NLP - Master Notes

June 16, 2021

1 Introduction

2 Automata and Regex

3 Language Models

3.1 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.
  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): $P(W_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w)+1}{c(w_{n-N+1}^{n-1})+V+1}$, 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?)

4 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.

4.1 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) called the **transition matrix**
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, called the **emission matrix**
"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:

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

Note: we also require an array of initial probabilities for each of the states.

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.

The viterbi algorithm takes in the following inputs:
O: observation space, S: state space, π: initial probabilities, Y:sequence of observations, A: transition matrix, B: emission matrix.

Algorithm 1 Viterbi

procedure VITERBI(O, S, π, Y, A, B)
    $P \leftarrow initMatix(rowSize = K, colSize = K)$ ▷ for max probabilities
    $M \leftarrow initMatix(rowSize = K, colSize = K)$ ▷ for argmax of above
    for state i =1,2,...,K **do** ▷ Assign the first set of probabilities
        $P[i, 1] = \pi_i * B_{iy_1}$
        $M[i, 1] = 0$
    ▷ Prob = max of previous max states leading to this state and evidence
    for observation j = 2,3,...,K **do**
        for state i = 1,2,...K **do**
            $P[i, j] = max_k(P[k, j-1] * A_{ki} * B_{iy_i})$
            $M[i, j] = argmax_k(P[k, j-1] * A_{ki} * B_{iy_i})$
    ▷ FinalState = max of final probs
    $z_T = argmax_k(P[k, T])$
    ▷ Backtrace through back pointers to construct output
    $X \leftarrow initArray(size = T)$
    $X[T] = z_T$
    for j = T, T-1, ..., 2 **do**
        $X[j-1] = S_{M[X[J],j]}$
    $return(X)$

4.2 Unsupervised HMM Learning for POS Tagging

Given *only* a known number of states and an un-labelled corpus (a series of observations), we wish to estimate the parameters of a hidden markov model which maximise the probability of seeing those states.
We can use maximum likelyhood principle to perform expectation maximisation.
We can also perform semi-supervised learning to exploit small amounts of labelled training data.

5 Statistical Parsing

5.1 Overview

5.2 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.

5.3 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*).

6 Statistical Machine Translation

Linguistic issues with machine translation:

- **Lexical divergence** - not all words in one language have a word in the other
- **Syntactical divergence** (e.g. SVO vs SOV)
- **Morphology** - languages with polysynthetic structures make it harder to represent the language as sets of words, because a single word can represent an entire sentence.

Vauqouis Triangle:

- Words
- parsing - syntactic structure
- SRL & WSD - Semantic Structure
- Semantic Parsing - Interlingua

7 Neural Machine Translation

The key difference between Neural and Statistical Machine Translation (NMT, SMT) is that SMT treats the problem of translation as a word to word alignment problem, where the alignment is a latent variable. NMT treats the entire source to target sentence as a more complicated function.

Key areas:

- Perceptrons are good for learning at the character level, good for linearly seperable data, however cannot solve certain problems (e.g. XOR)
- Feed forward neural networks can be used for learning characters using CNNs (e.g. MNIST)
- Sequence to Sequence (seq-2-seq) models are conditional language models
- LSTMs and GRUs are recurrent units which can perform sequence to sequence translation with memory. Hidden state of LSTM, for instance, can show embedded context of phrases.
- Attention: these models learn which components of each sequence are relevant to each other. Higher layers learn semantics, lower layers are better and morphology etc.
- ELMO: creating context specific embeddings for weighted task specific embeddings to be fed into the system.
- Transformers: Attention is All You Need: similar to CNN but no recursion. Embeddings are positional.
- BERT: deep bidirectional transformer
- ELMO and BERT can also be used for word sense dissambiguation, coreference resolution.
- BERT: good for some things, but cannot reason about it's knowledge (it cannot draw logical inferences from sentences). Struggles representing numbers, Representations are hierarchical rather than linear.

8 Semantic Parsing

8.1 Meaning Representation

Question: how to do question answering of natural language (**NL**)?
Entailment for question answering:

$$FOL \leftarrow parse\_to\_FOL(NL)$$

$$answer \models FOL$$

$$answer \leftarrow parse\_to\_english(answer)$$

Model Theoretic Semantics - defines the meaning of formal logic byp providing the set theoretic extensions of symbols (e.g. an interpretation of their extension to the real word/ model).

