Natural Language Processing: An Introduction

Lecture 1

Applications: Information extraction and sentiment analysis (negative and positive word detection for word reviews), machine translation (different languages), word sense disambiguation (words that have different meanings)

Mostly solved: parsing, spam detection, part of speech tagging, named entity recognition Hard: question answering, paraphrase, summarization, dialog

Ambiguity problems - crash blossom: a headline that has two meanings and the ambiguity causes humorous interpretation

Reasons NLP is hard: non-standard english (twitter feeds), segmentation issues, idioms, neologisms, world knowledge, tricky entity names. Therefore we use probabilistic models built from language data, with rough text features

Two kinds of errors in NLP: false positives (recognizing more than needed) and false negatives. In word tokenization: need to differentiate words (e.g. using uh in a sentence should not count as a word). Lemma: same stem, part of speech, same rough word sense. Wordform: the full inflected surface form.

Church and Gale proved that:

$$V > O(N^{\frac{1}{2}}) \tag{1}$$

where N = number of tokens, V = vocabulary, set of types.

Word segmentation algorithm: max match. Start a pointer at the beginning of the string. Find the longest word in the dictionary that matches the string starting at the pointer. Move the pointer over the word in the string. Repeat step 2 i.e. find the longest word. This works well for Chinese, not English.

Word normalization: Lemmatisation: finding the correct dictionary headword form. Porter's algorithm is used for stemming (number of rules). You can use a decision tree to determine things like ends of sentences and other segmentation features.

Minimum edit distance: the minimum number of operations needed to convert one string to another. (insertion, deletion, substitution). Levenshtein distance costs 2 for substitution. Use dynamic programming: Compute $D(i,j)$ where i is a substring of length i of string X and j for Y. Any non decreasing path from 0 to nm, is a sub alignment. An optimal alignment is composed of optimal sub alignments. Time is $O(nm)$ and backtrack takes $O(n+m)$.

Weighted min edit distance: Add a cost for each operation, therefore there exists a cost table that you can look up for each i and j e.g. instead of $D(i,j)+1$, write $D(i,j)+\delta(i,j)$;

Needleman-Wunsch algorithm is weighted minimum edit distance.

Lecture 4

Probabilistic language models: assign a probability to a sentence. Example for machine translation (p(high winds) > p (big winds)), speech recognition, summarization.

Chain Probability. Markov Assumption: Consider only the previous or maybe last two words, rather than the entire sentence.

$P(w_1w_2..w_n) = \prod_i P(w_i|w_{i-k}\dots w_{i-1})$.

Unigram model: $P(w_1w_2..w_n) = \prod_i P(w_i)$

Bigram model: $P(w_1w_2..w_n) = \prod_i P(w_i|w_{i-1})$

N-gram models (consider last n words) and so on. These are usually inefficient because language has long distance dependencies.

Estimating n-gram probabilities: Maximum likelihood estimate:

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

Evaluation language models: Assigning probabilities to sentences i.e. low P to unreal or grammatically incorrect sentences. Train parameters of model on a training set and then test their performance on a test set i.e. unseen data. Extrinsic (in vivo) evaluation: look at something external to the model and use it to evaluate i.e. look at performance on that external task e.g. using a machine translation or speech recognition system and running evaluations to compare tasks A and B (best language model is one that best predicts an unseen test set i.e. given a word, the best language model should give the best possible word to come after that. it gives the highest $P(sentence)$). Intrinsic evaluation: Perplexity. Perplexity is the inverse probability of the test set normalized by the number of words. Minimising perplexity is equivalent to maximizing probability. Perplexity is related to the average branching factor. Lower perplexity implies better model.

Shannon Visualisation Method: Choose a random bigram according to its probability. Keep going from the last word until you reach the terminating symbol. String them all together.

Generalisation: dealing with zeroes. Use add one estimation or Laplace Smoothing i.e. add one to all the counts.

The max likelihood estimate of some parameter of a model from a training set is one that maximized the likelihood of the training set, given the model. Add one smoothing is a blunt estimation, especially when large number of zeros, it takes off more probability mass from the others to distribute over the zeroes.

Backoff: when you want less context (maybe a trigram is seen only once, therefore use bigram, otherwise unigram)

Interpolation: mix unigram, bigram and trigram with some weighting factor. You can have simple interpolation or have lambdas conditional on context. The lambdas are set using a development set (used to set parameters and check for things). Choose lambda to maximize the likelihood or probability of the dev sat. First fix the ngram probabilities based on training data. Then search for lambdas such that likelihood of dev set is the highest.

