sub-tree is only affected by what the phrase type is (VP) but not the context. Context-free grammar is a tuple of 4 elements: G = (V, Σ, R, S) where V=the set of non-terminals, Σ= the set of terminals, R=the set of rules of the form X 🡪 Y1, Y2,...,Yn, where n ≥ 0,, S= the dedicated start symbol.

Structural ambiguity: Some sentences have more than one parse: structural

A picture containing diagram, line, design

Description automatically generatedambiguity. Could be caused by POS ambiguity. Attachment ambiguity (ambiguity w/o POS ambiguity)->Depends on where different phrases attach in the tree.Different attachments have different meanings:“I saw a girl with a telescope”. Prepositional Phrase (PP-) Attachment Ambiguity is common-> Did I see the girl w/ my telescope or did I see a girl that had a telescope?

Parsing algorithms: compute the structure(s) for an input string given a grammar (we may want to use the structure to interpret meaning). Ambiguity is a problem. For correctness: need to find the right structure to get the

right meaning.For efficiency: searching all possible structures can be very slow

Parser properties: All parsers have two fundamental properties: Directionality: the sequence in which the structures are constructed.

Top-down: start with root category (S), choose expansions, build down to words.

Bottom-up: build subtrees over words, build up to S.

Mixed strategies also possible (e.g., left corner parsers)

Search strategy: the order in which the search space of possible analyses is explored.

Top-down parser: Start with S node.Choose one of many possible expansions.Each of which has children with many possible expansions. Example picture to the left. Bottom-up parser: works in the opposite way – starts from input sentence and works its way up the tags.

Search strategies: Depth-first search: explore one branch of the search space at a time, as far as possible. If this branch is a dead-end, parser needs to backtrack. Breadth-first search: expand all possible branches in parallel (or simulated parallel). Requires storing many incomplete parses in memory at once.Best-first search:score each partial parse and pursue the highest-scoring options first

Chomsky Normal Form (CNF): Any CFG can be converted to an equivalent CNF. Unary preterminal rules, generation of words given PoS tags. Binary inner rules. Thus in CNF, a non-terminal rule either turns into a singular terminal word or into two other non-terminal tags (no more, no less!)

A picture containing text, font, diagram, line

Description automatically generatedTransformation from CFG to CNF form: 1)Get rid of empty (aka epsilon) productions:, 2) Get rid of unary rules:, 3) Break up rules w/ n>2 tags to expand into. (Binarisation)For example, expanded into the two rules and . The example picture shows step 3 in action and lossless Markovization in the context of PCFGs. Easily reversable postprocessing.

A black and white drawing of a fence

Description automatically generated with low confidenceCocke-Kasami-Younger (CKY): An efficient bottom-up parsing algorithm for CFGs. Can be used both for the recognition and parsing problems. Very important in NLP (and beyond). It is generalizable to probabilistic modeling / PCFGs. We are given a grammar G= (V, Σ, R, S), a sequence of words w = (w1, w2,...,wn), and our goal is to produce a parse tree for w. span(i,j) refers to words between fence posts i and j. Parsing one word: covers all words between i-l and i.

Parsing longer spans: Example to left where . Covers all words between min and max.

Signatures: Applications of rules is independent of inner structure of a parse tree We only need to know the corresponding span and the root label of the tree. Its signature = [min, max, C] where the vars can also be known as edges.

CKY Algorithm: Compute for every span a set of admissible labels (may be empty for some spans)- start from small trees (single words) and proceed to larger ones. When done, check if S is among admissible labels for the whole

sentence, if yes – the sentence belong to the language -that is if a tree with signature [0, n, S] exists. Furthermore,

If a rule can be derived from a chain of rules, do Unary Closure/transitive closure by adding another rule going from A -> C For the sentence seen in the triangle of boxes next to the formal CKY algorithm: where min = 0 and max = 1 is the word: “lead”. Where min = 1 and max = 3 is the substring: “can poison” and its rules come from boxes 2 and 3. This sentence is ambiguous as there is more than 1 route to have an S in box 6.