8.2 Semantic Role Labelling (SRL) Concepts

E.g. nominative and accusative. Some possible roles include: *Agent, Patient, Instrument, Beneficiary, Source, Destination* (roles aren't limited to people, they can be assigned to places, things etc.)

There *can* be syntactic clues (e.g. preposition words - "with", "for", "from" etc.), however ultimately very difficult to do.

Selectional Restrictions: by assigning a role to a concept, we imply certain characteristics of the concept. E.g.: a thing which is an "agent" should be animate (and not inanimate). Taxonomic abstraction hierarchies can be used to determine if such constraints are met (e.g. human = mammal = vertebrate = animate).

8.3 SRL Approaches

1. *Idea:* first apply a syntactic parse tree, then label the concepts
SRL $\subset$ *Sequence Labelling*
Inputs = Parse Tree Nodes
Classifier: labels the parse tree nodes with roles

Features for classifier:

- Paths from the candidate node to be labelled to the predicate along the parse tree.
- position: does the candidate preceed or follow the predicate?

- voice: active or passive?

Issues: relies on the correctness of syntactic parsing, which often has errors. Parse Tree (errors) — Classifier based SRL (more issues)

Semantic roles can be indicated by syntactic features/ locations of words in sentence, but this is not generally useful as there are many exceptions.

Selectional Restrictions: Some roles place restrictions on other roles: e.g. the patient of 'eat' should be a type of food.

8.4 Predicate Calculus / Functional Query Language

$$\text{Sentence} \implies \text{Predicate Logic} \tag{2}$$

Sentence: What is the smallest state by area?
Predicate Logic: answer(x_1, smallest(x_2,(state(x_1), area(x_1,x_2))))

$$\text{Predicate Logic} \implies \text{Functional Query Language (variable free)} \tag{3}$$

FOL: answer(x_1, smallest(x_2,(state(x_1), area(x_1,x_2))))
FQL: answer(smallest_one(area_1(state(all))))

8.5 Composing Meaning Representations from Parse Trees

Need to review this material - second half of lecture 8

9 Lexical Semantics

9.1 Word Sense Dissambiguation

Word Sense Disambiguation: which meaning is the word referring to?

WordNet is a lexical database linking words to nodes (meanings) with various relationships, e.g.: antonyms, attributes, similarness, causes, entailment, hyponym (plant is a hyponym of tree as it is a more specific instance).

How to achieve word sense dissambiguation:

1. Get list of possible senses of (word, POS) from WordNet
2. Encode the context of the word
3. Train a classifier on this data (input = word, POS, contect, output = word sense).
4. does this mean we need seperate classifiers per word?

Feature Encoding For Word Context:

- unordered surrounding bag of word (in sparse vector form) - gives general topic
- POS of neighbouring words
- "Local Collocations" - immediate neighbours of a word ot give context

9.2 The Yarowsky Algorithm

Semi-supervised method which first trains on the supervised data, then repeatedly expands the set of "supervised" training data by labelling unsupervised data for which the classifier is extremely confident. Then, the classifier re-trains with the larger supervised set until it is complete. At each iteration, increase the number of features ("seed words").

9.3 Coreference Resolution

The task of dissambiguating which pronouns refer to the same entity. E.g. **I** took **my** cat to the vet. Here, I and my refer to the same thing.

Steps: 1. detect the mentions of things (look for pronouns). 2. Cluster the mentions (challenging!).

Difficult recognition cases: "**It** is sunny". Here, "it" is a reference we should care about, but it's not a pro-noun, propper noun etc. Should we filter out singleton mentions to make clulstering easier?

Could also do reference detection and coreference resolution end to end.

