Natural Language Processing

Lecture V. N-gram Language Models and TF-IDF

Forrest Sheng Bao, Ph.D.

Dept. of Computer Science
Iowa State University
Ames, IA 50011

Sept. 14, 2021

Outline

Why language models

For many applications, the output is a sequence of words.

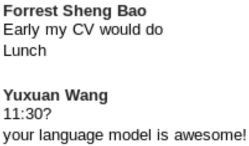
- Example 1: Machine translation (MT):

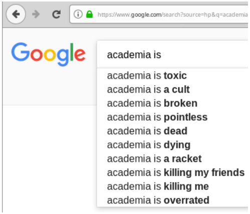
  Sprechen Sie MATLAB? $\Rightarrow$ Do you speak MATLAB?

- Example 2: Automatic Speech Recognition (ASR):

 $\Rightarrow$ "\$2M seed from Sequoia?"

- Example 3: Spell checking or spell suggestion





- The system needs to evaluate/rank a set of candidates, e.g., "\$2M seed from Sequoia?" over "\$2M seed from DFJ!"
- An easy way is to compute the likelihood that a candidate is a "making-sense" sentence.

Why language models

For many applications, the output is a sequence of words.

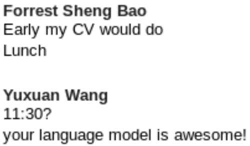
- Example 1: Machine translation (MT):

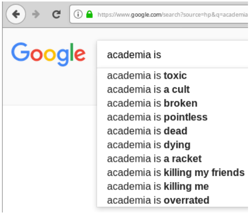
  Sprechen Sie MATLAB? $\Rightarrow$ Do you speak MATLAB?

- Example 2: Automatic Speech Recognition (ASR):

$\Rightarrow$ “\$2M seed from Sequoia?”

- Example 3: Spell checking or spell suggestion





- The system needs to evaluate/rank a set of candidates, e.g., “\$2M seed from Sequoia?” over “\$2M seed from DFJ!”
- An easy way is to compute the likelihood that a candidate is a “making-sense” sentence.

Why language models

For many applications, the output is a sequence of words.

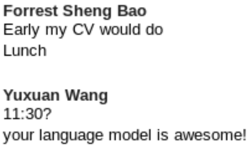
- Example 1: Machine translation (MT):

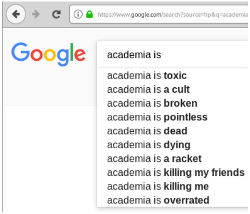
  Sprechen Sie MATLAB? $\Rightarrow$ Do you speak MATLAB?

- Example 2: Automatic Speech Recognition (ASR):

   $\Rightarrow$ "$2M seed from Sequoia?"

- Example 3: Spell checking or spell suggestion





- The system needs to evaluate/rank a set of candidates, e.g., "$2M seed from Sequoia?" over "$2M seed from DFJ!"
- An easy way is to compute the likelihood that a candidate is a "making-sense" sentence.

Why language models

For many applications, the output is a sequence of words.

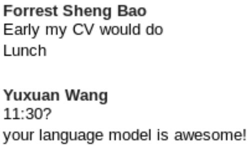
- Example 1: Machine translation (MT):

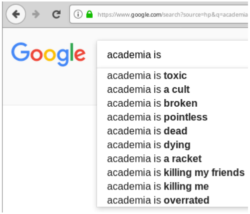
  Sprechen Sie MATLAB? $\Rightarrow$ Do you speak MATLAB?

- Example 2: Automatic Speech Recognition (ASR):

$\Rightarrow$ “$2M seed from Sequoia?”

- Example 3: Spell checking or spell suggestion





- The system needs to evaluate/rank a set of candidates, e.g., “$2M seed from Sequoia?” over “$2M seed from DFJ!”
- An easy way is to compute the likelihood that a candidate is a “making-sense” sentence.

Why language models

For many applications, the output is a sequence of words.

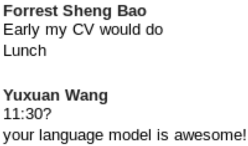
- Example 1: Machine translation (MT):

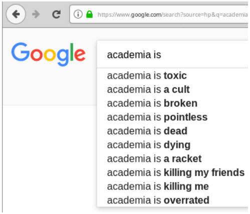
  Sprechen Sie MATLAB? $\Rightarrow$ Do you speak MATLAB?

- Example 2: Automatic Speech Recognition (ASR):

   $\Rightarrow$ “$2M seed from Sequoia?”

