NLP - Master Notes

June 12, 2021

1 Language Models

1.1 N-Gram Models

- Language (prediction) models which make the *Markov assumption* for an $(n-1)^{th}$ order Markov Model; i.e. that only the previous n-1 words have a probabilistic dependence on the current word.
- Probability of words 1 to n: $P(w_1^n) = \prod_{k=1}^{n} P(w_k | w_{k-N+1}^{k-1})$

  General steps for creating an n-gram model:

  1. choose a vocabulary
  2. add $< s >$ and $< /s$ symbols
  3. replace unknown words in the training corpus with $< UNK >$
  4. calculate probabilities (on an as needs basis?)
  5. calculate most probably words in order (until reaching end of sentence symbol) OR evaluate perplexity of test corpus using above formulas.
- *example:*

  Text:
  One cat sat. Three **cats sat**. Eight **cats sat**. The **cats** had nine lives.
  $\text{P(sat | cats sat)} = \frac{C(cats,sat)}{C(cat)} = \frac{2}{3}$
- **Definition of a language model**: a model which assigns probabilities to sentences, based on the training corpus. The sum of all the probabilities of all possible sentences (of arbitrary length) should equal 1.

  Trivial example:

  - Training corpus contains two sentences:
    1. "a b",
    2. "b a"

- Append $< sos >$ and $< /sos >$ to each:
  1. "s a b /s"
  2. "s b a /s"
- generate the probabilities of a bigram (N=2) language model:

  $P(a|s) = \frac{1}{2}$
  $P(b|s) = \frac{1}{2}$
  $P(b|a) = \frac{1}{2}$
  $P(a|b) = \frac{1}{2}$
  $P(/s|a) = \frac{1}{2}$
  $P(/s|b) = \frac{1}{2}$

- To calculate the probability of ALL possible sentences, we take the prob of all sentences of length 1, all sentences of length 2, etc. Note when calculating this, the TEST sentences need to include $< s >$ and $< /s >$
- For example: P(a) = P($< s > a < /s >$) = $P(a| < s >) * P(< /s >$ $|a) = 1/4$
- Probability is same for b, so the sum of all probabilities of sentence length 1 = $\frac{1}{2}$
  The sum of all sentences will be the infinite series:

$$P(all) = \frac{1}{2} + \sum_{i=2}^{\infty} \frac{i!}{(i-2)!} \frac{1}{2}^{i+1}$$

  Does this sum to 1? A proof would be cool.

- Sentence Generation: until you produce a $< /s >$ symbol, continually generate words using: $argmax_(w_k) \frac{C(w_{k-n+1},...,w_k)}{C(w_{k-n+1},...,w_{k-1})}$
- Perplexity: how well a model fits the data $PP(W) = P(w1, ...w_N)^{-\frac{1}{N}}$, for N words in the test corpus. A perplexity of 1 would be the lowest possible.
- Smoothing:
  - Laplace (add-one): $P(W_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w)+1}{c(w_{n-N+1}^{n-1})+V}$, where V is the vocabulary size
  - add δ, normalise by δV
- Interpolation: creating a linear combination of n-gram models of varying n: $\hat{P}(w_i|w_{i-1}, ...w_{i-n}) = \sum_{j=2}^{n} \lambda_j * P(w_i|w_{i-1}, ..., w_{i-1-j})$, where $\sum_j \lambda_j = 1$
- Back-off: back-off to lower n models until data is available (does this mean you can have arbitrary n?)

2 Part of Speech Tagging

A Part of speech tagger assigns for every word w_i in a sentence $\{w_i, ..., w_n\}$ a part of speech (POS) tag y_i. POS tags include parts of speech like verb (tag=V), noun, adverb etc.

Naive Baseline: More than half of words are ambiguous - i.e. they have more than one possible tag. If we count all the tags in a training set and produce $P(tag|word)$ for each word, we can simply choose the most likely tag for the word. This baseline has an accuracy of about 92%, only 5% less than the state of the art.

All extra efforts in more complex models (e.g. CRF, HMM) are about squeezing out this last 5%.

Named entity tagging (e.g. Washington State) is harder - it assumes groups of words together refer to one particular proper noun.

2.1 Hidden Markov Models

Suppose there are T tags (hidden states). An HMM will require a transition matrix giving the probability of any tag occurring after any other tag. This will be an T x T matrix of probabilities (we can compute this similarly to the bigram model, using the bigram assumption)
Suppose also there are N words. The HMM will also require probabilities of each word given a tab. This will be a T * N matrix. Again, this is computed using counts of word-tag pairs over total tag counts.
"The goal of part-of-speech tagging is to find the most probably sequence of $t_1...t_n$ tags given a sequence of N words $w_1...w_n$."
We achieve the goal of the HMM by the following equations:

$$
\begin{aligned}
t_{1:n} &= argmax_{t_1...t_n} P(t_1...t_n|w_1...w_n) && \text{max prb tags given words} \\
t_{1:n} &= argmax_{t_1...t_n} \frac{P(w_1...w_n|t_1...t_n)}{P(w_1...w_n)} && \text{by bayes rule} \\
t_{1:n} &= argmax_{t_1...t_n} P(w_1...w_n|t_1...t_n) && \text{since argmax, can discard denom} \\
t_{1:n} &= argmax_{t_1...t_n} \prod_{i=1}^{n} P(w_i|t_i) \prod_{i=1}^{n} P(t_i|t_{i-1}) && \text{by the HMM assumptions}
\end{aligned}
\tag{1}
$$

We are interested in a tag sequence satisfying the above equation. Brute forcing would be possible, but would take $O(T^N)$ (infeasible).

The viterbi algorithm attempts to solve this problem in a feasible time.

3 Statistical Parsing

3.1 Overview

3.2 Parse Trees, CNF, CFG, PCFG

Parse Trees: show the groupings of words in a sentence to their syntactic group. E.g. Noun Phrase (NP), Verb Phrase (VP). This helps downstream NLP tasks to determine the meaning of sentences, perform translations, etc.

Converting to Chomsky Normal Form: Chomsky Normal Form means that the right hand side of each rule must expand to two non terminals, or one terminal. [1]

1. Copy all conforming rules to the new grammar unchanged
2. Convert terminals within rules to dummy non-terminals
3. Convert unit productions
4. Make all rules binary and add them to new grammar.

PCFG: each rule is assigned a probability, and the sum of all probabilities per non terminal (on the left hand side of the rule) = 1.

3.3 CKY Algorithm, Probabilistic CKY

CYK is a **bottom up** parser - i.e. we start with the words and build the tree to the top.

CKY algorithm requires a grammar to be normalised to Chomsky Normal Form. This will naturally result in an (likely unbalanced) binary tree when expanded.

The **time complexity** of CKY parsing is $O(n^3)$, since there are n^2 cells to fill, and each cell requires querying n split points (*not 100% sure about this point*).

References

[1] D. Jurafsky and J. H. Martin, "Speech and language processing (draft)," *Chapter A: Hidden Markov Models (Draft of September 11, 2018). Retrieved March*, vol. 19, 2018.