COMS 4705 Notes

Weifan Jiang
wj2301@columbia.edu

1/18/2020

MOD 1-2

Definition of NLP

Natural Language Processing: NLU (understanding): computers process text input $\rightarrow$ NLG (generation): computers produce language to communicate

NLP application:

- Machine translation
- Information extraction (NLU): produce database-structured information from text input; useful for complex searches, statistical queries
- Text summarization
- Dialogue Systems

Basic NLP Problems

- Tagging: map strings to tagged sequences (part-of-speech tagging, name-entity recognition)
- Parsing: change sentense to a parse tree representing its grammar structure

NLP hardness

Ambiguity:

- At acoustic level: phrases with similar pronunciation
- At syntactic level: different grammar structure $\rightarrow$ different interpretations
- At semantic (meaning) level: word sense ambiguity
- At discourse level: anaphora (pronounce can refer to multiple discourse entities)

MOD 3: Languaging Modeling

The language modeling problem:

INPUT: $\mathbb{V} = \{\text{a set of words}\}$, $\mathbb{V}^\dagger = \{\text{sentences made by words in } \mathbb{V}\}$

Training data: a set of valid sentences in English

Output: a probability distribution p such that

$$\sum_{x \in \mathbb{V}^\dagger} p(x) = 1, p(x) \geq 0 \forall x \in \mathbb{V}^\dagger$$

Good language model $\rightarrow$ high probability to sentences more likely in English

Motivation of language modeling:

- speech recognition (similarly: optical character recognition, handwriting recognition)
- estimation techniques useful for other problems in NLP (recognize speech vs. wreck a nice beach)

A naive method for language modeling:

INPUT:

- N training sentences
- for any sentence x, $c(x)$ is the number of times this sentence is seen in training data

A naive estimate: $p(x) = \frac{c(x)}{N}$

Deficiency: assign probabality 0 to unseen sentences; no ability to generalize to unseen data

Goal: build models that generalize to new sentences

Markov Processes

Chain rule: $P(A, B) = P(A) \times P(B|A)$

First-order Markov assumption: $P(X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \approx P(X_i = x_i | X_{i-1} = x_{i-1})$, each variable is only dependent on one previous variable, (similarly we have second order Markov assumption)

First-order Markov Process:

$$
\begin{aligned}
&P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) \\
=&P(X_1 = x_1)\Pi_{i=2}^{n} P(X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) && \text{[Chain rule]} \\
\approx&P(X_1 = x_1)\Pi_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1}) && \text{[Markov assumption]}
\end{aligned}
$$

Second-order Markov Process:

$$
\begin{aligned}
&P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) \\
\approx&P(X_1 = x_1)P(X_2 | X_1 = x_1)\Pi_{i=2}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) && \text{[Markov assumption]} \\
=&\Pi_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})
\end{aligned}
$$

assumes $x_0 = x_{-1} = *$, special “start” symbol.

Trigram Language Model

A *trigram language model* consists of:

- A finite set $\mathbb{V}$ (of vocabulary)
- A parameter $q(w|u, v)$ for each trigram u, v, w such that $w \in \mathbb{V} \cup \{STOP\}$, $u, v \in \mathbb{V} \cup \{*\}$

For any sentence $x_1x_2 \dots x_n$, where $x_i \in \mathbb{V}$ for $1 \le i \le n-1$ and $x_n = STOP$, the probability of the sentence under the trigram model is:

$$p(x_1x_2 \dots x_n) = \Pi_{i=1}^{n} q(x_i|x_{i-1}, x_{i-2})$$

where we define $x_0 = x_{-1} = *$.

We treat sentences as generated by the second-order Markov process, which each word is generated only depend on two previous words.

Evaluating a Language Model: Perplexity

Test data: m unseen sentences: $s_1, s_2, \dots, s_m$

Perplexity $= 2^{-l}$ where $l = \frac{1}{M}\sum_{i=1}^{m} \log_2 p(s_i)$, and M is the total number of words in test data set.

Perplexity is a measure of effective branching factor; lower perplexity implies better fitting of language model to test dataset.

MOD 4: Estimation techniques for trigram language models

Maximum likelihood estimate

Trigram maximum-likelihood estimate:

$$q_{ML}(w_i|w_{i-2}, w_{i-1}) = \frac{COUNT(w_{i-2}, wi-1, w_i)}{COUNT(w_{i-2}, w_{i-1})}$$

sparse data problem: $|\mathbb{V}|^3$ possible trigrams while actual number of trigrams present in training data is limited; result in many counts being 0 (which causes parameter values to be undefined or 0).