- Example 3: Spell checking or spell suggestion





- The system needs to evaluate/rank a set of candidates, e.g., “$2M seed from Sequoia?” over “$2M seed from DFJ!”
- An easy way is to compute the likelihood that a candidate is a “making-sense” sentence.

Ranking candidate text strings

- For example, we can compute a probability for each string below:
- S1: “Please CALL me in 10 minutes.”
- S2: “Please ALL me in 10 minutes.”
- Which one is bigger? $P(S_1)$ or $P(S_2)$?

Language models

- A statistical language model is a probability distribution over sequences of words.
- Given a length l, a language model assigns a probability $P(w_1, \ldots, w_l)$ to the sequence.
- When $l = 1$, we have the probability for each individual word.
- When $l = 2$, we have the probability for each two-word pair.
- ...
- In the example above, the system will compare the probabilities $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_3 = \text{“Sequoia”})$ and $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“DFJ”})$.
- By chain rule, we have

$$
\begin{aligned}
&P(w_0 = \rhd, w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“Sequoia”}, w_5 = \lhd) \\
=&P(w_1 = \text{“\$2M”} | w_0 = \rhd) \\
\times&P(w_2 = \text{“seed”} | w_1 = \text{“\$2M”}, w_0 = \rhd) \\
\times&P(w_3 = \text{“from”} | w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \rhd) \\
\times&P(w_4 = \text{“Sequoia”} | w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \rhd) \\
\times&P(w_5 = \lhd | w_4 = \text{“Sequoia”}, w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \rhd)
\end{aligned}
$$

Language models

- A statistical language model is a probability distribution over sequences of words.
- Given a length l, a language model assigns a probability $P(w_1, \ldots, w_l)$ to the sequence.
- When $l = 1$, we have the probability for each individual word.
- When $l = 2$, we have the probability for each two-word pair.
- ...
- In the example above, the system will compare the probabilities $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_3 = \text{“Sequoia”})$ and $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“DFJ”})$.
- By chain rule, we have

$$
\begin{aligned}
&P(w_0 = \triangleright, w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“Sequoia”}, w_5 = \triangleleft) \\
=&P(w_1 = \text{“\$2M”} | w_0 = \triangleright) \\
\times&P(w_2 = \text{“seed”} | w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_3 = \text{“from”} | w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_4 = \text{“Sequoia”} | w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_5 = \triangleleft | w_4 = \text{“Sequoia”}, w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)
\end{aligned}
$$

Language models

- A statistical language model is a probability distribution over sequences of words.
- Given a length l, a language model assigns a probability $P(w_1, \ldots, w_l)$ to the sequence.
- When $l = 1$, we have the probability for each individual word.
- When $l = 2$, we have the probability for each two-word pair.
- ...
- In the example above, the system will compare the probabilities $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_3 = \text{“Sequoia”})$ and $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“DFJ”})$.
- By chain rule, we have

$$
\begin{aligned}
&P(w_0 = \triangleright, w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“Sequoia”}, w_5 = \triangleleft) \\
=&P(w_1 = \text{“\$2M”} | w_0 = \triangleright) \\
\times&P(w_2 = \text{“seed”} | w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_3 = \text{“from”} | w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_4 = \text{“Sequoia”} | w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_5 = \triangleleft | w_4 = \text{“Sequoia”}, w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)
\end{aligned}
$$

Language models

- A statistical language model is a probability distribution over sequences of words.
- Given a length l, a language model assigns a probability $P(w_1, \ldots, w_l)$ to the sequence.
- When $l = 1$, we have the probability for each individual word.
- When $l = 2$, we have the probability for each two-word pair.
- ...
- In the example above, the system will compare the probabilities $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_3 = \text{“Sequoia”})$ and $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“DFJ”})$.
- By chain rule, we have

$$
\begin{aligned}
&P(w_0 = \triangleright, w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“Sequoia”}, w_5 = \triangleleft) \\
=&P(w_1 = \text{“\$2M”} | w_0 = \triangleright) \\
\times&P(w_2 = \text{“seed”} | w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_3 = \text{“from”} | w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_4 = \text{“Sequoia”} | w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_5 = \triangleleft | w_4 = \text{“Sequoia”}, w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)
\end{aligned}
$$

Language models

- A statistical language model is a probability distribution over sequences of words.
- Given a length l, a language model assigns a probability $P(w_1, \ldots, w_l)$ to the sequence.
- When $l = 1$, we have the probability for each individual word.
- When $l = 2$, we have the probability for each two-word pair.
- ...
- In the example above, the system will compare the probabilities $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_3 = \text{“Sequoia”})$ and $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“DFJ”})$.
- By chain rule, we have

$$
\begin{aligned}
& P(w_0 = \triangleright, w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“Sequoia”}, w_5 = \triangleleft) \\
= & P(w_1 = \text{“\$2M”} | w_0 = \triangleright) \\
\times & P(w_2 = \text{“seed”} | w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times & P(w_3 = \text{“from”} | w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times & P(w_4 = \text{“Sequoia”} | w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times & P(w_5 = \triangleleft | w_4 = \text{“Sequoia”}, w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)
\end{aligned}
$$