How to deal with web scale grams: pruning (only store grams that have very large count). By Zipfs law, there are lots of elements with count 1. Or use entropy based pruning. Store indices instead of strings, huffman coding etc.

Advanced language model: Discriminative model: choose ngram weights such that they improve that particular task; speech recognition as opposed to machine translation

Good Turing smoothing: use the count of things we've seen once and use them to estimate the count of things that have never been seen. Create classes, N_c = count of things seen c times. P(things with zero freq) = $\frac{N_1}{N}$ and $c^* = \frac{c+1(N_{c+1})}{N_c}$

Good Turing Smoothing distributes the probability mass by taking some amount from every variable and giving it to the zero occurrence variable. If you observe the values after discounting, they are all reduced by some constant number and are close to the prior value. Therefore we could just reduce every probability by some small constant, different for each corpus. Absolute discounting interpolation subtracts 0.75 or some constant from every count. There $P(w_i|w_{i-1}) = (\frac{c(w_{i-1},w_i)-d)}{c(w_{i-1}}) + \lambda(w_{i-1})P(w)$ where λ is the interpolation weight and $p(w)$ is from the unigram model.

Kneser-Ney Smoothing: Intuition is that we want better estimates for probabilities of lower order unigrams (not just the highest count, but the most relevant word) This is because unigrams are useful when we haven't seen the bigram. So for each word count the number of bigrams it can possibly create and each bigram type is a novel continuation the first time it was seen. There Pcont(w) = the number of words that can occur before w.

Pcont(w) = Number of times wi-1 has a word before it/Number of bigrams

P(wi—wi-1) = (max(c(wi-1, wi) - d)/c(wi-1) + lambda(wi-1)Pcontinuation(wi)

For lambda, take all the discounts from the interpolation model and combine them; therefore it's similar to an aggregate of the probability mass we've discounted but somewhat normalized

lambda = d/c(wi-1) —w: c(wi-1, w) ¿ 0)— where the —— is the number of word types that can follow wi-1 i.e. the number of word types we discounted.

Recursive KN: Use the normal count for higher orders, when you get to lower orders, recurse until the base case when you use the continuation count.

Lecture 5

Noisy channel: Run each word through noise, generate candidate set. To find the correct word w, we want a word that maximizes the P(w—x) i.e. the P of the word given the misspelling

NCM: maximizing the product of two terms: likelihood and prior i.e. error or channel model and language model. P(x—w)P(w)

P(w) is from the language model; tells us how likely that word is. P(x—w) is from the noisy channel model and tells us how likely we are to misspell that particular word in that way.

P(x—w) : del(wi-1, wi)/count(wi-1wi) or ins(..)/(..) or sub or trans

(RW spell correction uses the same NCM, add the word itself to the candidate set) For real word spelling correction: for each word, generate a candidate set (w1, w2, wi) corresponding to edit distance 1. You can create a graph and find the most probable path.

HCI spell correction determines what to do based on the confidence factor.

State of the art noisy channel: 1. Weight the probability: w = argmax P(x—w)P(w)$^{l}ambda, wherewelearnlambdafromthedevset.Weneverjustmultiplyprioranderrormodel, bu$
$conferthemisspellingtopronunciation, andfindwordswhosepronunciationis1editdistanceawayfromthem$

Factors influencing misspelling: surrounding letters, nearby letters on the keyboard, homology on the keyboard (left hand), pronunciation, morpheme transformations

Allow richer edits (ph to f) Incorporate pronunciation into channel

Lecture 6

Male writers tend to use more determiners, females tend to use pronouns; personal pronouns.

Text classification and Naive Bayes: Input to TC: a document d, a fixed set of classes and training set of m hand labeled documents Output: a predicted class c belonging to C.

(Bag of words representation: isolate the important words, and represent the document by a list of words and counts)

Classification methods: Input: document d, fixed set of class C = c1, .. cnm a training set m of hand labeled documents (d1, c1) .. (dm, cm) Output: a learned classifier y:d -¿ c

Naive Bayes: Want C: argmax P(c—d) = argmax P(d—c)P(c)/P(d) = argmax P(d—c)P(c)

We want to find the class that has the maximum probability given the document. Use Bayes rule, ignore the denominator (P(d) is identical for all documents), and argmax P(d—c)P(c) (likelihood*prior)

P(class) is how often that class occurs, so you can count relative frequencies in a dataset e.g. do you normally have more positive classes or negative classes.