A picture containing text, screenshot, font, diagram

Description automatically generatedCKY Algorithm more formally: Chart can be represented by a Boolean array chart[min][max][C] (Relevant entries have 0 ≤ min < max ≤ n). chart[min][max][C] = true if the signature (min, max, C) is already added to the chart; false otherwise. Time complexity is θ(n3|R|) where |R| is the num of rules of the grammar. There exist algorithms with better asymptotical time complexity but the `constant' makes them slower in practice (in general) Implementation Preterminal rules:

For each wi from left to right:

For each preterminal rule C -> wi:

chart[i-1][i][C] = true

Implementation unary and binary rules:

For each max from 1 to n: (use max from 2 to n w/o unary)

For each min from max -1 down to 0:(use max-2 w/o unary)

For each syntactic category C: --------Binary Rules

For each binary rule C->C1 C2:

For each mid from min+1 to max -1:

If chart[min][max][C1] and chart[min][max][C2]

then chart[min][max][C] = true

For each syntactic category C:--------Unary Rules

For each unary rule C->C1:

If chart[min][max][C1]

then chart[min][max][C] = true

Probabalistic CFGs (PCFGs): Tries to reduce ambiguity. CFGs where each rule has a prob of how likely it is to be used given a choice of different expansions from the same tag. Then the prob P(T) of the sentence is all of the probs of the rules used multiplied together.

Treebank: a collection of sentences annotated with constituent trees

Distribution over trees: not all PCFGs give rise to a proper dist over trees - the sum over probs of all trees the grammar can generate may be less than 1: . However, any PCFG estimated with the maximum likelihood procedure are always proper. Let us denote by G(x) the set of derivations for the sentence x. The prob distribution defines the scoring P(T) over the trees T G(x) Then create rule probs using maximum likelihood procedure i.e where is the number of times the rule was used in the corpus and C(X) is

the number of times the nonterminal X appears in the treebank. Remember to smooth! Especially important for preterminal rules/generation of words.

CKY with PCFGs: Chart is represented by a double array chart[min][max][C] -stores probs for the most probable subtree with a given signature chart[0][n][S] will store the prob of the most probable full parse tree. For every C choose C1,C2 and mid such that P(T1) ⇥ P(T2) ⇥ P(C -> C1C2) is maximal, where T1 and T2 are left and right subtrees. Using CKY algo for CFGs, add initialisation of “double best” under the “ ----Binary rules” line and change part under “For each binary rule” line to:

for each mid from min +1 to max -1:

double t1­ = chart[min][mid[C1]

double t2­ = chart[min][mid[C2]

double candidate = t1 x t2 x P(C -> C1C2)

if candidate > best then best = candidate

chart[min][max][C] = best

A picture containing text, font, diagram, white

Description automatically generated Transitive rules mess up PCFGs.

Recovery of the tree: For each signature we store backpointers to the elements from which it was

Built. Start at [0,n,S]. Be careful with unary rules.

A picture containing diagram, line, sketch, design

Description automatically generatedSpeeding up PCFG CKY: Basic pruning: For every span (i,j) store only labels which have the prob at most N times smaller than the prob of the most probable label for this span. Check not all rules but only rules yielding subtree labels having non-zero probs. Coarse-to-fine pruning: Parse with a smaller (simpler) grammar, and precompute (posterior) probs for each spans, and use only the ones with non-negligible prob from the previous grammar

Weaknesses of PCFGs: They do not encode lexical preferences. They do not encode structural properties (beyond single rules). Close attachment is a-priori more likely irl for at least some example sets (a PP has attached to the closest possible preceding NP) but PCFGs do not have a preference for structure and what something attaches to apart from rules expanded.

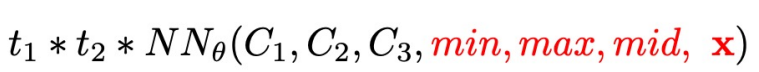
A picture containing diagram, line

Description automatically generatedVertical Markovization: An approach to enrich grammar. Helps PCFGs prioritise different structures such as close attachment over other attachments by giving the extra rules different probs depending on ancestors. Rule applications depend on past ancestors in the tree (not only parents). Example is of vertical order 2. Improves accuracy.

A picture containing text, font, diagram, screenshot

Description automatically generatedHorizontal Markovization: Binarisation. Example is of h=∞.Improves accuracy.