Language models

- A statistical language model is a probability distribution over sequences of words.
- Given a length l, a language model assigns a probability $P(w_1, \ldots, w_l)$ to the sequence.
- When $l = 1$, we have the probability for each individual word.
- When $l = 2$, we have the probability for each two-word pair.
- ...
- In the example above, the system will compare the probabilities $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_3 = \text{“Sequoia”})$ and $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“DFJ”})$.
- By chain rule, we have

$$
\begin{aligned}
&P(w_0 = \triangleright, w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“Sequoia”}, w_5 = \triangleleft)\\
=&P(w_1 = \text{“\$2M”} | w_0 = \triangleright)\\
\times&P(w_2 = \text{“seed”} | w_1 = \text{“\$2M”}, w_0 = \triangleright)\\
\times&P(w_3 = \text{“from”} | w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)\\
\times&P(w_4 = \text{“Sequoia”} | w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)\\
\times&P(w_5 = \triangleleft | w_4 = \text{“Sequoia”}, w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)
\end{aligned}
$$

Language models

- A statistical language model is a probability distribution over sequences of words.
- Given a length l, a language model assigns a probability $P(w_1, \ldots, w_l)$ to the sequence.
- When $l = 1$, we have the probability for each individual word.
- When $l = 2$, we have the probability for each two-word pair.
- ...
- In the example above, the system will compare the probabilities $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_3 = \text{“Sequoia”})$ and $P(w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“DFJ”})$.
- By chain rule, we have

$$
\begin{aligned}
&P(w_0 = \triangleright, w_1 = \text{“\$2M”}, w_2 = \text{“seed”}, w_3 = \text{“from”}, w_4 = \text{“Sequoia”}, w_5 = \triangleleft) \\
=&P(w_1 = \text{“\$2M”} | w_0 = \triangleright) \\
\times&P(w_2 = \text{“seed”} | w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_3 = \text{“from”} | w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_4 = \text{“Sequoia”} | w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright) \\
\times&P(w_5 = \triangleleft | w_4 = \text{“Sequoia”}, w_3 = \text{“from”}, w_2 = \text{“seed”}, w_1 = \text{“\$2M”}, w_0 = \triangleright)
\end{aligned}
$$

n-gram models

- Let's generalize: $P(w_1, \cdots, w_l) = \prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1})$
- Use a shorter history (n-th order Markov property):
  $\prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{l} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$
- Conditional probability from counting:
  $P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$
- When $n = 1$, 2 or 3, it's unigram, bi-gram or tri-gram respectively.
- Yes, unigram means no history but just the word itself. No order between words is considered.
- N-gram is one (simple but widely used) way to represent language models.

n-gram models

- Let's generalize: $P(w_1, \cdots, w_l) = \prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1})$
- Use a shorter history (n-th order Markov property): $\prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{l} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$
- Conditional probability from counting: $P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$
- When $n = 1$, 2 or 3, it's unigram, bi-gram or tri-gram respectively.
- Yes, unigram means no history but just the word itself. No order between words is considered.
- N-gram is one (simple but widely used) way to represent language models.

n-gram models

- Let's generalize: $P(w_1, \cdots, w_l) = \prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1})$
- Use a shorter history (n-th order Markov property):
  $\prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{l} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$
- Conditional probability from counting:
  $P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$
- When $n = 1$, 2 or 3, it's unigram, bi-gram or tri-gram respectively.
- Yes, unigram means no history but just the word itself. No order between words is considered.
- N-gram is one (simple but widely used) way to represent language models.

n-gram models

- Let's generalize: $P(w_1, \cdots, w_l) = \prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1})$
- Use a shorter history (n-th order Markov property):
$\prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{l} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$
- Conditional probability from counting:
$P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$
- When $n = 1$, 2 or 3, it's unigram, bi-gram or tri-gram respectively.
- Yes, unigram means no history but just the word itself. No order between words is considered.
- N-gram is one (simple but widely used) way to represent language models.