Multinomial Naive Bayes: assume independence of variables and bag of words. First attempt: for MLE, simply use frequencies in the data. Use a mega document: example positive document composed of all positive entries in the data set. Have to use add 1 smoothing, otherwise might be zero. Steps: from training corpus, extract vocal. Calculate P(cj). Calculate P(wk,cj)

In Naive Bayes, each class is a unigram language model. Compute probabilities of each class given the sentence. Whichever has a higher P classifies the sentence.

A 2x2 contingency table is a truth table showing true positive, true negative, false neg and positive, depending on the classes and whether or not it was selected or not.

Accuracy = tp + tn/ tp + tn + fp + fn. Accuracy is not a good factor because we are not interested in true negatives, we want a measure of the true positives or false negatives.

Precision (percentage of selected items that are correct) = tp / tp + fp and recall (percentage of correct items that are selected) = tp / tp + fn

F measure: combined measure that assesses the P/R tradeoff (weighted harmonic mean), therefore F = 1 / A/P + (1-A)1/R or F = (B$^2+1)PR/B^2P+RBalancedFmeasuresetsB=1$.

A high biased algorithm is one that does not overfit too much on a small amount of data

If we have more than once class and want to combine multiple performance measures, we use macro or micro averaging. Macro: compute performance for each class, then average. Micro: collect decisions for all classes, compute contingency table sand evaluate. (add two contingency tables together).

Dev set is to test the algorithm or program while running, maybe to find bugs. You can split the training set in different ways as training + dev set and run the classification each time.

Practical ways of helping: use logs to prevent underflow, tweak performance by collapsing terms (equation, chemical formula), upweight title words etc.

1 Lecture 7

Learning sentiment lexicons: semi-supervised learning: use a small amount of info e.g. a few labeled samples and then bootstrap a lexicon Hatsivassiloglou: Intuition for identifying word polarity: words joined by 'and' have same polarity, 'but' has different polarity. They first labeled a seed set of positive and negative seeds. Classifier finds how similar words are, resulting in a graph. Then cluster the graph.

Turney algorithm: first extract a phrasal lexicon.then learn polarity of each phrase. then rate a review by the average polarity of its phrases.

(PMI intuition: P(x, y)/P(x)p(y) how much more do events x and y co-occurs than if they were independent)

Finding aspect or target of sentiment: first find the most highly frequent phrases across reviews. Then filter by rules like "occurs right after sentiment word".

Sentiment classification for movie reviews: Steps: tokenize words, feature extraction and then classify using one of the classifiers above.

Tokenization: may want to keep capitals for expression, phone numbers or dates and emoticons, keep track of twitter hashtags that refer to something. Might have to deal with negation (didn't love) and superlative adjectives.

Use binarized or boolean multinomial naive bayes: we clip all word counts in document because we don't need term frequency (fantastic 5 times means roughly the same thing). Therefore remove all duplicate words from the corpus.

Not the same as multivariate bernoulli naive bayes. You can similarly use log(freq(w)) also.

Cross validation: break up data into 10 folds and inside each fold you have the same number of different classes i.e. test set has same distribution as the training set. For each fold choose the fold as the temporary test set. Train on the other 9 folds and compute the performance on the test fold. Then report average performance of 10 runs.

Problems: subtlety: it's hard to determine if something is negative even for humans. some negative reviews may have positive words that do have a negation but in a subtle way.

Lecture 8

Generative models are the language models and Naive Bayes models we looked at before. Discriminative models use conditional probabilities; they give a high accuracy performance, they allow automatic building of language independent NLP systems.

Generative models generate the observed data from the hidden data. (HMM, naive byes, ngram models) Discriminative: take the data but put a probability over hidden structure.

Features f are elementary pieces of evidence that link aspects (d) of what we observe with a category (c) that we want to predict.Models assign to each feature a weight: positive indicates that the configuration f(c, d) is correct. Use empirical count or expectation of a feature: summation over all (c, d) f(c,d) Model the expectation of a feature: E(fi) = summation over all (c, d) (P(c,d).fi(c,d)) In NLP usually a feature specifies: 1. an indicator function 2. a particular class. Each feature picks out a data subset and suggests a label for it.

(features can simply be words in an article i.e. presence of words and document class (text categorization), features can be words in context or length of sentences (word sense disambiguation), features can be words, its tag and the previous word (POS tagging)).

Linear regression seems like it wouldn't work for text categorization using the features method because they numbers may go out of bounds and they aren't probabilities.