Splitting: Pos Tags: Add more Pos tags if there are instances of a tag being used in too many situations. Boosts performance by 2%. Other symbols:Split determiners: on demonstrative ("those") and others (e.g.,"the", "a"). Split adverbials: on phrasal and not ("quickly" vs. "very"). Boosts F1. Inducing splits (through EM): Learning types of nonterminals from data, i.e. automatically enriching the grammar (Latent-annotated PCFGs, LA-PCFG).One can think of this as a type of clustering of tree contexts of non-terminal symbols. Example:PN-1 =”My”,N-652”=”Dog”,NP-156=”My Dog”. ~90F1

Anchored rules: A rule prob is not constant but predicting for a given span in the chart E.g., a neural network predicts the prob of a rule for a specific operation of the chart.~96F1 

Parser Evaluation: Intrinsic evaluation: Automatic: evaluate against annotation provided by human experts (gold standard) according to some predefined measure. Manual:according to human judgment. Extrinsic evaluation: score syntactic repre by comparing how well a system using this repre performs on some task E.g., use syntactic repre as input for a semantic analyzer and compare results of the analyzer using syntax predicted by different parsers

Standard settings: Automatic intrinsic evaluation- gold standard made by linguists. Split into training, dev, and test sets.

A picture containing text, font, screenshot

Description automatically generatedA picture containing diagram, white, sketch, drawing

Description automatically generatedBracket score: most standard measure for automatic evaluation of constituent parsers. Scores how well individual phrases (and their boundaries) are identified. It regards a tree as a collection of brackets: min, max, C] (subtrees of CKY). The set of brackets predicted by a parser is compared against the set of brackets in the tree annotated by a linguist. Precision, recall and F1 are used as scores. Example of a tree and its corresponding bracketing notation shown.

A picture containing text, font, screenshot, line

Description automatically generated

Exact match: for automatic evaluation of constituent parsers. Percentage of trees predicted correctly

Crossing brackets: for automatic evaluation of constituent parsers. Percentage of phrases boundaries crossing

Dependency metrics: for automatic evaluation of constituent parsers. Scores dependency structure corresponding to the constituent tree (percentage of correctly identified heads)

Dependency Parsing: Replacing one word with another with the same POS will never result in a different

parsing decision, even though it should! Example: Kids saw birds with fish vs.kids saw birds with binoculars Some of the algorithms for PCFGs can be adapted to dependency parsing. CKY can be adapted but is of O(Gn5). Eisner algo is of O(Gn3). Use Shift-reduce.

A way to fix PCFGs, LEXICALISATION: Create new categories, this time by adding the lexical head of the phrase. Example: kids saw birds with binoculars has S-saw as saw is the lexical head. Makes grammar more specific but grammar blows up + sparse data. Need to do complex smoothing or increasing emphasis on automatically learned subcategories. Need a better way.

Lexicalized Constituency Parse: take lexicalised tree and remove phrasal categories i.e S-saw -> saw. Remove duplicated terminals. Continue to collapse chains of duplicates.Dependency parse: top pic is example of dependency parse using lexicalized constituency parsing. Linguists have long observed that the meanings of words within a sentence depend on one another, mostly in asymmetric, binary relations. Though some constructions don’t cleanly fit this pattern: e.g., coordination,relative clauses. Bottom pic example equivalently, but showing word order (head → modifier) Because it is a tree, every word has exactly one parent. Content vs. Functional Heads: Some treebanks prefer content heads, others prefer functional heads. Content heads use the noun of a PP as the head. A picture containing line, diagram, screenshot, font

Description automatically generated

Functional heads use the function word as the head. A picture containing text, screenshot, line, diagram

Description automatically generated

A picture containing text, font, line, screenshot

Description automatically generatedEdge Labels: It is often useful to distinguish different kinds of head → modifier relations, by labeling edges. Example above. Important relations for English include subject, direct object, determiner, adjective modifier, adverbial modifier, etc. (Diff treebanks use slightly diff label sets.)

Dependency Paths: For information extraction tasks involving real-world relationships between entities, chains of dependencies can provide good features A picture containing text, diagram, font, line

Description automatically generated

Projectivity: A sentence’s dependency parse is said to be projective if every subtree (node + all its descendants) occupies a contiguous span of the sentence = The dependency parse can be drawn on top of the sentence without any crossing edges. Other sentences are nonprojective = dependency parsing lines cross each other. Rare in english, but common in other languages.