n-gram models

- Let's generalize: $P(w_1, \cdots, w_l) = \prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1})$
- Use a shorter history (n-th order Markov property):
  $\prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{l} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$
- Conditional probability from counting:
  $P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$
- When $n = 1$, 2 or 3, it's unigram, bi-gram or tri-gram respectively.
- Yes, unigram means no history but just the word itself. No order between words is considered.
- N-gram is one (simple but widely used) way to represent language models.

n-gram models

- Let's generalize: $P(w_1, \cdots, w_l) = \prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1})$
- Use a shorter history (n-th order Markov property):
  $\prod_{i=1}^{l} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{l} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$
- Conditional probability from counting:
  $P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$
- When $n = 1$, 2 or 3, it's unigram, bi-gram or tri-gram respectively.
- Yes, unigram means no history but just the word itself. No order between words is considered.
- N-gram is one (simple but widely used) way to represent language models.

Bag of words

The bag of words and the background words as seen in Gates building, CMU

Bag of words (cont.)

- A bag-of-words (BOW) model is basically a uni-gram model: an orderless representation of a document.
- A common use of a BOW model is create feature vectors using the frequencies of words, i.e., term frequency.
- Background words are those of extreme high frequency. They are earlier called stop words.
- When many researchers use the term “bigram” or “trigram”, they are actually talking about bag-of-double-words or bag-of-triple-words.
- See `https://en.wikipedia.org/wiki/Bag-of-words_model`.
- Also, how to build a spam filter using BOW: `https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering`

Bag of words (cont.)

- A bag-of-words (BOW) model is basically a uni-gram model: an orderless representation of a document.
- A common use of a BOW model is create feature vectors using the frequencies of words, i.e., term frequency.
- Background words are those of extreme high frequency. They are earlier called stop words.
- When many researchers use the term “bigram” or “trigram”, they are actually talking about bag-of-double-words or bag-of-triple-words.
- See `https://en.wikipedia.org/wiki/Bag-of-words_model`.
- Also, how to build a spam filter using BOW: `https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering`

Bag of words (cont.)

- A bag-of-words (BOW) model is basically a uni-gram model: an orderless representation of a document.
- A common use of a BOW model is create feature vectors using the frequencies of words, i.e., term frequency.
- Background words are those of extreme high frequency. They are earlier called stop words.
- When many researchers use the term “bigram” or “trigram”, they are actually talking about bag-of-double-words or bag-of-triple-words.
- See `https://en.wikipedia.org/wiki/Bag-of-words_model`.
- Also, how to build a spam filter using BOW: `https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering`

Bag of words (cont.)

- A bag-of-words (BOW) model is basically a uni-gram model: an orderless representation of a document.
- A common use of a BOW model is create feature vectors using the frequencies of words, i.e., term frequency.
- Background words are those of extreme high frequency. They are earlier called stop words.
- When many researchers use the term "bigram" or "trigram", they are actually talking about bag-of-double-words or bag-of-triple-words.
- See `https://en.wikipedia.org/wiki/Bag-of-words_model`.
- Also, how to build a spam filter using BOW: `https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering`

Bag of words (cont.)

- A bag-of-words (BOW) model is basically a uni-gram model: an orderless representation of a document.
- A common use of a BOW model is create feature vectors using the frequencies of words, i.e., term frequency.
- Background words are those of extreme high frequency. They are earlier called stop words.
- When many researchers use the term “bigram” or “trigram”, they are actually talking about bag-of-double-words or bag-of-triple-words.
- See https://en.wikipedia.org/wiki/Bag-of-words_model.
- Also, how to build a spam filter using BOW: https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering

Bag of words (cont.)

- A bag-of-words (BOW) model is basically a uni-gram model: an orderless representation of a document.
- A common use of a BOW model is create feature vectors using the frequencies of words, i.e., term frequency.
- Background words are those of extreme high frequency. They are earlier called stop words.
- When many researchers use the term “bigram” or “trigram”, they are actually talking about bag-of-double-words or bag-of-triple-words.
- See https://en.wikipedia.org/wiki/Bag-of-words_model.
- Also, how to build a spam filter using BOW: https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering

skip-grams and CBOW

- We may also add a gap between words.
- 1-degree Skip-grams from the sentence “I am a PhD student”: (“I”, “a”), (“am”, “PhD”), (“a”, “student”).
- A language model using skip-gram:

$$P(w_i | w_{i-k}, \ldots, w_{i-1}, \quad , w_{i+1}, \ldots, w_{i+k})$$

- NOT THE CASE!
- Instead of estimating the probability giving the neighboring words, a skip-gram language model actually estimates the probabilities of neighboring words given a word:
-

$$\sum_{j \in [-k..-1] \cup [1..k]} \log P(w_{i+j} | w_i)$$

- The first equation is actually called Continues BOW (CBOW) model.