You can use a maxent classifier when you want to assign data points to one of a number of classes. eg. sentence boundary detection (for example A.C.L. we might want to look at features before the period or after the period if there isn't a capital letter, etc), sentiment analysis, parsing

Feature based Linear Classfiers: Assign each feature fi, a weight lambda$_i . Consdiereachclassforanobserv$
$summationlambda_i . fi(c, d) andchoosetheclasscthatmaximizessummationlambda_i . fi(c, d). Waystochoosew$
$Perceptron(findacurrentlymisclassifiedexampleandnudgeweightsinthedirectionofitscorrectclassificat$

Different types of models: 1. Exponential: we want the summation lambda$_i . f_i toalwaysbepositive, sotake$
$theexponentsincreasetheamountalittle, sothefunctionisdominatedbythelargestvotes, thereforeit'sasoftm$

(Maxent models in NLP are essentially the same as multi class logistic regression models in statistics or ML. The parameterization is slightly different with tons of sparse features. (but might be overparameterized). Secondly, feature functions are the most important here and it shows how you can put text into the model more simply.

Define features or indicator functions, features represent sets of data points that are distinctive enough to deserve model parameters. Encode each feature as a unique string (eg. actual word = running, end = ing, num = decimal) and each feature gets a real number weight.

Building a maxent model: iterative process where features are usually added at each step during the development process. eg. first add a number of features, test on dev set, then refine the model to add more and keep testing on dev set. finally test on test set.

Problem with Naive Bayes and how maxent models can help (naive over counts): Naive Bayes over counts when there are features that tell you the same thing i.e. multiple features that refer to the same instance. Maxent solves this because the features are weighted therefore model expectations of features match the observed (empirical) expectations.

Likelihood value: log likelihood: logP(C—D,lambda) = summation logP(c—d, lambda) = summation log(exp(summation lambda$_i.fi(c,d)/summation(exp(summationlambda_i.fi(c,d)))$

Lecture 9

IE: find and understand laminated relevant parts of text. Goal. help people to organize information for people, or put info in semantically precise forms for computers to work with further.

Use of NER: sentiment can be attributed to NEs, lots of IE relations are associations between named entities, and sometime lots of answers to questions in QA are named entities

Precision and recall are good metrics for IR or text categorization because we're considering the entire document, but here we're considering sets of words or sequences. NER usually makes boundary errors (it could detect one boundary correctly in which case it would be partially right but not what we would want).

(In NER boundary errors decrease both precision and recall because they increase the denominator i.e. fps and fns. Therefore not selecting the word at all would increase the evaluation metric).

Named entity recognition (NER) : find and classify names in text uses: names can be indexed or linked, sentiment can be attached to names, used for question answering, for semantic recognition e.g. in a calendar

ML sequence model approach to NER: 1. collect a set of representative training models (containing entities and context related to them) 2. label each token for its entity class or other. 3. design feature extractors appropriate to text and class. 4. train s sequence classifier to predict the labels from the data.

Testing or classifying: 1. receive testing documents 2. run sequence model inference to label each token. 3. appropriately output recognized entities

Encoding classes: IO encoding (C +1 labels) or IOB encoding (2c + 1 labels)

IO encoding can see the boundary between 2 names if one is PER, other is ORG.

Sequence labelling tasks: POS tagging, NER, word or text segmentation

Features for sequence labeling: current word, previous.next word for context. POS tags. label for context (eg. PER-PER is likely to occur), another feature is word shapes (medicinal names, biological sequences, chemical names etc)

Inference in systems: We have sequence data and we want to classify each character or word to its class at the sequence level. Look at each decision individually: look at all its data, do feature extraction over that data and build a maxent model over that data. At each specific item of data, do data extraction for that item i.e. look at it's features through a local classifier.

A CMM or MEMM is one i which the classifier makes a single decision at a time, conditioned on evidence from observations and previous decisions.

Maximum entropy sequence models: (POS tagging, segmentation: BI rep: begin and inside, NER)

Bean inference: at each position, keep the top k complete sequences, at each stage consider the completed subsequences and extend each sequence in each local way. These created extensions compete for the k slots at the next position. Disadvantage: inexact; the best sequence overall might not be in the beam

Viterbi inference: dynamic programming or memoization. Requires small window of state influence i.e. each label depends only on a few other labels within a small window. (not too bad, because its hard to get long distance interactions anyway). Disadvantage: hard to implement long distance interactions between states

Conditional Random Fields: a whole-sequence conditional model, rather than a chaining of local models. But if fi features ram in local, cone sequence likelihood can be calculate using dynamic programmings.