A picture containing text, line, diagram, plot

Description automatically generatedConstituency Tree → Dependency Tree: Head rules: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. Example: S → NP VP thus whatever is the VP in S is the head. Heuristics to scale this to large grammars: e.g., within an NP, last immediate N child is the head.Then, propagate heads up the tree using each head rule!

A picture containing text, font, line, symbol

Description automatically generatedShift Reduce Parser: Transition based parsing. 3 possible actions: LeftArc: Assign head-dependent relation between s1 and s2; pop s2 RightArc: Assign head-dependent relation between s2 and s1; pop s1

Shift: Put w1 on top of the stack. A picture containing text, screenshot, font, number

Description automatically generated

Transition-based Parsing: Using shift-reduce. Latent structure is just edges between words. Train a classifier to predict next action (Shift, LeftArc, or RightArc), and proceed left-to-right through the sentence. O(n) time complexity! Only finds projective trees (without special extensions) Nivre’s MaltParser.

Graph-based Parsing: From the fully connected directed graph of all possible edges, choose the best ones that form a tree. Edge-factored models: Classifier assigns a nonnegative score to each possible edge; maximum spanning tree algorithm finds the spanning tree with highest total score in O(n2) time. Edge-factored assumption can be relaxed (higher-order models score larger units; more expensive). Unlabeled parse → edge-labeling classifier (pipeline). McDonald’s MSTParser. Can be formulated as constraint-satisfaction with integer linear programming(Martins’s TurboParser)

Graph-based vs. Transition-based vs. Conversion-based: TB: Features in scoring function can look at any part of the stack; no optimality guarantees for search; linear-time; (classically) projective only

GB: Features in scoring function limited by factorization; optimal search within that model; quadratic-time; no projectivity constraint

CB: In terms of accuracy, sometimes best to first constituency-parse, then convert to dependencies (e.g., Stanford Parser). Slower than direct methods, some treebanks are available solely in dependency form.

Compositional Semantics: Semantics is concerned with how expressions relate to ‘the world’. This includes

A picture containing font, text, handwriting, typography

Description automatically generatedboth their truth conditions and denotation (literal meaning) and their connotation (other associations)

For now, we’re interested only in literal meaning. What is meaning? NLP usually takes a more pragmatic view: can the computer behave as though it understands (in order to do what we want)?

Eliza: 1969 computer program made by Joseph Weizenbaum. Simulated a psychotherapist of the Rogerian school (in which the therapist often reflects back the patient's words to the patient).

Question Answering: Like Alexa. A machine that answers questions in natural language,may have access to knowledge bases,may have access to vast quantities of English text.

Semantics: Sentential semantics: how word meanings combine. Who did what to whom; when, how, why John loves Mary ⇒ Someone loves Mary. NOT that Mary loves John. Lexical semantics: the meanings of individual words E.g., John is male, Mary is female

Sentential syntax: Sentential syntax reveals information about sentence meaning. John loves Mary → love(j,m) Captures linguistic generalisations about grammaticality (substitutability), generates an unbounded set of grammatical sentences via a finite lexicon and finite rules (recursion), we can induce probabilistic grammars from a treebank, and so tackle (pervasive) syntactic ambiguity. Syntactic ambiguity (almost) always yields a semantic ambiguity. Decisions about how to resolve syntactic ambiguity are tied up with decisions about (intended) meaning. Resolving syntactic ambiguities does not, however, resolve all semantic ambiguities– word sense, semantic scope, anaphoric expressions so reasoning about context is also very important

Semantic Representation: The semantic representation should: be unambiguous (> 1 semantic representation for I made her duck etc), support automated inference be verifiable: determine if the sentence is true with respect to a model of the world. Use First Order Logic. Don’t use propositional logic as it wont capture the internal structure of a proposition (meaning) We’re unable to express important relationships between, e.g. Everyone eats rice Someone eats rice, Everyone eats something.