skip-grams and CBOW

- We may also add a gap between words.
- 1-degree Skip-grams from the sentence "I am a PhD student": ("I", "a"), ("am", "PhD"), ("a", "student").
- A language model using skip-gram:

$$P(w_i|w_{i-k},\ldots,w_{i-1},\quad ,w_{i+1},\ldots,w_{i+k})$$

- NOT THE CASE!
- Instead of estimating the probability giving the neighboring words, a skip-gram language model actually estimates the probabilities of neighboring words given a word:
-

$$\sum_{j\in[-k..-1]\cup[1..k]} \log P(w_{i+j}|w_i)$$

- The first equation is actually called Continues BOW (CBOW) model.

skip-grams and CBOW

- We may also add a gap between words.
- 1-degree Skip-grams from the sentence “I am a PhD student”: (“I”, “a”), (“am”, “PhD”), (“a”, “student”).
- A language model using skip-gram:

$$P(w_i | w_{i-k}, \ldots, w_{i-1}, \quad , w_{i+1}, \ldots, w_{i+k})$$

- NOT THE CASE!
- Instead of estimating the probability giving the neighboring words, a skip-gram language model actually estimates the probabilities of neighboring words given a word:
-

$$\sum_{j \in [-k..-1] \cup [1..k]} \log P(w_{i+j} | w_i)$$

- The first equation is actually called Continues BOW (CBOW) model.

skip-grams and CBOW

- We may also add a gap between words.
- 1-degree Skip-grams from the sentence “I am a PhD student”: (“I”, “a”), (“am”, “PhD”), (“a”, “student”).
- A language model using skip-gram:

$$P(w_i|w_{i-k}, \ldots, w_{i-1}, \quad , w_{i+1}, \ldots, w_{i+k})$$

- NOT THE CASE!
- Instead of estimating the probability giving the neighboring words, a skip-gram language model actually estimates the probabilities of neighboring words given a word:
-

$$\sum_{j \in [-k..-1] \cup [1..k]} \quad \log P(w_{i+j}|w_i)$$

- The first equation is actually called Continues BOW (CBOW) model.

skip-grams and CBOW

- We may also add a gap between words.
- 1-degree Skip-grams from the sentence “I am a PhD student”: (“I”, “a”), (“am”, “PhD”), (“a”, “student”).
- A language model using skip-gram:

$$P(w_i | w_{i-k}, \ldots, w_{i-1}, \quad , w_{i+1}, \ldots, w_{i+k})$$

- NOT THE CASE!
- Instead of estimating the probability giving the neighboring words, a skip-gram language model actually estimates the probabilities of neighboring words given a word:
-

$$\sum_{j \in [-k..-1] \cup [1..k]} \log P(w_{i+j} | w_i)$$

- The first equation is actually called Continues BOW (CBOW) model.

skip-grams and CBOW

- We may also add a gap between words.
- 1-degree Skip-grams from the sentence “I am a PhD student”: (“I”, “a”), (“am”, “PhD”), (“a”, “student”).
- A language model using skip-gram:

$$P(w_i | w_{i-k}, \ldots, w_{i-1}, \quad , w_{i+1}, \ldots, w_{i+k})$$

- NOT THE CASE!
- Instead of estimating the probability giving the neighboring words, a skip-gram language model actually estimates the probabilities of neighboring words given a word:
-

$$\sum_{j \in [-k..-1] \cup [1..k]} \log P(w_{i+j} | w_i)$$

- The first equation is actually called Continues BOW (CBOW) model.

skip-grams and CBOW

- We may also add a gap between words.
- 1-degree Skip-grams from the sentence “I am a PhD student”: (“I”, “a”), (“am”, “PhD”), (“a”, “student”).
- A language model using skip-gram:

$$P(w_i | w_{i-k}, \ldots, w_{i-1}, \qquad , w_{i+1}, \ldots, w_{i+k})$$

- NOT THE CASE!
- Instead of estimating the probability giving the neighboring words, a skip-gram language model actually estimates the probabilities of neighboring words given a word:
-

$$\sum_{j \in [-k..-1] \cup [1..k]} \log P(w_{i+j} | w_i)$$

- The first equation is actually called Continues BOW (CBOW) model.

Order of n-grams and interpolation

- Larger n: the string is specific but sparse, e.g., "academia is vicious" may not be in training data.
- Smaller n: dense but general, e.g., lots of "academia is" or maybe "is vicious"
- To balance, mix different orders by interpolation:

$$\lambda_3 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i)$$

- The challenge is how to choose weights. Many people use brutal force.
- Better methods to be presented later.

