Language Modeling

Probabilistic Language Models

Probabilistic Language Models learn the probability of the next token, based on previous tokens. Given
that a set of words w_1,w_2, w_3……w_k predict the next word as w_{k+1}

$P(w_{k+1} \mid w_k, w_{k-1}, w_{k-2} \ldots)$ This is multi-class classification problem inspired by naive-bayes

N-gram models

An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language.

Consider “I have a dream” exists within a corpus.
P(I have a dream) = P(I) · p(have) · p(a) · p(dream)

Eg: a tri-gram model:

< start >	I	want	a	dream	job	have	dog	company
(< start >, I)	0	0	0.0714	0	0	0	0.142	0
(I,want)	0	0	0	0.0714	0	0	0	0
(want,a)	0	0	0	0	0.0714	0	0	0
(a,dream)	0	0	0	0	0	0.0714	0	0
(dream,job)	0	0	0	0	0	0	0	0
(i,have)	0	0	0	0.142	0	0	0	0
(have,a)	0	0	0	0	0.0714	0.0714	0	0
(a,dog)	0	0	0	0	0	0	0	0
(dream,company)	0	0	0	0	0	0	0	0
(company, < / start>)	0	0	0	0	0	0	0	0
(job, < / start>)	0	0	0	0	0	0	0	0
(dog, < / start>)	0	0	0	0	0	0	0	0

Building N-grams: Maximum Likelihood Estimation

Maximum likelihood estimation maximizes a likelihood function, that, under the assumed statistical model, the observed data is most probable.

For n-grams, the likelihood function for the most probable word at position k is

$$P\left(w_k | w_{k-1}, w_{k-2} \ldots w_1\right) = \frac{Count\left(w_k, w_{k-1}\right)}{Count\left(w_{k-1}, w_{k-2} \ldots w_1\right)}$$

N-gram models are trained by building a dictionary that given a set of n-1 words, predicts n^{th} word by maximum likelihood estimation. Thus, n-gram assumes that

N-gram assumption: The probability of each word is independent, given the words before it, and only depends on the fraction of their occurrence within the corpus

Evaluation

To evaluate the performance of n-gram (or) autoregressive models, the following metrics are used

- ***Average Loglikelihood:*** of a text document from a corpus is
  - $logP_{eval}(text)_{avg} = \frac{1}{|\ |word|\ |} \sum_{word} logP_{train}(word)$
  - where $|\ |word|\ |$ is number of words in the text
- ***Cross entropy****:* of a text document from a corpus is negative of average log-likelihood
  - Cross Entropy (text) = $-\ logP_{eval}(text)_{avg}$
- ***Perplexity***: of a text document from a corpus is defined as exponential of cross-entropy
  - $Perplexity(text) = e^{cross\_entropy(text)}$
  - A lower perplexity indicates a higher model's understanding of a language

Accounting OOV (Out of vocabulary words)

To account for OOV words, the following algorithms are used

- Laplace smoothening helps to tackle problem of zero probability. We add an additional `[UNK]` token to training set and modify likelihood function
    - **K-Smoothening**:

$$P\left(w_k | w_{k-1}, w_{k-2} ... w_1\right) = \frac{Count\left(w_k, w_{k-1}\right) + k}{Count\left(w_{k-1}, w_{k-2} ... w_1\right) + k \cdot V}$$

        - where k is a randomly chosen hyperparameter and $V$is vocabulary size (Number of unique n-grams in the training corpus)
    - **Add 1 Smoothening**:

$$P\left(w_k | w_{k-1}, w_{k-2} ... w_1\right) = \frac{Count\left(w_k, w_{k-1}\right) + 1}{Count\left(w_{k-1}, w_{k-2} ... w_1\right) + V}$$

        - where $V$is vocabulary size (Number of unique n-grams in the training corpus). Add 1 smoothening is a simplified version of K-Smoothening where $K = 1$
    - Effect of Smoothening

$$P_{train}([UNK]) = \frac{k}{V}$$

- Back-off and Interpolation
    - **Back-off**: this technique falls back to lower-order N-gram models when a higher-order N-gram model is unavailable

$$P_{backoff}\left(w_n | w_{n-1}, w_{n-2}, ... w_1\right) = \{P_{train}\left(w_n | w_{n-1}, w_{n-2}, ... w_1\right) \; Count\left(w_n, w_{n-1}, ... w_1\right) \geq 0 \; P_{backoff}\left(w_{n-1} | w_{n-2}, w_{n-3}, ... w_1\right)$$

- **Interpolation**: This technique combines the likelihoods of multiple N-gram models to estimate the next word. In linear interpolation, we typically take a weighted sum of likelihoods as a predictor for the next token

$$P\left(w_n | w_{n-1}, w_{n-2}, ... w_1\right) = \lambda_1 P\left(w_1\right) + \lambda_2 P\left(w_2 | w_1\right) + ... \; \lambda_{n-1} P\left(w_{n-1} | w_{n-2}, w_{n-3} ... w_1\right) + \lambda_n P\left(w_n | w_{n-1}, w_{n-2}, ... w_1\right)$$