Bigram maximum-likelihood estimate:

$q_{ML}(w_i|w_{i-1}) = \frac{COUNT(wi-1,w_i)}{COUNT(w_{i-1})}$

Unigram maximum-likelihood estimate:

$q_{ML}(w_i) = \frac{COUNT(w_i)}{COUNT()\text{ total number of words}}$

Bias-Variance trade-off in maximum-likelihood estimators: unigram estimater will converge quickly to true distribution, but fails to capture context (since not conditioned on previous words); trigram estimator can capture context, but since many counts are 0, it requires large amount of training data to converge slowly.

Linear Interpolation

Let

$$\begin{aligned} q(w_i|w_{i-2}, w_{i-1}) = \quad & \lambda_1 \times q_{ML}(w_i|w_{i-2}, w_{i-1}) \\ & +\lambda_2 \times q_{ML}(w_i|w_{i-1}) \\ & +\lambda_3 \times q_{ML}(w_i) \end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$ and $\lambda_i \ge 0$ for all i.

linear interpolation validation as probability distribution

Given $\mathbb{V}' = \mathbb{V} \cup \{STOP\}$, can shown by algebra that:

- $\sum_{w \in V'} q(w|u,v) = 1$
- $q(w|u,v) \geq 0$ for all $w \in \mathbb{V}'$.

estimate the weights (λ values)

Validation set: hold out a part (say 5% of training set as the validation set.

Let $c'(w_1, w_2, w_3)$ to be the number of times the trigram (w_1, w_2, w_3) is seen in the validation set.

Choose $\lambda_1, \lambda_2, \lambda_3$ that maximizes $L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1, w_2, w_3} c'(w_1, w_2, w_3) \log q(w_3|w_1, w_2)$

Maximization of L also minimizes perplexity.

vary λ values

Take a function that partitions histories by counts:

$$\Pi(w_{i-2}, w_{i-1}) = \begin{cases} a \text{ IF COUNT}(w_{i-2}, w_{i-1}) = 0 \\ b \text{ IF COUNT}(w_{i-2}, w_{i-1}) = 1 \\ \dots \end{cases}$$

Introduce dependencies of λ values on the partition: when calculating $q(w_i|w_{i-2}, w_{i-1})$, use respective $\lambda_1^{\Pi(w_{i-2}, w_{i-1})}, \lambda_2^{\Pi(w_{i-2}, w_{i-1})}, \lambda_3^{\Pi(w_{i-2}, w_{i-1})}$ values.

Effect: λ varies depend on which partition the bigram falls into (thus vary depends on COUNT).

MOD 6

The tagging problem

- Part-of-speech recognition: take a sentence as input, assign a part-of-speech (Noun/Verb/Preposition/Adverb/Adjective) to each word.

  Problem: word may have multiple possible part-of-speech (profit can both be noun or verb)
- Named entity recognition: take a sentence as input, identify named entity (Company, name, location) in the sentence.

Each named entity recognition problem can map to a part-of-speech recognition problem by using tags representing named entities.

Tagging Problem can be set up as a supervised learning problem:

Training set: sentences with their part-of-speech tags.\ Goal: learn a function that takes a sentence as input, maps it to its tagging sequence.

Constraints in (any) tagging problem:

- Local: A word is more likely to be one part-of-speech than another
- Contextual: Some tag sequence is more likely than others

Conditional Models

Goal: learn a functioin f mapping inputs x to label $f(x)$

- Learn distribution $p(y|x)$ from training examples
- For any test input x, define $f(x) = argmax_y p(y|x)$.

Conditional models directly estimates $p(y|x)$: "discriminative"

Generative Models (alternative to conditional models)

- Learn a joint distribution $p(x, y)$ over training examples (usually $p(x, y) = p(y)p(x|y)$ prior $\times$ conditional generative model)
- Then by Bayes Theorem: $p(y|x) = \frac{p(y)p(x|y)}{p(x)}$ where $p(x) = \sum_y p(y)p(x|y)$.