Order of n-grams and interpolation

- Larger n: the string is specific but sparse, e.g., "academia is vicious" may not be in training data.
- Smaller n: dense but general, e.g., lots of "academia is" or maybe "is vicious"
- To balance, mix different orders by interpolation:

$$\lambda_3 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i)$$

- The challenge is how to choose weights. Many people use brutal force.
- Better methods to be presented later.

Order of n-grams and interpolation

- Larger n: the string is specific but sparse, e.g., “academia is vicious” may not be in training data.
- Smaller n: dense but general, e.g., lots of “academia is” or maybe “is vicious”
- To balance, mix different orders by interpolation:

$$\lambda_3 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i)$$

- The challenge is how to choose weights. Many people use brutal force.
- Better methods to be presented later.

Order of n-grams and interpolation

- Larger n: the string is specific but sparse, e.g., "academia is vicious" may not be in training data.
- Smaller n: dense but general, e.g., lots of "academia is" or maybe "is vicious"
- To balance, mix different orders by interpolation:

$$\lambda_3 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i)$$

- The challenge is how to choose weights. Many people use brutal force.
- Better methods to be presented later.

Order of n-grams and interpolation

- Larger n: the string is specific but sparse, e.g., “academia is vicious” may not be in training data.
- Smaller n: dense but general, e.g., lots of “academia is” or maybe “is vicious”
- To balance, mix different orders by interpolation:

$$\lambda_3 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i)$$

- The challenge is how to choose weights. Many people use brutal force.
- Better methods to be presented later.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where
  - is the number of total vocabulary;
  - is the probability that a word is known to the model (it can be totally arbitrary, e.g.,);
  - is the probability of the word in a unigram model (thus, 0 if the word is not in the model).
- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where

  - N is the number of total vocabulary;
  - λ is the probability that a word is known to the model (it can be totally arbitrary, e.g., 0.95);
  - $P_{\text{model}}(w_i)$ is the probability of the word in a unigram model (which is 0 if the word is not in the model).

- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where
  - N is the number of total vocabulary,
  - λ is the probability that a word is known to the model (it can be totally arbitrary, e.g., 0.95),
  - $P_{\text{model}}(w_i)$ is the probability of the word in uni-gram model (and it is 0 if the word is not in the model).
- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where
  - N is the number of total vocabulary,
  - λ is the probability that a word is known to the model (it can be totally arbitrary, e.g., 0.95),
  - $P_{\text{model}}(w_i)$ is the probability of the word in a unigram model (thus, 0 if the word is not in the model).
- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where
  - N is the number of total vocabulary,
  - λ is the probability that a word is known to the model (it can be totally arbitrary, e.g., 0.95),
  - $P_{\text{model}}(w_i)$ is the probability of the word in a unigram model (thus, 0 if the word is not in the model).
- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where
  - N is the number of total vocabulary,
  - λ is the probability that a word is known to the model (it can be totally arbitrary, e.g., 0.95),
  - $P_{\text{model}}(w_i)$ is the probability of the word in a unigram model (thus, 0 if the word is not in the model).
- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where
  - N is the number of total vocabulary,
  - λ is the probability that a word is known to the model (it can be totally arbitrary, e.g., 0.95),
  - $P_{\text{model}}(w_i)$ is the probability of the word in a unigram model (thus, 0 if the word is not in the model).
- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing

- What about unknown words?
- The problem is known as OOV (out of vocabulary)
- Lazy way: ignore them – closed vocabulary.
- For uni-gram:

$$P_{\text{smoothed}}(w_i) = \lambda P_{\text{model}}(w_i) + (1-\lambda)\frac{1}{N},$$

  where
  - N is the number of total vocabulary,
  - λ is the probability that a word is known to the model (it can be totally arbitrary, e.g., 0.95),
  - $P_{\text{model}}(w_i)$ is the probability of the word in a unigram model (thus, 0 if the word is not in the model).
- It means that all unknown words have the equal probability $\frac{1-\lambda}{N}$.

Smoothing for bi-grams

- Observation: some words have more words following it while others have fewer.
- Make the smoothing depend on the context, i.e., λ is not the same for all words.
- Also leverages the unigram model, i.e., interpolation.
- $P_{\text{smoothed}}(w_i|w_{i-1}) = \lambda_{w_{i-1}} P_{\text{model}}(w_i|w_{i-1}) + (1 - \lambda_{w_{i-1}}) P_{\text{model}}(w_i)$
- Witten-Bell smoothing:

$$\lambda_{w_{i-1}} = \frac{c(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

  where $u(w_{i-1})$ is the number of unique words after w_{i-1} and $c(w_{i-1})$ the total count of w_{i-1} in the training corpus.