Logical vs. Commonsense inference: John buttered toast at midnight on the lawn ⇒ Someone buttered toast, Someone did something at midnight

Davidsonian Semantics: Introducing an event argument e to ‘action’ predicates is very useful:Tense: Fred ate rice: ∃e(eat(e, fred, rice) ∧ e ≺ n)Modifiers: Fred ate rice with a fork at midnight:

∃e(eat(e, fred, rice) ∧ e ≺ n∧ ∃x(with(e, x) ∧ fork(x))∧

at(e, midnight)

Note how the second sentence entails the first via ∧-elimination! Compositionality: The meaning of a complex expression is a function of the meaning of its parts and of the rules by which they are combined. So you can build a logical form of a sentence by specifying: Lexical meanings: Associate each word in the lexicon with a FoL expression. Composition rules: Augmenting each syntax rule in a CFG with instructions for composing the FoL expressions on the RHS into a FoL expression for the LHS.

Lambda Calculus and Beta Reduction: If ϕ is a well-formed FoL expression and x is a variable, then λxϕ is a well-formed FoL expression. It’s a function, known as a λ-term. λ-terms have interesting semantics, but they also offer a way of substituting (free) variables in an FoL expression with values.

Creating a function λxϕ from an expression ϕ is called Lambda (λ) abstraction. Function application is called Beta (β) reduction. Example:

λyλx(∃e(eat(e, x, y) ∧ e ≺ n))(rice) becomes

λx(∃e(eat(e, x, rice) ∧ e ≺ n))

If we introduces variables of ‘higher type’ then we can substitute variables corresponding to properties, relations etc with values that can be λ-terms. Example:

λP.P(fred):the properties of Fred (man, tall,. . . )

λP.P(fred)(man) becomes man(fred)

Rules with two daughters specify in semantics which daughter is the functor and which the argument

Semantic Ambiguity: some FOL are syntactically ambiguious whilst their English counterparts arent. The ambiguity arises because every and a each has its own scope: Interpretation 1: every has scope over a Interpretation 2: a has scope over every. Scope is not uniquely determined either by left-to-right order, or by position in the parse tree. To reduce problem with scope, Semantic Underspecification: Build LFs via syntax that underspecify the relative semantic scopes of the quantifiers- Partial description of a FoL formula so Syntax-Tree:LF is 1:1, but the LF describes several FoL formulae

and hence several interpretations.Sometimes the surrounding context will help us choose between interpretations: Every student has access to a computer. It can be borrowed from the ITO.(⇒ a outscopes every)

The LF constructed in the grammar features: 1. FoL bits, 2. constraints on how they can combine into an FoL formula. The constraints are satisfied by more than one FoL formula. Label nodes of the tree: l1, l2 . . . Supply constraints on what FoL expressions appear at those labels. All hs must equal a (unique) l; no free variables So there are two solutions:

∃ outscopes ∀: h2 = l2, h4 = l5, h3 = l3h5 = l1

∀ outscopes ∃: h2 = l2, h4 = l5, h3 = l1, h5 = l3

LF construction via the grammar must now λ-abstract labels, as well as predicates, arguments to predicates etc.

Distributional hypothesis: similar contexts imply similar meanings. Linguistic distribution: the set of contexts that a particular item (here, word) occurs in

Distributional semantics: Represent each word wi as a vector of its contexts. Distributional semantic models also called vector-space models. In an example, each dimension is a context word; = 1 if it co-occurs with wi, otherwise 0.

First-order co-occurrence (syntagmatic association):

Typically nearby each other. Wrote is a first-order associate of book.

Second-order co-occurrence (paradigmatic association):

Have similar neighbours. Wrote is a second-order associate of said and remarked .Defining Context: Usually ignore stopwords. Large & Small contexts possible. can consider relations other than cooccurrence in a sequence: e.g., dependency relation from parser.All of these for semantic similarity; for syntactic similarity, use a small window (1-3 words) and track only frequent words. Collocations: which words occur unusually often in the context of w: more than we’d expect by chance?

Pointwise Mutual information: what collocations include w? PMI tells us how much more/less likely the cooccurrence is than if the words were independent where P(x,y) = Actual prob of seeing words x and y together and P(x)P(y) = Predicted prob of same, if x and y are indep. It is over-sensitive to the chance co-occurrence of infrequent words

A black text on a white background

Description automatically generated with low confidenceAlternatives to PMI, How to measure similarity:assume we have context vectors for two words and . Each contains PMI (or PPMI) values for all context word. Like points in high -dimensional space.Vector space representation:

cat=(v1,v2), Euclidean distance: doesn’t work well if even one dimension has an extreme value

Dot product:Vectors are longer if they have higher values in each dimension.So more frequent words have higher dot products.But we don’t want a similarity metric that’s sensitive to word frequency. Thus, normalise for longer vectors.

