Natural Language Processing
Lecture 3

Lecture by Dr. Kfir Bar
Typeset by Steven Karas

2019-03-26
Last edited 20:58:00 2019-03-26

Disclaimer These notes are based on the lectures for the course Natural Language Processing, taught by Dr. Kfir Bar at IDC Herzliyah in the spring semester of 2018/2019. Sections may be based on the lecture slides prepared by Dr. Kfir Bar.

1 Homework

The homework is due tomorrow.

2 Agenda

- Smoothing

3 Smoothing

A naive model is overly specialized, and we want to generalize. Smoothing is an easy way to do that, by simply adding 1 to all the counters.

3.1 Laplace Smoothing

For all possible n-grams, add the count of one:

$$\Pr[w_n \mid w_1^{n-1}] = \frac{C(w_n w_1^{n-1}) + 1}{C(w_1^{n-1}) + V}$$

Where C is a count of n-grams, N is the count of history, and V is the vocabulary size.

In practice, V is often a hyperparameter as some function of the number of types. Similarly, rather than adding 1, we add some smaller λ to avoid punishing very established probabilities.

3.2 Witten-Bell smoothing

An unseen n-gram is one that just hasn't been seen yet. If we were given sufficient data, we would expect to see it. We can estimate this based on the size of our corpus.

Let N be the number of tokens, T be the number of seen types, and V be the number of known types. Define Z as the number of unseen types that are in the vocabulary:

$$Z = \sum_{i:c_i=0} 1$$

This gives the probability of seen unigrams:

$$p_i^* = \frac{c_i}{N+T}$$

The probability of unseen unigrams would then be:

$$p_i^* = \frac{T}{Z(T+N)}$$

Or the bigram case:

$$p^*_{\text{unseen}}(w_i \mid w_x) = \frac{T(w_x)}{Z(w_x)(N(w_x) + T(w_x))}$$
$$p^*_{\text{seen}}(w_i \mid w_x) = \frac{c(w_x, w_i)}{N(w_x) + T(w_x)}$$
$$Z(w_x) = V - T(w_x)$$

4 Backoff

4.1 Linear interpolation

Solve the sparseness in large n-gram models by mixing them with weighted smaller n-gram models:

$$\Pr[w_n \mid w_1 \dots w_{n-1}] = \sum_{i=0}^{n} \lambda_i \Pr[w_n \mid w_{n-i} \dots w_{n-1}]$$

5 Part of speech tagging

We want to tag the POS tags of every word in a given sentence. We use tag sets to classify specific words according to their role in a sentence. For example, part of the Penn Historical Treebank tag set:

Tag	Meaning	Tag	Meaning
NN	Noun	VBS	Verb - present
NNS	Noun - plural	VBD	Verb - past
NNP	Noun - proper	VBP	Verb - progressive
ADV / RB	Adverb	IN	preposition
DET	Definite article	ART	Indefinite article
ADJ	Adjective	...	...

Ambiguity is the primary obstacle to doing this perfectly. For example, "Around" can be either a preposition, a particle, or an adverb. As such, while humans can perform this task with near 100% accuracy, while SOTA achieves close to 97% accuracy.

In the worst case, several taggings are syntactically valid, yet semantically nonsense:

- Time (VB) flies (NN) like (RB) an (ART) arrow (NN)
- Time (NN) flies (VB) like (RB) an (ART) arrow (NN)
- Time (ADJ) flies (NN) like (VB) an (ART) arrow (NN)
- Enraged (ADJ) cow (NN) injures (VB) farmer (NN) with (...) ax (NN) - a similar, yet different issue

5.1 Supervised learning

Given a tagged dataset, learn how to model and predict tags based on the given data. There are many publicly available tagged datasets.

5.1.1 Markov Models

A Markov model is a state machine that assigns a probability to transition to each future state. Such a model satisfies the Markovian property:

$$\Pr[s_{ik} \mid s_{i1}, \dots, s_{ik-1}] = \Pr[s_{ik} \mid s_{ik-1}]$$

Meaning that the current state is only dependent upon the immediately previous one. We can define the adjacency matrix A of the graph as:

$$A_{ij} = \Pr[s_j \mid s_i]$$

It is common and useful to augment this adjacency matrix with start and end sequence states. The probability that we observe a sequence of states $(s_1, \dots, s_t)$:

$$\Pr[s_1, \ldots, s_T] = \prod_{t=1}^{T} A_{s_{t-1}s_t}$$

We will cover the use of MLEs on Markov models later on today.

5.1.2 Hidden Markov Models

Model observations into the model, we get that given a sequence of observations o and model how they map to a sequence of states s:

$$o = (o_1, \ldots, o_T)$$
$$s = (s_1, \ldots, s_T)$$

where the probability of seeing any particular state given an observation:

$$\Pr[s \mid o] = \frac{\Pr[o \mid s] \Pr[s]}{\Pr[o]} = \frac{\Pr[o \mid s] \Pr[s]}{\sum_{s'} \Pr[s', o]}$$

Note that the denominator is constant with regards to s, so we can ignore it. To maximize the probability for sequences observed in a corpus:

$$\max_{A,B} \frac{\Pr[o \mid s; A, B] \Pr[s; A, B]}{\sum_{s'} \Pr[s', o; A, B]}$$

In order to find the most likely state $\hat{s}$ given an observation o:

$$\begin{aligned}
\hat{s} &= \arg\max_s \Pr[s \mid o] \\
&= \arg\max_s \Pr[o \mid s] \Pr[s] \\
&= \arg\max_s \prod_{t=1}^{T} B_{s_t o_t} \cdot A_{s_{t-1}s_t}
\end{aligned}$$

Formal model

$$M = (A, B, \pi)$$

Where A is the state transition adjacency matrix, B is the emission matrix, and π is a vector of initial probabilities.[1]

It's important to remember that HMM does not output pure probabilities, but rather scores that cannot be compared.

5.2 Viterbi algorithm

A dynamic programming algorithm that runs in polytime. Decoding is the process of finding the best state-sequence that generates the observations.

$$\begin{aligned}
\delta_j(t) &= \max_{s_1, \ldots, s_{t-1}} \Pr[s_1 \ldots s_{t-1}, o_1 \ldots o_{t-1}, s_t = j, o_t] \\
\delta_j(t+1) &= \max_i \delta_i(t) a_{ij} b_{jo_{i+1}} \\
\delta_j(1) &= \pi_j b_{jo_1} \\
\psi_j(t+1) &= \arg\max_i \delta_i(t) a_{ij} b_{jo_{t+1}}
\end{aligned}$$

5.3 Unsupervised learning

Given a dataset without tags, learn features that model patterns.

[1] It is possible we accidentally swapped the order of the matrix multiplication

6 Next Week

References

[1] Yoav Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[2] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., 2009.

[3] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.