CKY algorithm: A cubic time algorithm both in size of the length of the sentence and the size of the grammar in terms of number of non terminals. Uses a parse triangle or chart based on the parse tree. Since it's half a square, the number of squares we have is O(n^2)$andwecanfilleachsquareinO(n), therefore the time taken by CKY is O(n^3)$.

Fill up the chart working in layers, grouping words at each layer. e.g. lowest layer corresponds to one word for each cell, then the layer above has two words for each cell and so on, based on an existing grammar rule.

Binarization is vital i.e. each rule has at most two things on the RHS, else the running time won't be cubic and it'll lead to a higher order polynomial algorithm or exponential.

Lecture 10

Relation extraction: which entities in a sequence of text refer to one another.

Used to create new structured bases or augment existing bases.

How to build relation extractors: 1. hand written patterns. 2. supervised machine learning 3. semi unsupervised: bootstrapping, distant supervision, unsupervised learning from the web

Hand written patterns: (if you see "is a", "such as", "is located in", you know the relation between the two. Adv: high precision, dis: low recall, have to think of all possible patterns

Supervised: choose set of relations you want to extract, chose a set of relevant named entities, find and label the data (choose a representative corpus, label NEs in the corpus, hand label relations and break data into training, dev and test)

Evaluation of supervised RE: P = number of correctly extracted relations/total number of extracted relations and R = number of correctly extracted relations/total number of gold standard relations

Unsupervised: gather a set of seed pairs that have relation R: iteratively find sentences with these pairs. look at the context between or around the pair and generalize the context to create patterns. use the patters to grep for more pairs.

Lecture 11

Smoothing Maxent Models: the parameters that we fit might be spiky estimates which we don't want.

If we don't smooth, feature weights might be infinite in an iterative process. Also if we consider features that are complementary, we could've just replaced it with one or take a feature that measures how much the two features differ in weights.

Lecture 12

POS: noun, verb (main, modals), adj, adv, propositions, particles, interjections, numbers, closed class (determiners, conjunctions, pronouns)

Closed classes: determiners, pronouns, prepositions because there aren't likely to be more invented words of these classes. Open classes include nouns, verbs, adjectives etc.

POS tagging problem is to determine the POS tag for a particular instance of a word. Steps: Start with a string of words (ssentence) and see which tags can work for each word in contextt.

Uses: text to speech (read or read), used to write reg expressions, used as input to a parser

POS tagging performance: tag accuracy (baseline is about 90

Main sources of information: knowledge of neighboring words for context of the word, knowledge of word probabilities (some words are never used as adjectives/verbs etc), look at the current word prefix (-un or -in), suffix (-ly), word shape etc.

Improve supervised results: add a feature that looks at the previous word or next word add a feature that lowercases the words and then checks (for instance the start of a sentence; if unknown in lowercase the probably a name, if known in lowercase form then we have its tag)

POS Tagging accuracies: most freq tag: accuracy 90HMM models: 95maxent model: use features 93TnT (HMM++): use feature based ideas for predicting unknown words: 96MEMM tagger: $96 and 87 Bidirectional dependencies : maxent model with bidirectional dependencies : 97$

Summary: change from generative to discriminative foes not improve results by a large amount. One profits from models for specifying dependence on overlapping features of the observation (spelling, suffix).

An MEMM allows integration of rich features of the observations, but can suffer from assuming independence from following observations. Improve this adding dependence on following words. MEMM, CRF improve accuracy. However, higher accuracy comes at the price of much slower training.

Lecture 13

Constituency parsing: Organise words into nested constituents. We can find a constituent by looking at words that usually stay together in different sentences. (if you normally reorganize words without that nested constituent, it usually doesn't make sense). One piece of syntactic evidence is that noun phrases precede verbs.

The phrases are structured using CFGs. X bar theory: In natural language, phrases can contain intermediate constituents that have to be projected from the head. Second, the system of projected constants may be common,

Dependency structure: shows which words depend on or modify or are arguments of which other words.

Classical parsing problems: we can add constraints to limit unlikely parses, but then some texts have no parse whatsoever. But sometimes if it is less constrained; a simple sentence can have multiple parses and we're unaware of which one to choose.

Statistical parsing systems: treebanks. Building a treebank allows reusability of labour, broad coverage, frequencies and distributional information and a way to evaluate systems.

Attachment ambiguities: For various phrases, how do we decide attachment? i.e. deciding what part of the sentence that a particular word or phrase modifies.