Normalized dot product:

= cosine of the angle between vectors. Ranges from -1 (vectors pointing opposite directions) to 1 (same direction) Other similarity measures: Jaccard measure,Dice measure,Jenson-Shannon divergence. Evaluation: Extrinsic may involve IR, QA, automatic essay marking. Intrinsic is often a comparison to psycholinguistic data-Relatedness judgments, Word association Comparing to human data: Human judgments provide a ranked list of related words/associations for each word w. Computer system provides a ranked list of most similar words to w. Compute the Spearman rank correlation between the lists (how well do the rankings match?) Often report on several data sets, as their details differ

Compositionality in a vector space: implies some operator ⊕ such that meaning(w1w2) = meaning(w1) ⊕ meaning(w2). Possible operators= vector addition (doesn’t work very well), tensor product, nonlinear operations learned by neural networks

Problems for Question Answering & lexical semantics: words may have different meanings (senses), words may have the same meaning (synonyms), words can refer to a subset (hyponym) or superset (hypernym) of the concept referred to by another word – we need to have database of such A is-a B relationships, called an ontology, words may be related in other ways, including similarity and gradation. Sometimes we need to do inference – a problem for sentential, as well as lexical, semantics. Some of these problems can be solved with a good ontology, e.g., WordNet(English) is a hand-built resource containing 117,000 synsets: sets of synonymous words

Synsets are connected by relations such as – hyponym/hypernym (IS-A: chair-furniture) – meronym (PART-WHOLE: leg-chair) – antonym (OPPOSITES: good-bad). Word Sense Ambiguity: One word form, same category, but more than one sense (homonyms): I put my money in the bank. vs. He rested at the bank of the river. Words can have multiple (related or unrelated) sensesWords often exhibit sense ambiguities that fall into (semi)-predictable patterns(regular polysemy). Words are typically semantically ambiguous, there’s a lot of regularity (and hence predictability) in the range of senses a word can take, those senses also influence the word’s syntactic behaviour but all regularities admit (arbitrary) exceptions. Word senses can be productive, making a dictionary model (like WordNet)inadequate

But it’s a dominant model in CL these days, and works quite well in lots of cases.

Lumping vs. Splitting: For any given word, lexicographer faces the choice: Lump usages into a small number of senses? Or Split senses to reflect fine-grained distinctions?

Synsets and Relations in WordNet: Synsets (“synonym sets”, effectively senses) are the basic unit of organization

in WordNet.

– Each synset is specific to Ns,Vs,ADJs, or ADVs. Synonymous words belong to the same synset. Polysemous words belong to multiple synsets numbered roughly in descending order of frequency. Synsets are organized into a network by several kinds of relations, including: Hypernymy (Is-A) and Meronymy (Part-Whole)

WordNet is not complete. Neologisms missing etc.

Word sense disambiguation (WSD): Given a sense ambiguous word, find the sense in a given context

WSD as classification: Given a word token in context, which sense (class) does it belong to? We can train a supervised classifier, assuming sense-labeled training data. SensEval and SemEval provide data.

Naive Bayes for WSD: • Naive Bayes requires estimates of:– The prior probability of each class (sense) – The probability of each feature given each class. These can be estimated from the training data. What features?

Simple features: Directly neighboring words (and/or their lemmas), Any content words in a 50 word window, Syntactically related words, Syntactic role in sense, Topic of the text,Part-of-speech tag, surrounding part-of-speech tags. Problem if features are correlated->MaxEnt

Evaluation: Extrinsic: test as part of IR, QA, or MT system

Intrinsic: evaluate classification accuracy or precision/recall against gold-standard senses Baseline: choose the most frequent sense (sometimes hard to beat)

Issues with WSD: Not always clear how fine-grained the gold-standard should be- difficult/expensive to annotate corpora with fine-grained senses

• Classifiers must be trained separately for each word

– Hard to learn anything for infrequent or unseen words

– Requires new annotations for each new word

– Motivates unsupervised and semi-supervised methods

Semantic Classes: Other approaches, such as named entity recognition and supersense tagging, define coarse-grained semantic categories like person, location,artifact.

Like senses, can disambiguate: APPLE as organization vs. food.

