Faculty of Computer Science, Dalhousie University *30-Oct-2025*

CSCI 4152/6509 — Natural Language Processing

Lecture 11: N-gram Model and Markov Chain Model

Location: Studley LSC-Psychology P5260 Instructor: Vlado Keselj
Time: 14:35 – 15:55

Previous Lectures

- Joint Distribution and Fully Independent Model review
- Classification example:
  - Joint Distribution Model
  - Fully Independent Model
  - Naïve Bayes Model
- Naïve Bayes classification model
  - Assumption, definition, graphical representation
  - Number of parameters
  - Pros and cons, additional notes
  - Bernoulli and Multinomial Naïve Bayes

---

13 N-gram Model and Markov Chain Model

The main motivation for exploring the N-gram Model as a probabilistic model is an NLP task known as *Language Modeling. Language Modeling* can be defined as the task of buidling a probabilistic model that can estimate the probably of an arbitrary sentence in context of a natural language; or in other words estimating probabilty $\mathrm{P}(\textit{sentence})$. A *Language Model* is a model that can be used to make this estimate.

As an alternative and equivalent definition, a Language Model is a model that given a string of tokens in a language, such as words or characters, it can predict the most likely token that can follow this sequence.

One application of this problem is in speech recognition. In speech recognition, we are interested in

$$\arg\max_{\text{sentence}} \mathrm{P}(\text{sentence}|\text{sound})$$

This is equal to:

$$\begin{aligned}\arg\max_{\text{sentence}} \mathrm{P}(\text{sentence}|\text{sound}) &= \arg\max_{\text{sentence}} \frac{\mathrm{P}(\text{sentence},\text{sound})}{\mathrm{P}(\text{sound})} \\ &= \arg\max_{\text{sentence}} \mathrm{P}(\text{sentence},\text{sound}) \\ &= \arg\max_{\text{sentence}} \mathrm{P}(\text{sound}|\text{sentence})\mathrm{P}(\text{sentence})\end{aligned}$$

It is easier to estimate $\mathrm{P}(\text{sound}|\text{sentence})$ than $\mathrm{P}(\text{sentence},\text{sound})$, and it is done by an *acoustic model,* while $\mathrm{P}(\text{sentence})$ is estimated by a *language model.*

N-gram model is very simple and it is among the most successful models for language modelling; trigram ($n = 3$) word model in particular. In an n-gram model, we calculate joint distribution for all n-tuples of consecutive words

(or characters). For example, in the trigram model, we count the number of occurrences of each triple of consecutive words from a corpus. Using this statistics, we can estimate the probability of arbitrary word w_3 following two given words w_1 and w_2: $\mathrm{P}(w_3|w_1w_2)$. It is useful to assign some small probability to unseen triples as well (using a technique called *smoothing*). If we use two "dummy" words '$\cdot$' at the beginning of each sentence, then the probability of arbitrary sentence can be calculated as:

$$\mathrm{P}(w_1w_2\ldots w_k) = \mathrm{P}(w_1|\cdot\cdot)\mathrm{P}(w_2|w_1\cdot)\mathrm{P}(w_3|w_2w_1)\ldots\mathrm{P}(w_k|w_{k-1}w_{k-2})$$

Slide notes:

N-gram Language Model

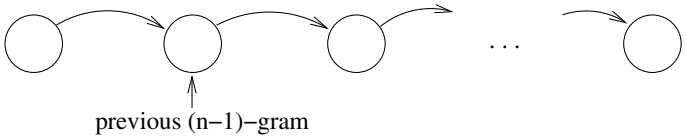
- Predict next word using $(n-1)$ previous words
- Example assumption with $n = 3$:

$$\begin{aligned}\mathrm{P}(w_1w_2\ldots w_k) = \\ \mathrm{P}(w_1|\cdot\cdot)\mathrm{P}(w_2|w_1\cdot)\mathrm{P}(w_3|w_2w_1)\ldots \\ \mathrm{P}(w_k|w_{k-1}w_{k-2})\end{aligned}$$

N-gram Model: Notes

- Reading: Chapter 4 of [JM]
- Use of log probabilities
  - similarly as in the Naïve Bayes model for text
- Graphical representation

Graphical Representation



Use of log probabilities

Multiplying a large number of probability values, which are typically small, gives a very small result (close to zero), so in order to avoid floating-point underflow, we should use addition of logarithms of the probabilities in the model.