Catalan numbers: exponentially growing series in tree like structures.

In top or bottom down parsing we do an exponential amount of work, because sometimes you keep building the same structure over again. Repeated work in parses (top down or bottom up). We want to avoid doing the same work twice.

Words are good predictors of attachment.

Dependency parsing from notes: For a dependency tree create a directed graph but vertices as words in a sentence and directed edges as dependencies. (for a tree: exactly one root with no incoming edges, each vertex has exactly one incoming edge, there exists a unique path from root to every vertex) These constraints ensure that each word has a single head and the dependency structure is connected.

An edge from head to dependent is projective if there is an edge from head to every word between the head and dependent. Entire tree is projective if all edges are projective.

Reranking and self training: (train a parser using a normal generative model. get a ranking of trees for the top k values. train a second model on the more specific/desired features. then re rank the k values). This is done because we cannot add the specific feature vector in the generative model, if it is too specific and has not occurred in the training data, the P will be zero. Statistical parsers have shown improvements largely due to incorporation of a large number of features that they are trained on (treebank data). This has led to concerns because some parses are too finely tuned to the corpus.

Lecture 15

A grammar consists of a set of rules such as S = NP VP, etc. A CFG is a four tuple G = (T, N, S R) i.e. terminals, non terminals, start symbol, rules or productions of the form N -¿ (NUT)* (non terminals are categories of phrases: noun phrases, verb phrases, preposition phrase). Therefore terminals are the lexicon or vocal, non terminals are phrases, rules describe the allowed structure of the constituents.

NLP phrase structure grammar: G = (T, C, N, S, L, R) where C = pre terminal symbols, L is the lexicon (a set of items of the form X -¿ x, where X is in P and x is in T)

PCFGS: G = (T, N, S, R P) where P is a probability function such that for all X belonging to N, summation P(X -¿ x) = 1 A grammar G generates a language model L.

P(t) is the probability of a tree: the product of probabilities of the rules used to generate it. P(s) is the p of a string (sentence): the sum of probabilities of the trees which have that string as their yield (for instance some words can be used as nouns or adjectives or verbs and we get different probabilities from different trees based on how they are used)

Grammar transforms: CNF: X -¿ YZ or X -¿ w (Steps for transforming: remove empties and unaries) This can be thought of as a transformation for efficient parsing. Binarization is crucial for cubic time CFG parsing.

(Penn Treebank has functional tags, so while processing we get rid of functional tags. We keep unaries because it's easier to decode).

CKY: Exact polynomial time parsing. We have a PCFG and a sentence and what we want to find, is a sentence structure licensed by the grammar i.e. the most probable sentence structure if we deal with PCFGs. Uses a data structure called parse triangle or chart. The bottommost row corresponds to words in the sentence. The second last corresponds to two words and the topmost corresponds to the entire sentence. We can fill in each square in O(n) and we fill in the upper half of the square.

Algorithm: for each span, store the probability of the nonterminals that can occur (things that can't have zero probability).

Lecture 16

Problems with PCFG: independence assumption: P of a word depends only on its tag or it's parent node, but not on other information i the tree. This independance assumption can lead to inaccurate probabilities and the final parse may not be the most optimal. Lack of sensitivity to structural preference. If the P of PP after a NP is very high, there may be multiple NPs in the sentence and only one of them is the most grammatically correct, but all parses for every NP PP will have the same probability.

Head word gives a good representation of phrase structure and meaning. It puts the properties of words back into a PCFG. This gives better estimates of probability.

Therefore for each S - NP VP, replace it by S(headword) - NP (headword) VP(headword) for their respective heads.

Lexicalisation: for each category try to find its head. You can identify heads using rules such as: NP - DT NN (the RHS is more likely to be a noun than the left). Therefore once the head of each context free rule has been identified, lexical items can be propagated bottom-up.

Charniak's model: This is bottom up, but probabilistic conditioning is top-down. S - NP VP, we know head of VP is head of S, but we need to find head of NP. Therefore we use a probability distribution that conditions of the node category, parent category and parent headword. In this way expand each node. However we need to get started: assign a "root" node that goes to S, and then we can have some special probability distributions dependent on S and its headword and so on.

You can do linear interpolation to reduce sparse probabilities. Different interpolated models leave out different headwords and they are weighted based on how much those headwords are expected to have been seen.

(problem: hard to estimate bigram probabilities from small treebanks)

PCFGS and Independence: At any node the nodes below it are independent of material outside it. Any information that connects behavior inside and outside a node must flow through the node.