Unlike senses, which are refinements of particular words, classes are typically larger groupings. Unlike senses, classes can be applied to words/names not listed in a lexicon.

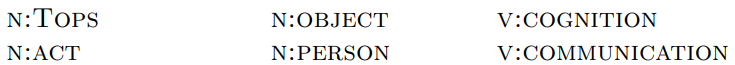
Named Entity Recognition: Recognizing and classifying proper names in text is important for many applications. A kind of information extraction. Different datasets/named entity recognizers use different inventories of classes.

– Smaller: person, organization, location, miscellaneous

– Larger: sometimes also product, work of art, historical event, etc., as well as numeric value types (time, money, etc.)

• NER systems typically use some form of feature-based sequence tagging, with features like capitalization being important.

• Lists of known names called gazetteers are also important

Supersenses in WordNet: goes beyond NER to cover all nouns and verbs. Example of some supersenses: 

A picture containing text, font, diagram, line

Description automatically generatedA picture containing text, font, white, origami

Description automatically generatedA picture containing text, diagram, screenshot, design

Description automatically generatedNeural models and word embeddings: Green boxes = Word representation – vector (input for model/ algorithm)

One-hot vectors as word representations: In a vocab size of v, words are turned into vectors of size v where one 1 represents the word and the rest of the vector is made of 0s. Example: 0001000, 01000000… Issues: - very long, do not capture semantic similarity between words

Latent Semantic Analysis (LSA): Learning a more compact space.

A picture containing text, font, screenshot, line

Description automatically generatedContext: surrounding words in L-sized window

Matrix element: N(w,c) = num of times word w appears in context c

To make it work reasonably well, you need something more sophisticated (e.g., PMI) Box to the left below: This is either the ‘raw’ cooccurrence matrix N, or its transformations (e.g., PMI) A picture containing text, diagram, screenshot, line

Description automatically generated

Prediction-based (aka neural) methods: Mikolov’s Skipgram

A picture containing text, screenshot, font, diagram

Description automatically generated

A picture containing text, screenshot, font, line

Description automatically generated

A picture containing text, screenshot, font, diagram

Description automatically generatedHow do we calculate the probabilities ? For each word w we have two vectors:- when it is a central word - when it is a context word

The probability of the context word o given the central word c is:

Where = dot product: measures similarity of o & c. Larger dot product = larger prob. Where =Normalise over entire vocab to get prob dist.

What do we optimize?

A picture containing text, font, line, white

Description automatically generated

Where left-hand summation agrees w/ the plan to go over all text, the second summation is the sliding window, and the log prob is the prob of the context word given the central. To optimise loss, we rely on gradient descent. In practice, we optimize one word at a time.

A picture containing text, font, white, line

Description automatically generated

A picture containing text, font, white, line

Description automatically generated

A picture containing text, diagram, line, screenshot

Description automatically generated

Relation to LSA: It is possible to show that optimizing the skipgram objective (with the negative sampling modification) corresponds to factorizing PMI matrixA picture containing text, diagram, screenshot, plan

Description automatically generated

A picture containing text, screenshot, font, diagram

Description automatically generatedClassification: P(class = k|…) step is ”get dist over classes”

A picture containing text, screenshot, font, line

Description automatically generatedLogistic Regression uses a similar format where the feature extractor has some manual input features, you get h then feed h into this:

Where you take the dot product of h w/ the feature weights for every class. Do softmax.

A picture containing text, screenshot, diagram, line

Description automatically generatedNN Classifier: Process text using a neural network before putting into the logistic regression model. You feed h into a d-sized vector. What do we optimize? Optimize conditional log-likelihood, as with logistic regression. The NN could be a bag of words model.

A diagram of a cell

Description automatically generated with medium confidenceRecurrent Neural Networks RNN cell: The initial vector h0 is the zero vector. A picture containing text, font, diagram, line

Description automatically generated

Text representation with RNN: The first cell has the initial vector & the first word of the input. In an RNN with one layer, only one output for the entire layer -> the last ht. In a multi-layer RNN, the ht out that is optional in the diagram gets fed into the next layer on the same t with the last layer at the last t outputting the last ht. In a bi-directional RNN, some layers process the input from left to right, the rest process it from right to left. Unlike the multi-layer RNN, the different directions directly have the input text as their input rather than passing ht to the next layer. The forwards and backwards RNN results are then concatenated together.