Slide notes:

N-gram Model as a Markov Chain

- N-gram Model is very similar to Markov Chain Model
- Markov Chain consists of
  - sequence of variables V_1, V_2, ...
  - probability of V_1 is independent
  - each next variable is dependent only on the previous variable: V_2 on V_1, V_3 on V_2, etc.
  - Conditional Probability Tables: $P(V_1)$, $P(V_2|V_1)$, ...
- Markov Chain is identical to bi-gram model, but higher-order n-gram models are very similar as well

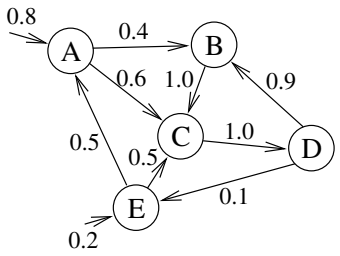
Markov Chain: Formal Definition

A *stochastic process* in general is a family of random variables $\{V_i\}$, where i is an index from a set I. A stochastic process is also denoted as $\{V_i, i \in I\}$, or $\{V_t, t \in T\}$, with intuition coming from time index. The index set I can

be an arbitrary ordered set, but we will usually assume they it is either finite or countably infinite (i.e., enumerable), and then process can be denoted as $\{V_i\}_{i=1}^{\infty}$. A process is called a *Markov process* if given the value of V_t, for some index t, the values of V_s, where $s > t$, do not depend on values of V_u, where $u < t$. In case of a finite or countably infinite index set, this means that the value of V_i depends only on the value of the previous variable V_{i-1}. In this case, the Markov process is called a *Markov Chain.*

A Markov Chain can be described similarly to a Deterministic Finite automaton (DFA), but instead of reading input, we assume that we start in a random state based on a probability distribution, and change states in sequence based on a probability distribution of the next state given the previous state.

For example, a Markov chain could be illustrated in the following way.



This model could generate the sequence $\{A, C, D, B, C\}$ of length 5 with probability:

$$0.8 \cdot 0.6 \cdot 1.0 \cdot 0.9 \cdot 1.0 = 0.432$$

assuming that we are modelling sequences of this length.

If we want to model sequences of arbitrary length, we would also need a stopping probability.

Evaluating Language Models: Perplexity

- Evaluation of language model: extrinsic and intrinsic
- Extrinsic: model embedded in application
- Intrinsic: direct evaluation using a measure
  In extrinsic evaluation, the language model is embedded in a wider application, and the performance of the model is measured through the performance of the application. For example, we can evaluate performance of a language model by measuring improvement in a speech recognition application in which it is embedded.
  In intrinsic evaluation, we directly evaluate the language model using some measure, such as perplexity.
- Perplexity, W — text, $L = |W|$,

$$\text{PP}(W) = \sqrt[L]{\frac{1}{P(W)}} = \sqrt[L]{\prod_i \frac{1}{P(w_i|w_{i-n+1} \dots w_{i-1})}}$$

- Weighted average branching factor

Perplexity

Use of Language Modeling in Classification

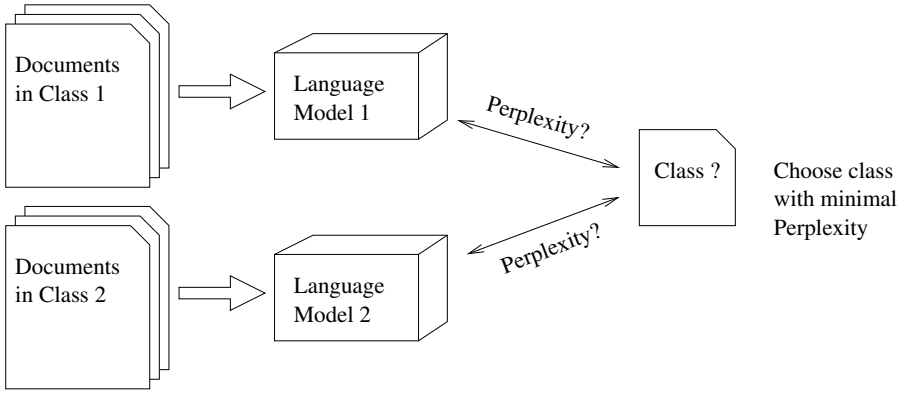
- Perplexity, W — text, $L = |W|$,

$$\text{PP}(W) = \sqrt[L]{\frac{1}{P(W)}} = \sqrt[L]{\prod_i \frac{1}{P(w_i|w_{i-n+1} \dots w_{i-1})}}$$

– Text classification using language models

The perplexity measure tells us how well a language model predicts an existing text. It is also called a weighted branching factor. For example, if we generate a text using words from a 100,000-word vocabulary, and we have not better way to predict words than randomly choosing them using a uniform distribution, then the perplexity measure for any text will be about 100,000. A lower perplexity measure means that the model is "predicting" a text better. For example, a perplexity of 4 means that the model predicts the next word with odds of 1 to 4, which would be quite good.

