Machine Learning for Language Processing (L101)

Ann Copestake

Computer Laboratory
University of Cambridge

October 2017

Outline of today's lecture

- From last time
- Smoothing
- POS tagging overview
- HMMs for POS tagging
- Imperfect training data
- State-of-the-art in POS tagging

Questions or comments about previous lecture?

Generative models

- NB is a generative model: we train a model of the joint distribution of observations and classes, $P(\vec{f}, c)$.
- Hence, for multinomial NB, this is equivalent to a unigram model.
- Contrast discriminative models, where we train the posterior distribution of the class given the observation $P(c|\vec{f})$
- Also: discriminant functions — we just train a mapping from the observation to the class label without the probability.

From last time

- Vocabulary is a list of all words in the documents (excluding any in a stop list).
- Feature vector $\vec{f}$ for document d: for each item w_i in the vocabulary, generate 1 if w_i is in d, 0 otherwise.
- Estimate $P(f_i|c)$ as the fraction of documents of class c that contain w_i.
- Estimate $P(c)$ as the proportion of documents which have class c.

However, this doesn't work . . .

- Zipf's Law, Heaps' Law/Herdan's Law: no matter how much data we collect (tokens), we will never see all words (types) of the possible vocabulary.
- Hence, there will be words in the test data that are unseen in the training data.
- For these, $P(f_i|c)$ will be estimated as 0.
- Set vocabulary to be only the words in the training data?
- But what about words which only appear in one category in the training data?
  - Is there really a zero probability they should appear in another category?
  - Multiplication in NB means even strong evidence from other words could be ignored.

However, this doesn't work . . .

- Zipf's Law, Heaps' Law/Herdan's Law: no matter how much data we collect (tokens), we will never see all words (types) of the possible vocabulary.
- Hence, there will be words in the test data that are unseen in the training data.
- For these, $P(f_i|c)$ will be estimated as 0.
- Set vocabulary to be only the words in the training data?
- But what about words which only appear in one category in the training data?
    - Is there really a zero probability they should appear in another category?
    - Multiplication in NB means even strong evidence from other words could be ignored.

Additive smoothing

- In Bayesian terms, need a prior distribution (before we look at the training data).
- Simplest option: assume a uniform probability for each word in a vocabulary for each category.
- additive smoothing / Laplace smoothing: add a small pseudocount α to each count:
- add-one smoothing: $\alpha = 1$:

$$\hat{P}(f_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|}$$

where V is the vocabulary (i.e., feature vector dimension)

Additive smoothing, continued

- We don't smooth $\hat{P}(c)$ — why not?
- α is a hyperparameter: determine optimum value experimentally (on development data). Although not strictly allowed if we view this as a prior!
- Choice of V? What do we allow ourselves to know? Can we 'just learn from data'?
- Ristad (1995). Friedman and Singer (1999): hierarchical prior, works for unbounded alphabets.

POS tagging

They can fish.

- They_PNP can_VM0 fish_VVI ._PUN

Lower ranked:

- They_PNP can_VVB fish_NN2 ._PUN
- They_PNP can_VM0 fish_NN2 ._PUN no full parse

tagset (CLAWS 5) includes:

NN1	singular noun	NN2	plural noun
PNP	personal pronoun	VM0	modal auxiliary verb
VVB	base form of verb	VVI	infinitive form of verb

POS lexicon fragment

they	PNP
can	VM0 VVB VVI NN1
fish	NN1 NN2 VVB VVI

- Lexicon could be acquired from a dictionary/grammar.
- Possible tag sequences could also come from a grammar.
- For ML approach, we want to acquire probabilities of tags and tag sequences from data.

Why POS tag?

Not often considered as a task until early 1990s, but much easier and faster than full parsing:

- Preprocessing before parsing to reduce search space or for unknown words.
- Simple source of syntactic features for other tasks: e.g., named entity recognition (NER).

  Sports Direct hit by slide in pound.
- Aiding investigation of language: lexicographers, corpus linguistics.