Smoothing for bi-grams

- Observation: some words have more words following it while others have fewer.
- Make the smoothing depend on the context, i.e., λ is not the same for all words.
- Also leverages the unigram model, i.e., interpolation.
- $P_{\text{smoothed}}(w_i|w_{i-1}) = \lambda_{w_{i-1}} P_{\text{model}}(w_i|w_{i-1}) + (1 - \lambda_{w_{i-1}}) P_{\text{model}}(w_i)$
- Witten-Bell smoothing:

$$\lambda_{w_{i-1}} = \frac{c(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

where $u(w_{i-1})$ is the number of unique words after w_{i-1} and $c(w_{i-1})$ the total count of w_{i-1} in the training corpus.

Smoothing for bi-grams

- Observation: some words have more words following it while others have fewer.
- Make the smoothing depend on the context, i.e., λ is not the same for all words.
- Also leverages the unigram model, i.e., interpolation.
- $P_{\text{smoothed}}(w_i|w_{i-1}) = \lambda_{w_{i-1}} P_{\text{model}}(w_i|w_{i-1}) + (1 - \lambda_{w_{i-1}}) P_{\text{model}}(w_i)$
- Witten-Bell smoothing:

$$\lambda_{w_{i-1}} = \frac{c(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

where $u(w_{i-1})$ is the number of unique words after w_{i-1} and $c(w_{i-1})$ the total count of w_{i-1} in the training corpus.

Smoothing for bi-grams

- Observation: some words have more words following it while others have fewer.
- Make the smoothing depend on the context, i.e., λ is not the same for all words.
- Also leverages the unigram model, i.e., interpolation.
- $P_{\text{smoothed}}(w_i|w_{i-1}) = \lambda_{w_{i-1}} P_{\text{model}}(w_i|w_{i-1}) + (1 - \lambda_{w_{i-1}}) P_{\text{model}}(w_i)$
- Witten-Bell smoothing:

$$\lambda_{w_{i-1}} = \frac{c(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

where $u(w_{i-1})$ is the number of unique words after w_{i-1} and $c(w_{i-1})$ the total count of w_{i-1} in the training corpus.

Smoothing for bi-grams

- Observation: some words have more words following it while others have fewer.
- Make the smoothing depend on the context, i.e., λ is not the same for all words.
- Also leverages the unigram model, i.e., interpolation.
- $P_{\text{smoothed}}(w_i|w_{i-1}) = \lambda_{w_{i-1}} P_{\text{model}}(w_i|w_{i-1}) + (1 - \lambda_{w_{i-1}}) P_{\text{model}}(w_i)$
- Witten-Bell smoothing:

$$\lambda_{w_{i-1}} = \frac{c(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

  where $u(w_{i-1})$ is the number of unique words after w_{i-1} and $c(w_{i-1})$ the total count of w_{i-1} in the training corpus.

Evaluating a language model

- Likelihood: Given a test set W_{test}, compute the score $\prod_{\mathbf{w} \in W_{test}} P(\mathbf{w}|M)$ where M is the model and $\mathbf{w}$ is a sequence of words.
- Example: Let $W_{test} =$ $\{$"I am a Linuxer", "I am an entrepreneur", "I am a Pythonista"$\}$. The score is $P($"I am a Linuxer"$) \times P($"I am an entrepreneur"$) \times$ $P($"I am a Pythonista"$)$
- But, what if the model covers some sentences very well but not others?
- Entropy
- And, coverage.

Evaluating a language model

- Likelihood: Given a test set W_{test}, compute the score $\prod_{\mathbf{w} \in W_{test}} P(\mathbf{w}|M)$ where M is the model and $\mathbf{w}$ is a sequence of words.
- Example: Let $W_{test} = \{\text{“I am a Linuxer”}, \text{“I am an entreprenuer”}, \text{“I am a Pythonista”}\}$. The score is $P(\text{“I am a Linuxer”}) \times P(\text{“I am an entreprenuer”}) \times P(\text{“I am a Pythonista”})$
- But, what if the model covers some sentences very well but not others?
- Entropy
- And, coverage.

Evaluating a language model

- Likelihood: Given a test set W_{test}, compute the score $\prod_{\mathbf{w} \in W_{test}} P(\mathbf{w}|M)$ where M is the model and $\mathbf{w}$ is a sequence of words.
- Example: Let $W_{test} = \{\text{“I am a Linuxer”}, \text{“I am an entreprenuer”}, \text{“I am a Pythonista”}\}$. The score is $P(\text{“I am a Linuxer”}) \times P(\text{“I am an entreprenuer”}) \times P(\text{“I am a Pythonista”})$
- But, what if the model covers some sentences very well but not others?
- Entropy
- And, coverage.

