CS846
Machine Learning for Software Engineering

Pengyu Nie

Language Modeling and N-gram Models

Language models

N-gram models

Perplexity

Acknowledgements: many slides adapted from Jessy Li & Milos Gligoric's ECE-W382V at UT Austin

Natural Language Examples

- Speech recognition. Suppose it is noisy and you hear someone say something that sounds like
  - Theater owners worry about popcorn sales. OR
  - Theater owners worry about unicorn sales.
- A person will probably assume the first option is what was actually said. This is a sentence that has higher probability.
  - Is it more likely to talk about popcorn or about unicorns (if no other info is available)?
  - Which is more similar theater:popcorn or theater:unicorn?

Natural Language Examples

- Spelling correction
  - I can stop by tomorrow.
  - I cab stop by tomorrow.

- The second sentence is less likely. So not only we can figure out that there is possibly an error there, but also what was the intended correct word.

Natural Language Examples

- Predicting next word
  - Once upon a …
  - I'd like to make a collect …
  - Let's go outside and take a …

Programming Language Examples

- Predicting next token
  - `import numpy as …`
  - `static public void main ( String [ ] args ) …`
- Coding convention
  - `String userInput = new Scanner(System.in).next();`
  - `String user_input = new Scanner(System.in).next();`
  - `String userInput = new Scanner(System.out).next();`

Probabilistic Language Modeling

- Compute the probability of a sequence of words/tokens:
  - $P(W) = P(w_1, w_2, w_3, \ldots, w_T)$
- Compute probability of an upcoming word
  - $P(w_5|w_1, w_2, w_3, w_4)$
- A model that computes either of these $P(W)$ or $P(w_t|w_1, w_2, \ldots w_{t-1})$ is called a language model

Chain Rule of Probability

- $P(W)$ or $P(w_t|w_1, w_2, \dots w_{t-1})$

$$
\begin{aligned}
&P(W) \\
&= P(w_1, w_2, \dots, w_{T-1}, w_T) \\
&= P(w_T|w_1, w_2, \dots, w_{T-1}) \cdot P(w_1, w_2, \dots, w_{T-1}) \\
&= \cdots \\
&= \prod_{t=1 \dots T} P(w_t|w_1, w_2, \dots, w_{t-1})
\end{aligned}
$$

W = (import, numpy, as, np)

$$
\begin{aligned}
&P(W) \\
&= P(import, numpy, as, np) \\
&= P(np \mid import, numpy, as) \cdot P(import, numpy, as) \\
&= P(np \mid import, numpy, as) \cdot P(as \mid import, numpy) \\
&\cdot P(import, numpy) \\
&= P(np \mid import, numpy, as) \cdot P(as \mid import, numpy) \\
&\cdot P(numpy \mid import) \cdot P(import)
\end{aligned}
$$

N-gram Language Model, Markov Assumption

$$
\begin{aligned}
&P(W) \\
&= P(w_1, w_2, \dots, w_{T-1}, w_T) \\
&= P(w_T | w_1, w_2, \dots, w_{T-1}) \cdot P(w_1, w_2, \dots, w_{T-1}) \\
&= \cdots \\
&= \prod_{t=1 \dots T} P(w_t | w_1, w_2, \dots, w_{t-1})
\end{aligned}
$$

$$
P(w_t | w_1, w_2, \dots, w_{t-1}) \approx P\left(w_t \middle| w_{t-(n-1)}, \dots, w_{t-1}\right)
$$

$$
P(W) = \prod_{t=1 \dots T} P(w_t | w_1, w_2, \dots, w_{t-1}) \approx \prod_{t=1 \dots T} P\left(w_t \middle| w_{t-(n-1)}, \dots, w_{t-1}\right)
$$

- n-gram: n consecutive tokens
- n-gram language model: computes the probability of the upcoming token conditioning on the previous (n-1) tokens

Simplest Case: Unigram Model

$$P(W) = \prod_{t=1\ldots T} P(w_t|w_1, w_2, \ldots, w_{t-1}) \approx \prod_{t=1\ldots T} P\left(w_t|w_{t-(n-1)}, \ldots, w_{t-1}\right)$$