POS tagging problem task specification

- which language? English? Turkish? Japanese?
- tagset?
- genre? newpaper headlines, chemistry texts etc, etc
- errors in the data?

  He walked in into the room.
- Accuracy for rare words? rare uses of words?

Nearly all published work is on a limited range of standard datasets: fairly small, inconsistencies and errors in annotation. Effect on real task may not correlate well with performance of POS tagger on standard dataset.

POS tagging as a ML problem

- Classification of items in a sequence.
- Almost always treated as supervised learning.
- Available training data is somewhat limited: human annotators require fairly extensive training, annotation guidelines are lengthy, but inter-annotator agreement can be good (especially compared to most semantic tasks).
- Decide on (approximate) model, learn probabilities (efficiently), apply model (efficiently).

Modelling POS tagging as a ML problem

- HMM: Hidden Markov Model — POS tags are hidden states.
- transition probabilities and emission probabilities.
- Standard POS tagging uses HMMs in a simplified way: probabilities taken from annotated corpora (supervised).
- HMMs can be used unsupervised, but performance for POS tagging isn't good.
- Efficient application via Viterbi algorithm.
- Basic model must be augmented with smoothing and treatment of unknown words.

Assigning probabilities

Estimate the sequence of n tags as the sequence with the maximum probability, given the sequence of n words:

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

By Bayes theorem:

$$P(t_1^n | w_1^n) = \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$$

Tagging a particular sequence of words so $P(w_1^n)$ is constant:

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

Approximations

Bigram assumption: probability of a tag sequence approximated by the product of the two-tag sequences:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1})$$

Probability of the word estimated on the basis of its own tag alone:

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

Hence:

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

More details

- Maximise the overall tag sequence probability — use Viterbi dynamic programming (details in J+M).
- Actual systems use trigrams — smoothing and backoff are critical: insufficient data to use 4-grams etc.
- Unseen words.
- Preprocessing: what is a word? formulae etc
- Genre effects: e.g., tag for ‘I’ (chemistry?)

More details

- Maximise the overall tag sequence probability — use Viterbi dynamic programming (details in J+M).
- Actual systems use trigrams — smoothing and backoff are critical: insufficient data to use 4-grams etc.
- Unseen words.
- Preprocessing: what is a word? formulae etc
- Genre effects: e.g., tag for ‘I’ (chemistry?)

Smoothing for POS tagging

- Some tag sequences are possible but rare, words will not be seen with all their possible POS tags.
- Use backoff for tag sequences: trigram counts modified by bigram and unigram counts with appropriate parameter.
- e.g., replace all infrequent words (e.g., count less than 5) with UNK.
- But: rare tags for frequent words?
- Sometimes zero probabilities are correct:
  so tagged as a verb?
  determiner followed directly by a verb?
- Lots of experimentation . . .

Estimating tags for unknown words

- Distribute the probabilities according to the frequence of open class tags.
- But morphology: e.g., word ending in ‘ing’ can’t be VVD.
- Additional features: incorporating into HMM is messy . . .
- Most languages have much richer morphology than English, so can make more use of affixes.
- Also: capitalization etc: ‘Bill’ vs ‘bill’, ‘Gates’ vs ‘gates’.

Improvements to HMMs

- Speed/accuracy trade-off: e.g., ideally want to incorporate word sequence information:

  I have a bad cold . . .
  There is a large cold . . .
- Discriminative models better for proper treatment of additional features (but HMM-based TnT very effective in practice).
- Bidirectional: HMM maximizes over sequence, but fully bidirectional is better.
- Character based models: morphology, capitalization etc.
- Until recently, lots of feature engineering.

POS tagging with LSTMs

Paper by Plank et al (2016), in course readings (details on LSTMs in lecture 7 or 8):

- Different natural languages, different language families.
- LSTMs can make use of pre-trained embeddings (unsupervised).
- Performance is close to the likely ceiling, but still quite low on unseen items in some languages.
- Best LSTM variant clearly better than TnT (c 25% reduction in error rate), but TnT still better with very limited training data.

Question to think about again: what is the task?