Anaphora: specifically when pronouns refer to previously refered concepts

9.4 Hobb's naive algorithm

10 Vector Space Semantics

10.1 Word Weighting

The first idea is that we have a corpus of documents, and we want to represent the documents in single vectors. Then we can compare documents as comparing two vectors. To start with, the dimension of the vector will be the size of the pre-processed vocabulary derived from the corpus. Then, we can sum the one-hot encoding of each weighted word. Weighting is defined below:

- Weights: term which appear more frequently are more important. should normalize term frequency by dividing by the most common term in the document. $tf_{ij} = \frac{count(word_i \ in \ doc_j)}{max_i(word_i \ in \ doc_j)}$.
- Inverse document frequency: $idf_i = log_2(\frac{N}{df_i})$, where N is the number of documents, df_i is the number of documents containing term i. Log is for smoothing. Higher values for more unusual terms. $idf_i \in [0, \infty]$. E.g. when a word i is in literally every document, $idf_i = 0$. If a term is in every 16 documents, $idf_i = 4$
- Typical tf-idf weighting: $w_{ij} = tf_{ij} idf_j$ for word i in document j.
- Cosine Similarity:

$$cossim(\hat{a}, \hat{b}) = \frac{\hat{a} \cdot \hat{b}}{|\hat{a}| \cdot |\hat{b}|}$$

$$cossim(\hat{a}, \hat{b}) = \frac{\sum_i^{len(\hat{a})} \hat{a}_i * \hat{b_i}}{\sqrt{\sum_i^{len(\hat{a})} \hat{a}_i * \sum_i^{len(b)} \hat{b_i}}}$$

Distributed Representations: Bag of words approach above makes extremely large sparse matrices. Need a more compact way of representing...

PCA or SVD can achieve this. Basically: which dimensions capture the most variance/ information within the data?

10.2 Word-2-Vec:

The naive way above of representing words from a dictionary as a vector with one-hot encoding (sparce binary matrices with dictionary size) is a poor use of memory for single words or ordered words, but reasonable if we want to represent dense un-ordered sets of words.

What it doesn't do however is capture any useful information about the individual words and how they are related. Isn't **orange** similar to **apple** more than it is similar to **exuberantly**?

It turns out by using a variation of a *neural autoencoder* to predict *context words* we can create vector representations of words which are both **more compact** (smaller dimension) and **geometrically useful**. By the later point, I mean to say **apple** and **orange** will have a smaller (some metric) distance than apple and **exuberantly**.

The two variations of this auto-encoder, *Skip-Gram* and *CBOW*, both produce the same thing, but use different approaches to training.

The assumption which both of these methods use is that similarity of words can be learned by taking immediate context words as relevant information. This does mean we loose the ability to consider long term dependencies, however word-2-vec is more of a corpus wide average than a question answering system, so it should be a fair assumption to make.

SkipGram

- Input: word to be embedded
- Output: surrounding context words
- Good for: Higher dimensions, large corpus

CBOW

- Input: surrounding context words
- Output: word to be embedded
- Good for: smaller corpus, faster training

10.3 Keyword Extraction

This section biefly covers the analysis of web pages by keywords - a task Google pioneered.

Pages are important if they are linked to by other important pages.

Page Rank Value for a page u, given all the linked pages B_u and the number of links from page v (i.e. the number of things v points to).

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

This algorithm converges after unsupervised iterations from random initial values.

Adding a damping factor:

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Text rank applies a similar idea to text processing. We construct a graph where the nodes are words and the links are relationships between words within the corpus.

RAKE (rapid automatic keyword extraction) algorithm creates this graph so that we can train the model:

- break up corpus into sentences (candidate keywords)
- build a word co-occurence graph
- calculate scores for words based on frequency of the vertices in the graph (like with page rank)

11 Discourse

Linguistic Devices: Cue phrases, paragraphs, content flow.
Discourse relations: ways in which sentences relate to each other.
Types:

- Explanation: Provides explanation for previous sentence. (what was the cause from the effect)
- Elaboration: Adding more information to a previous sentence.
- Result: A thing happened as a result of the previous sentence (what is the effect of the cause).
- Parallel: Extra information
- Occasion: What was the situational motivation for doing something?

We can construct a binary tree using nodes as these discourse types, and leafes as sentences and other nodes. We could also constructa a graph, a flat structure etc. Multiple theories of discourse.

Rhetorical Structure Theory (RST)
Considers nucleus's and satellites. The nucleus is more important. Elementary Discourse Units (EDU) form satelites. Nucleus can be a group of related satelites / nucleus's.

PDTB - flat structure, WSJ, larger database

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.