Overly strong assumptions: rewrites get used where they don't belong.

Therefore refine the grammar symbols: state splitting: you can improve a pcfg by encoding for each non terminal, information about its parent nt.

Problem: too much state splitting leads to too much sparseness.

Unlexicalised parsing: Here grammar rules are not systematically specified to the level of lexical items but you can have some information. Closed vs. open class words: (we're allowed to encode information to distinguish if/whether and verb forms like have/has been. This is different to the bilexical idea of semantic heads)

Horizontal markovisation: merge states (if you have a string of proper nouns or verbs, we don't want to expand to different levels, we would rather know that we have a string of PP). Vertical markovisation: rewrites depend on past k ancestor nodes.

In a basic PCGF we find that unary rules are used often so high probabilities are given. Solution: mark the unary rewrites with U.

Problem: treebank tags are too coarse. Solution: subdivide tags.

Yield splits: sometimes behavior of category depends on something inside its further yield.

Distance/recursion splits: mark states higher up (above the parent)

Lecture 18

IR: Finding material of an unstructured nature from large collections. (Data in structured form: database, lists etc and unstructured: normal text)

Assume you have a static collection of documents. The goal is to retrieve documents relevant to the users query.

Precision: fraction of retrieved documents that are relevant. Recall: Fractions of relevant documents in the collection that are retrieved.

We can use term document matrix notation, where we count the number of times a word occurs in a document and represent the word by that vector. If we then want to find a document that contains word a and word b but not word c, use logical operations on the vectors and the resulting vector shows you (by the 1 and 0 positions) which documents satisfy the criteria. However this is not useful for very large documents and a large number of words.

Inverted index: For each term t, store a list of all documents that contain t. Therefore, for each term keep a postings list i.e. a linked list that contains docIDs of the documents that the word occurs. The postings in the posting list are sorted by docID.

Initial stage of text processing: tokenization, normalization, stemming, stop words.

Indexer steps: Assume two docs and we are given a list of words and the document that they appear in. Sort the words alphabetically, multiple entries in a single document are merged, and if words occur in both lists, create a postings list in sorted order for words.

Query processing: "word1 and word2" Steps: locate word1, retrieve postings, locate word2, retrieve postings, merge the two postings i.e. intersect the document sets. Merge operation: walk through the two postings simultaneously, in time linear in the total number of postings entires. keep pointers to the first of both elements and check if they point at the same, if not, advance the lesser one until it crosses the first. At every match i.e. when pointers point to the same, increment count.

Phrase queries: active area of research: implicit phrase queries? For phrase queries it no longer suffices to use term:doc matrices, because this doesn't give information about adjacency or relative position of terms, just their occurrence in the matrix. One attempt: byword indexes: dictionary consists of word1 word2, word2 word3 for every pair of consecutive words. Problems are false positives and index blowup due to a bigger dictionary.

For a phrase query: get postings list for each word in the phrase. You can use combination schemes: common phrases are considered as bywords.

HMM

f : X -¿ Y maps the input to the output which varies for different functions. For example, speech recognition maps sound waves or audio input to text, chinese=english translation maps chinese characters to english words and so on. Therefore our goal is to learn a function from input x to output y, given a set of training examples (xi, yi). We can use a conditional model i.e. distributive or a generative model. A conditional model defines conditional probability p(y—x) for an x, y pair. We simply take f(x) = argmax over all y (p(y—x)) i.e. take the mostly likely label. If p(y—x) by our model is close to the true conditional distribution then our model is close to optimal.

An alternative approach is a generative model. Instead of p(y—x) find the joint probability p(x, y) over (x,y) pairs. The parameters for the model are estimated from the training examples. p(x, y) = p(y)p(x—y).

Therefore a generative tagging model is a function p such that: for any (x1..xn, y1..yn) in S, p(x1..xn, y1..yn) ¿= 0 summation over all (x1..xn, y1..yn) in S (p(x1..xn, y1..yn) = 1

Therefore if the above is our generative model, the function from x1..xn to y1.. yn is f(xi..xn) = argmax over y1..yn (p(x1..xn, y1..yn)

An HMM is different from a Markov model because it had two additional requirements. It has a separate set of observation symbols O, not drawn from the same set of states Q. Also the observation likelihood function is not limited to values 0 and 1, it can take values in between.

Therefore we have a set of states, transition probabilities between states and observation likelihoods i.e. each expressing the probability of an observation being generated from a state.

