Faculty of Computer Science, Dalhousie University *19-Oct-2023*

CSCI 4152/6509 — Natural Language Processing

Lecture 14: N-gram Model and Smoothing

Location: Rowe 1011 Instructor: Vlado Keselj
Time: 16:05 – 17:25

Previous Lecture

- Naïve Bayes classification model (continued)
  - Spam detection example
  - Computational tasks
  - Number of parameters
  - pros and cons, additional notes
  - Bernoulli and Multinomial Naïve Bayes
- N-gram model
  - Language modeling
  - N-gram model assumption

Slide notes:

N-gram Model as a Markov Chain

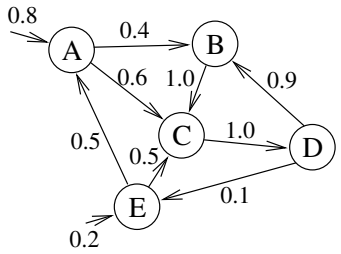
- N-gram Model is very similar to Markov Chain Model
- Markov Chain consists of
  - sequence of variables V_1, V_2, ...
  - probability of V_1 is independent
  - each next variable is dependent only on the previous variable: V_2 on V_1, V_3 on V_2, etc.
  - Conditional Probability Tables: $P(V_1)$, $P(V_2|V_1)$, ...
- 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, $L = |W|$,

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

- Weighted average branching factor

Perplexity

Use of Language Modeling in Classification

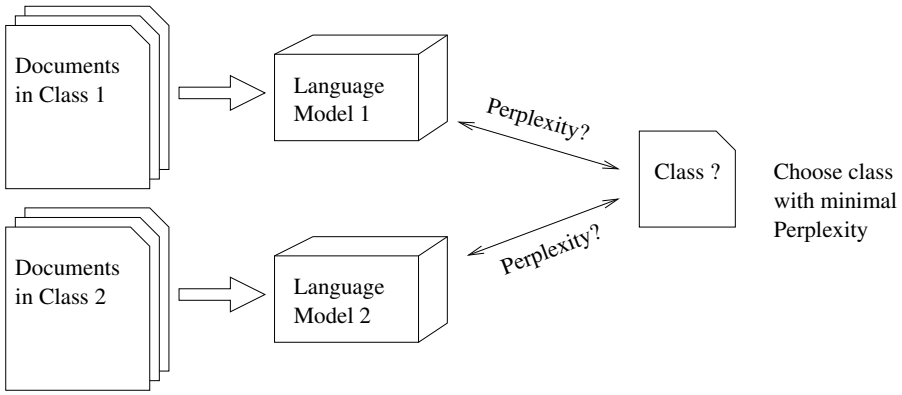
- Perplexity, W — text, $L = |W|$,

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

- 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 $\approx$ 0.363636363636364
m	1	1/11 $\approx$ 0.0909090909090909
p	2	2/11 $\approx$ 0.181818181818182
s	4	4/11 $\approx$ 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:

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

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 $\approx$ 0.135135135135135
m	2	2/37 $\approx$ 0.0540540540540541
p	3	3/37 $\approx$ 0.0810810810810811
s	5	5/37 $\approx$ 0.135135135135135
other letters($\times$22)	1($\times$22)	1/37 $\approx$ 0.027027027027027 ($\times$22)
Total:	37	1.0

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

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

13.1.2 Witten-Bell Discounting

In the context of data compression, Witten and Bell (1991) analyzed several smoothing methods, under the title "The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression". They considered three methods A, B, and C, and then, based on a Poisson process modelling, the methods P, X, and XC. It is interesting to note that the method X uses the same, or very similar, idea as in the Good-Turing smoothing.

The method C is what is usually referred to as the Witten-Bell smoothing. It uses an intuitive idea from data compression. Let us assume that we use a data compression method, which uses a dictionary of tokens w_1, w_2, ..., w_r, so far. As long as the new tokens are from this set, we can encode them in some way, but whenever a new token

appears, we need a special 'escape' code to introduce this token to the vocabulary. In this way, we can think of new tokens appearing as a new event, beside the events of seeing existing tokens. This is supported in practice by seeing approximately constant rate of new words appearing in a text after some initial reading. We can use the frequency of such 'escape' code as an estimate of the probability of seeing previously unseen events, and make sure that we allocate that much probability distribution mass for the smoothing purposes.