Output of generative model:

$$\begin{aligned} f(x) &= argmax_y p(y|x) \\ &= argmax_y \frac{p(y)p(x|y)}{p(x)} \\ &= argmax_y p(y)p(x|y) && \text{Since } p(x) \text{ is constant} \end{aligned}$$

Hidden Markov Models: an instance of generative modeling approach

Basic Approach:

- Input sentence: $x = x_1 x_2 \dots x_n$
- Use a HMM to define $p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$ for any sentence x, and tag sequence y of the same length
- The most likely tagged sequence for x is $argmax_y p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$

Trigram Hidden Markov Models:

For any sentence $x = x_1 x_2 \dots x_n$ where $x_i \in \mathbb{V}$ for $1 \le i \le n$, and any tag sequence $y = y_1 y_2 \dots y_n y_{n+1}$ where $y_i \in \mathbb{S}$ for $1 \le i \le n$ and $y_{n+1} = STOP$, the joint probability of sentence and tag is:

$$p(x_1, \dots, x_n, y_1, \dots, y_{n+1}) = \Pi_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \Pi_{i=1}^{n} e(x_i | y_i)$$

assuming $x_0 = x_{-1} = *$.

Parameters:

- Trigram: $q(s|u, v)$ for $s \in \mathbb{S} \cup \{STOP\}$ and $u, v \in \mathbb{S} \cup \{*\}$.
- Emission parameters: $e(x|s)$ for any $s \in \mathbb{S}$, $x \in \mathbb{V}$.

hidden markov model parameter estimation

Smoothed Estimation:

- $q(Vt|Dt, JJ) = \lambda_1 \times \frac{Count(Dt,JJ,Vt)}{CountDt,JJ} + \lambda_2 \times \frac{Count(JJ,Vt)}{Count(JJ)} + \lambda_3 \times \frac{Count(Vt)}{Count()}$ with $\lambda_1 + \lambda_2 + \lambda_3 = 1$ and all $\lambda \ge 0$.

  (Similar to linear interpolation method in trigram model parameter estimate.)

- $e(Base|vt) = \frac{Count(Vt,Base)}{Count(Vt)}$

Problem: Low frequency word If x is never seen in training data, $e(x|y) = 0$ for all y, which makes $p(x,y) = 0$ for all tag sequences y.

Solution:

1. Split vocabulary into two sets: frequent words (count ≥ 5 in training); low frequency words (all other words)
2. Maps low frequency words to a small finite set (typically depends on spelling features: prefixes, suffixes, etc.)

Then, we can compute values such as $p(firstword|Noun)$ to resolve the issue of low frequency word. This method is simple, but clearly heuristic, human expertise is needed to design the mapping from low frequency words to smaller sets.

pros and cons of HMMs

Pros:

- HMM taggers are simple to train (compile counts from training corpus).
- Perform well (over 90% on named entity recognition)

Cons:

- Modeling $e(word|tag)$ can be difficult if "words" are complex. (solves with the heuristic method of grouping low-count words, not most ideal)

The Viterbi Algorithm: find optimal solution to hidden markov models.

Motivation: Finding the tag sequence $y_1 \dots y_n$ that maximizes $p(x_1 \dots x_n, y_1 \dots y_n)$ by brute force searching through all sequences is inefficient: the number of possible tag sequences is exponential: $|\mathbb{S}|^{|\mathbb{V}|}$

The Algorithm:

- n: length of sequence
- S_k: set of possible tags at position k ($S_{-1} = S_0 = \{*\}$, $S_k = \mathbb{S}$ for other k values)
- $\pi(k, u, v)$: max possibility of tag sequence ending in u, v at position k.

Base Case: $\pi(0, *, *) = 1$

Recursive Case: for $k = 1 \dots n$, for $u \in S_{k-1}$, $v \in S_k$:

- $\pi(k, u, v) = \max_{w \in S_{k-2}}(\pi(k-1, w, u) \times q(v|w, u) \times e(x_k, v))$
- $bp(k, u, v) = arg\max_{w \in S_{k-2}}(\pi(k-1, w, u) \times q(v|w, u) \times e(x_k, v))$

Result:

- Optimal probability: $\max_{u \in S_{n-1}, v \in S_n}(\pi(n, u, v) \times q(STOP|u, v))$
- Optimal tag sequence:
  - $(y_{n-1}, y_n) = arg\max_{(u,v)}(\pi(n, u, v) \times q(STOP|u, v))$
  - For $k = n-2, n-1, \dots, 1$, $y_k = bp(k+2, y_{k+1}, y_{k+2})$

  Return $y_1 \dots y_n$ as the optimal tag sequence

Runtime: $O(n|\mathbb{S}|^3)$, linear on sequence length (much better than brute force search, which is exponential on sequence length)