$n = 1$

$$P(W) \approx \prod_{t=1\ldots T} P(w_t)$$

W = (import, numpy, as, np)

$$P(W)$$
$$= P(import, numpy, as, np)$$
$$\approx P(import) \cdot P(numpy) \cdot P(as) \cdot P(np)$$

DEMO: generate code from unigram model

Bigram model

$$P(W) = \prod_{t=1\ldots T} P(w_t|w_1, w_2, \ldots, w_{t-1}) \approx \prod_{t=1\ldots T} P\left(w_t|w_{t-(n-1)}, \ldots, w_{t-1}\right)$$

$n = 2$

$$P(W) \approx \prod_{t=1\ldots T} P(w_t|w_{t-1})$$

- special tokens
  - <s> begin of sequence
  - </s> end of sequence

W = (import, numpy, as, np)

$$P(W)$$
$$= P(import, numpy, as, np)$$
$$\approx \cancel{P(import)}\ P(import \mid \langle s \rangle)$$
$$\cdot P(numpy \mid import)$$
$$\cdot P(as \mid numpy)$$
$$\cdot P(np \mid as)$$
$$\cdot P(\langle /s \rangle \mid np)$$

Bigram model, Maximum Likelihood Estimation

$$P(W) = \prod_{t=1\ldots T} P(w_t|w_1, w_2, \ldots, w_{t-1}) \approx \prod_{t=1\ldots T} P\left(w_t|w_{t-(n-1)}, \ldots, w_{t-1}\right)$$

$n = 2$

$$P(W) \approx \prod_{t=1\ldots T} P(w_t|w_{t-1})$$

$$P(w_t|w_{t-1}) = \frac{c(w_{t-1}, w_t)}{c(w_{t-1})}$$

how to estimate these probabilities?

W = (import, numpy, as, np)

$$P(W)$$
$$= P(import, numpy, as, np)$$
$$\approx P(import \mid <s>)$$
$$\cdot P(numpy \mid import)$$
$$\cdot P(as \mid numpy)$$
$$\cdot P(np \mid as)$$
$$\cdot P(</s> \mid np)$$

DEMO: bigram model

Practical Issues

- Log probability
  - $\log(p_1 \cdot p_2) = \log p_1 + \log p_2$
  - avoid underflow
  - adding is faster than multiplying
- Smoothing/Backoff
  - $p = 0$ for tokens unseen in the training set (out-of-vocabulary)
    - cause problem when calculating log probability … and perplexity (later)
    - cannot generalize to testing set (e.g., new code using new libraries)
  - Add-1 smoothing / Laplace smoothing:

$$P_{MLE}(w_t|w_{t-1}) = \frac{c(w_{t-1}, w_t)}{c(w_{t-1})} \longrightarrow P_{Add1}(w_t|w_{t-1}) = \frac{c(w_{t-1}, w_t) + 1}{c(w_{t-1}) + |V|}$$

DEMO: add these and scale to larger n-grams

  - Backoff: use (n-1)-gram model when n-gram count is 0 (with a scale)
  - Kneser-Ney smoothing… https://github.com/kpu/kenlm

Perplexity, Evaluating Language Models

- The best language model is one that best predicts an unseen test set (i.e., gives the highest $P(W)$)
- Perplexity:
  the inverse probability of the test set, normalized by the number of tokens
  - lower = better

$$PP(W) = P(W)^{-\frac{1}{T}} = \sqrt[T]{\frac{1}{P(w_1, w_2, \ldots, w_T)}}$$

- (Related) Cross-entropy

$$H(W) = -\sum_{t=1\ldots T} \frac{1}{T} \log P(w_t)$$
$$= \log PP(W)$$

- Intrinsic evaluation

Perplexity is a bad approximation unless the test data looks just like the training data, so generally only useful in pilot experiments.

DEMO: perplexity of n-gram models

Remarks

- In practice, use existing implementations of n-gram models
  - nltk (in Python): https://www.nltk.org/_modules/nltk/model/ngram.html
  - KenLM (in C++): https://github.com/kpu/kenlm