Example: Let us consider again using the example of training data '`mississippi`' to train a unigram model, and then use it to estimate probability of the string '`river`'.

We will consider again estimating probability of 26 lower-case letters from the word '`mississippi`'. As in the case of unsmoothed n-grams, we will count letters: '`m`' 1 time, '`i`' 4 times, '`s`' 4 times, and '`p`' 2 times. However, we also note that we saw 4 different letters, which is equivalent to seeing 'escape' character 4 times, so we will reserve count 4 for unseen events in future as well. This is how we get the following probabilities using the Witten-Bell discounting:

Letter	Modified counts	Estimated frequency
i	4	4/15 $\approx$ 0.266666666666667
m	1	1/15 $\approx$ 0.0666666666666667
p	2	2/15 $\approx$ 0.133333333333333
s	4	4/15 $\approx$ 0.266666666666667
new letters total	4	4/15 $\approx$ 0.266666666666667
Total:	15	

When we split the probability reserved for new letters equally among the remaining $26 - 4 = 22$ letters, we obtain the final estimated frequency:

Letter	Estimated frequency
i	4/15 $\approx$ 0.266666666666667
m	1/15 $\approx$ 0.0666666666666667
p	2/15 $\approx$ 0.133333333333333
s	4/15 $\approx$ 0.266666666666667
other letters	$\frac{4}{15 \cdot 22} = 2/165 \approx 0.0121212121212121$

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

$$\mathrm{P}(\text{`river'}) = \mathrm{P}(\mathtt{r})\mathrm{P}(\mathtt{i})\mathrm{P}(\mathtt{v})\mathrm{P}(\mathtt{e})\mathrm{P}(\mathtt{r}) = \frac{2}{165} \cdot \frac{4}{15} \cdot \frac{2}{165} \cdot \frac{2}{165} \cdot \frac{2}{165} \approx 5.75642615879697 \cdot 10^{-9}$$

Formulae for Witten-Bell Discounting If we want to express this in terms of formulae, we will denote that we saw r distinct tokens in a text of length n, or we can say that we saw n events and r 'escape' events, so the probability of seeing new tokens is $\frac{r}{n+r}$. Hence the unigram probability for seen tokens:

$$P(w) = \frac{\#(w)}{n+r}$$

and the total probability for unseen tokens is:

$$\frac{r}{n+r}$$

It is convenient that in the previous formulae we did not need to know the vocabulary size. If we do know the vocabulary size, we can now divide the probability for unseen tokens equally, and obtain:

$$P(w) = \frac{r}{(n+r)(|V| - r)}$$

for unseen tokens w.

Bigrams and Higher-order n-grams

The probabilities for bigrams and higher-order n-grams are smoothed in a similar way:

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

for seen bigrams ba, where r_b is the number of different tokens following b. The number $\#(b)$ does not represent necessarily the exact number of occurrences of b in this case. More precisely, it is the number of occurrences of b except at the end of text; i.e., the number of occurrences of b where b is followed by another token. The remaining probability mass for unseen events:

$$\frac{r_b}{\#(b) + r_b}$$

is used for unseen bigrams that start with b, and is usually divided according to lower-order n-grams; which would be unigrams in this case. If N_b is the set of all tokens that never follow b in the training data, then:

$$P(a|b) = \frac{r_b}{\#(b) + r_b} \cdot P(a)/\Sigma_{x \in N_b} P(x)$$

for unseen bigrams ba.

The next model: HMM. Our next probabilistic model is the Hidden Markov Model (HMM), and it is applicable to the task of labelling tokens of a sequence, such as the task of Part-of-Speech tagging (POS Tagging). Before that, we will make a review of the parts of speech in English, which are quite applicable with some changes to other natural languages as well.

Slide notes:

The Next Model: HMM

- HMM — Hidden Markov Model
- Typically used to annotate sequences of tokens
- Most common annotation: Part-of-Speech Tags (POS Tags)
- First, we will make a review of parts of speech in English