Language modelling: the language model assigns the probability to a sequence of words relying on the chain rule.

Conditional Language modelling: Where we condition on source x,

Used for encoder-decoder.

Training the language model: Same way as training a classifier.

)

Sequence-to-sequence modelling: Machine translation:

Encoder-Decoder framework: encoder - reads source sequence and produces its representation; • decoder - uses source representation from the encoder to generate the target sequence. Can be used for translating sentences to different languages. Encoder uses the source input whilst the decoder uses previous history. Outputs h that is then transformed from a size d vector to |V| (the vocab size) using a linear layer like the NN classifier. Last encoder states: near-paraphrases seem close in the space.

Training Encoder-Decoder: Loss = same as training the language model but also conditioned on x. A picture containing text, diagram, screenshot, line

Description automatically generated

Inference/Decoding: The simplest idea – at each step, pick the most likely token but note: argmaxy(Product()) is different from Product(argmaxyt()).

Beam search: At each step, you keep top k best approx possibilities. Generate beam\_size most probable tokens, look at probabilities of substrings using tokens, take top k hypothesis. Repeat steps from last word of remaining hypothesis. A picture containing text, font, diagram, screenshot

Description automatically generated

A picture containing text, screenshot, font, number

Description automatically generatedAttention Intuition: At every step, the decoder decide on which input tokens to focus. Calculates what to focus on from output of each step of the encoder and decoder and then passes the result to the decoder. Attention weights:dist over source weights. Attention output: weighted sum of encoder states w/ attention weights

Log Laws: log(M\*N) = log(M) + log(N) , log(M/N) = log(M) – log(N),

log(M^k) = klog(M), log(1) = 0, log\_x(x) = x, log\_x(x^k) = k, x^(log\_x(k)=k Natural Log: ln e^x = x

Exponent Laws: b^-n = 1/(b^n), (b^m)(b^n) = b^(m+n), (b^m)/(b^n)=b^(m-n), (b^m)^n=b^mn, (ab)^m = (a^m)(b^m)

Probability rules: If events are mutually exclusive: P(A or B) = P(A) + P(B), otherwise: P(A or B) = P(A) + P(B) – P(A,B) If A and B are independent: P(A,B) = P(A)\*P(B), otherwise P(A,B)=P(A)⋅P(B∣A). If A&B are independent:P(B|A)=P(B), otherwise P(B∣A) =P(A,B)P(A)

Testing independence:test if P(B|A)=P(B) or P(A,B)=P(A)\*P(B)

Bayes Theorem:

Expected Value: Where P(a) is some pmf or pdf

Standard Deviation:

Normal Distribution:

Syntax of FOL: ∃=existential quantifier, ∀ =universal quantifier

→ = implication=!p or q , ↔ = bi-conditional = (p → q) & (q → p) =

(p & q) or (!p & !q), ≺ = precedes/happened before.

And-Elimination: From “P & Q” we can get P. From “P and Q” we can get Q. From a conjunction, we can derive either conjunct.

Entailment FOL: The formula φ is logically implied by a formula ψ, if φ is true in all the interpretations in which ψ is true. “S entails T if all the possible ways that the world could conceivably be that make S true are also situations where T is true.”

Davidsonian Semantics: e is an event. Can be used to capture past tense. Example: ∃e(eat(e, fred, rice) ∧ e ≺ n) = There exists an eating event e done by fred of rice & e happened before now

Well-formed Formulas (WFF): 1) Any proposition is a WFF = P , Q , S

2) if a is a WFF, a is a WFF. 3)If a is a WFF and b is a WFF, (a & b) is a WFF, (a || b) is a WFF, (a b) is a WFF, and (ab) is a WFF. 4) Nothing else is a WFF Demorgan’s law: ¬(p∨q)⇔(¬p)∧(¬q) , ¬(p∧q)⇔(¬p)∨(¬q)EM Algo: Needs an initialisation of params and arrays. E-step= Use the observed data to estimate vals of the missing data. M-step: update params. Check for convergence & repeat until converged.

Types of proof: Induction: Prove thing is true for n=1, assume it is true for n=k, and then prove it holds for n=k+1. Contradiction: assume ¬P is true and prove ¬P can’t hold. Indirect: Do direct proof on contrapositive (¬q→¬p)