Lecture 20

Lemma: the stem that is inflected in different ways Homographs: spelled the same Homophones: pronounced the same

A polysemous word has related meaning (school, bank etc.)

Uses of Wordnet: IR, IE, QA, MT Synset: set of words that have similar meaning Wordnet has hypernym, hyponyms, meronyms

Distributional similarity: Thesauri have problems with recall, words or phrases may not be present, connections between sentences and units in a sentence

Therefore use distributional models: VSM of meaning. In a term-doc matrix, words are similar if they have similar vectors i.e. they appear in the same context. Use a paragraph or a window of some number of words

Use PMI instead of tf-idf: PMI asks if two words w1 and w2 co occur more than they occur independently. It equals log P(x, y)/P(x)P(y)

Lecture 22

Single document summarization: Given one document produce abstract, outline. For multiple: given group, provide gist.

Query focused: summarization based on a question asked about the document (complex form of question answering).

Extractive summary: create summary from phrases in the document. Abstractive: express ideas using different words.

Simple baseline: first sentence. Basic summarization algorithm: 1. content selection (choose sentences to extract) 2. information ordering (use document order itself). 3. sentence realization (keep original sentence)

1. unsupervised content selection: use tfidf, mutual information and log likelihood ratio. topic signature based: choose words that are informative either by LLR or by appearing in the query. then weigh a sentence by the weight of the words.

2. supervised content selection: given a labeled training set of good summaries, align the sentences in document with sentences in the summary. then extract features (position, length of sentence, word informativeness, cohesion). we can then train a binary classifier (should we put the sentence in the summary or no?). problems: training data is hard, alignment is difficult

Evaluation: ROUGE (recall oriented understudy for listing evaluation)

Query focused multi document summarization: Bottom up (snippet method): find a set of relevant documents. extract informative sentences from documents. order and modify sentences into an answer. top down (IR method): build specific answers for different question types.

Coreference: Coreference resolution is to identify all noun phrases or things mentioned that refer to the same entity. Applications: machine translation, text summarization, IE and QA

B$^3 evaluation : Consideralltheclustersformed. Thecolorsofeachiteminaclusterrepresentthegoldstand$

coreference is when two mentions refer to the same entity in the world. Anaphora is when a term (anaphor) refers to another term (the antecedent) and the interpretation of the anaphor is in some way determined by the interpretation of the antecedent

Research: task: generate sentences to describe C£ code snippets and SQL queries. dataset: stack overflow posts and responses tagged with C£, SQL or oracle

Traditional pronominal anaphora resolution: Hobbs naive algorithm: 1. begin at the NP that immediately dominated the pronoun (syntactic parse)

Knowledge based pronominal coreference: require knowledge of the world to attribute reference (city council, permit example). Methods: 1. treat coreference chains as a sequence of pairwise links. Do binary classification with independent pairwise decisions. 2. use a ranking problem: explicitly rank all antecedents 3. entity-mention models: each mention is linked to some discourse entity somewhere. explicitly cluster mentions of the same discourse entity.

Mention-pair model: find if the current mention coercers with the previous mention given the surrounding context. Features: gender or number. syntactic constraints. recency. grammatical role. parallelism.

HMM Ordering

MMR: maximal marginal relevance: iterative method for content selection from multiple documents. Iteratively, greedily choose the best sentence to insert in the summary so far. relevant: maximally relevant to user's query (high cosine sim to query) novel: minimally redundant with the summary so far (low cosine sim to summary)

Combine intuition of LLR + MMR: score each sentence based on LLR, include sentence with highest score in summary. Iteratively add into the summary high scoring sentences that are not redundant with summary.

IR method: build separate templates or classes for each question type and then find specific information in the document and you can use classifiers to extract the information.

Terms

A **Hidden Markov Model** is a statistical Markov model where the system being modeled is assumed to be a Markov chain with unobserved or hidden states. In an HMM the state or the sequence that the states go through is not directly visible, but the output dependent on the state is visible.

Entropy is a measure of the amount of information held. It is the smallest number of bits that can be used to represent some data.

Review

Active learning: A way to decide what data to annotate manually. Steps: Start with small randomly selected data (dataset D1). Train a classifier on D1. Run classifier on D/D1. Find points for which the decision is least certain.

Active learning is a special case of semi-supervised machine learning in which a learning algorithm is able to interactively query the user (or some other information source) to obtain the desired outputs at new data points. In statistics literature it is sometimes also called optimal experimental design.

Project ideas: sense of words like “sunset” or “battleground”