We can do text classification using language models and perplexity in the following way: We build language models for different classes by training them using training data for each class. Then, we measure perplexity of each model on a test document, and we choose the class whose model has lowest perplexity, as shown in the following figure:



Slide notes:

Unigram Model and Multinomial Naïve Bayes

– It is interesting that classification using Unigram Language Model is same as Multinomial Naïve Bayes with all words

13.1 N-gram Model Smoothing

Smoothing is any technique in probabilistic modeling used to avoid zero probabilities in a model trained from training data. Namely, due to the sparse data problem, we may easily have one of the probabilities estimated to zero by the MLE process. Since these probabilities are typically used as factors in products, this easily leads to a zero probability being assigned to a full configuration that happen not to be seen in the training data. To avoid this situation, we use smoothing techniques. Generally speaking, the smoothing techniques take some probability from seen and predictable events and assign it to the unseen events. There are several well-known smoothing techniques used in the n-grams model: add-one smoothing (or Laplace smoothing), Witten-Bell smoothing, Good-Turing smoothing, Kneser-Ney smoothing (described in the new edition of the Jurafsky and Martin textbook), etc. We will now take a closer look at the Laplace (add-one) and the Witten-Bell smoothing. These techniques can be generalized to other models as well.

Example: Character Unigram Probabilities

– Training example: `mississippi`
– What are letter unigram probabilities?
– What would be probability of the word '`river`' based on this model?

The letter unigram probabilities without smoothing:

Letter	Counts	Estimated frequency
i	4	4/11 ≈ 0.363636363636364
m	1	1/11 ≈ 0.0909090909090909
p	2	2/11 ≈ 0.181818181818182
s	4	4/11 ≈ 0.363636363636364
other letters	0	0
Total:	11	1.0

The probability of the word '`river`' would be 0 in this model, since it contains letters with the probability 0, so the product of those letter probabilities would be 0:

$$\mathrm{P}(\text{`}\texttt{river}\text{'}) = \mathrm{P}(\texttt{r})\mathrm{P}(\texttt{i})\mathrm{P}(\texttt{v})\mathrm{P}(\texttt{e})\mathrm{P}(\texttt{r}) = 0 \cdot \frac{4}{11} \cdot 0 \cdot 0 \cdot 0 = 0.0$$

13.1.1 Add-one Smoothing (Laplace Smoothing)

The *add-one smoothing* is also known the *Laplace smoothing*. We start with the count of 1 for all events, and thus prevent any events to end up with the probability 0. In a unigram example, it would mean that all tokens $w \in V$, where V is the vocabulary, start with count 1. If $|V|$ is the vocabulary size, and n is the number of tokens in a text, the smoothed unigram probabilities end up to be

$$P(w) = \frac{\#(w) + 1}{n + |V|}$$

where $\#(w)$ denotes the number of occurrences of the token w in the training text.

Similarly, for bigram smoothing for example, the estimated probability would be:

$$P(a|b) = \frac{\#(ba) + 1}{\#(b) + |V|}$$

If the vocabulary size is very large compared to $\#(b)$, which happens with words, for example, or if b is relatively rare, then this kind of smoothing takes too much of the probability distribution from the seen events and assigns it to the unseen events.

Mississippi Example: Add-one Smoothing

- Let us again consider the example trained on the word: `mississippi`
- What are letter unigram probabilities with add-one smoothing?
- What is the probability of: `river`

With the Add-one smoothing, we would start with count 1 for each letter in the vocabulary. If we assume that our vocabulary consists of all lower-case letters in the English alphabet, the total alphabet size would be 26. Since each letter count would start with 1, with 11 letters in the word 'mississippi', we would have a total count of 37. This leads to the following table of smoothed probabilities:

Letter	Modified counts	Estimated frequency
i	5	5/37 ≈ 0.135135135135135
m	2	2/37 ≈ 0.0540540540540541
p	3	3/37 ≈ 0.0810810810810811
s	5	5/37 ≈ 0.135135135135135
other letters(×22)	1(×22)	1/37 ≈ 0.027027027027027 (×22)
Total:	37	1.0

The probability of the word '`river`' in this model would be:

$$\mathrm{P}(\text{`}\texttt{river}\text{'}) = \mathrm{P}(\texttt{r})\mathrm{P}(\texttt{i})\mathrm{P}(\texttt{v})\mathrm{P}(\texttt{e})\mathrm{P}(\texttt{r}) = \frac{1}{37} \cdot \frac{5}{37} \cdot \frac{1}{37} \cdot \frac{1}{37} \cdot \frac{1}{37} \approx 7.21043363591149 \cdot 10^{-8}$$