Evaluating a language model

- Likelihood: Given a test set W_{test}, compute the score $\prod_{\mathbf{w} \in W_{test}} P(\mathbf{w}|M)$ where M is the model and $\mathbf{w}$ is a sequence of words.
- Example: Let $W_{test} = \{\text{“I am a Linuxer”}, \text{“I am an entreprenuer”}, \text{“I am a Pythonista”}\}$. The score is $P(\text{“I am a Linuxer”}) \times P(\text{“I am an entreprenuer”}) \times P(\text{“I am a Pythonista”})$
- But, what if the model covers some sentences very well but not others?
- Entropy
- And, coverage.

Evaluating a language model

- Likelihood: Given a test set W_{test}, compute the score $\prod_{\mathbf{w} \in W_{test}} P(\mathbf{w}|M)$ where M is the model and $\mathbf{w}$ is a sequence of words.
- Example: Let $W_{test} = \{\text{“I am a Linuxer”}, \text{“I am an entreprenuer”}, \text{“I am a Pythonista”}\}$. The score is $P(\text{“I am a Linuxer”}) \times P(\text{“I am an entreprenuer”}) \times P(\text{“I am a Pythonista”})$
- But, what if the model covers some sentences very well but not others?
- Entropy
- And, coverage.

Other language models

Other ways to estimate $P(w_1, \ldots, w_l)$ beyond n-gram:

- Exponential language models: maximum entropy language models, log-bilinear language models.
- neural language models: based on neural networks, embedding

Naïve Bayes spam filter

- A classical application of the BOW model
- Two classes: Spam (S) vs ham (H, non-spam)
- Based on Bayes theorem, given one word w, the chances that a message is spam: $P(S|w) = \frac{P(S,w)}{P(w)} = \frac{P(w|S)P(S)}{P(w|H)P(H)+P(w|S)P(S)}$
- Usually we further assume that $P(H) = P(S)$. Thus $P(S|w) = \frac{P(w|S)}{P(w|H)+P(w|S)}$. Takeaway: all you need is $P(w|H)$ and $P(w|S)$ which can be obtained via counting.
- Naïve Bayes spam filters assume that words appear in an email indepdently. So for two independent words w_1 and w_2 (not necessarily sequential), we have: $P(S|w_1,w_2) = \frac{P(w_1|S)P(w_2|S)}{P(w_1|S)P(w_2|S)+(1-P(w_1|S))(1-P(w_2|S))}$
- Detailed derivation at `http://www.paulgraham.com/naivebayes.html` and `https://www.mathpages.com/home/kmath267/kmath267.htm`

Naïve Bayes spam filter

- A classical application of the BOW model
- Two classes: Spam (S) vs ham (H, non-spam)
- Based on Bayes theorem, given one word w, the chances that a message is spam: $P(S|w) = \frac{P(S,w)}{P(w)} = \frac{P(w|S)P(S)}{P(w|H)P(H)+P(w|S)P(S)}$
- Usually we further assume that $P(H) = P(S)$. Thus $P(S|w) = \frac{P(w|S)}{P(w|H)+P(w|S)}$. Takeaway: all you need is $P(w|H)$ and $P(w|S)$ which can be obtained via counting.
- Naïve Bayes spam filters assume that words appear in an email indepdently. So for two independent words w_1 and w_2 (not necessarily sequential), we have: $P(S|w_1, w_2) = \frac{P(w_1|S)P(w_2|S)}{P(w_1|S)P(w_2|S)+(1-P(w_1|S))(1-P(w_2|S))}$
- Detailed derivation at `http://www.paulgraham.com/naivebayes.html` and `https://www.mathpages.com/home/kmath267/kmath267.htm`

Naïve Bayes spam filter

- A classical application of the BOW model
- Two classes: Spam (S) vs ham (H, non-spam)
- Based on Bayes theorem, given one word w, the chances that a message is spam: $P(S|w) = \frac{P(S,w)}{P(w)} = \frac{P(w|S)P(S)}{P(w|H)P(H)+P(w|S)P(S)}$
- Usually we further assume that $P(H) = P(S)$. Thus $P(S|w) = \frac{P(w|S)}{P(w|H)+P(w|S)}$. Takeaway: all you need is $P(w|H)$ and $P(w|S)$ which can be obtained via counting.
- Naïve Bayes spam filters assume that words appear in an email indepdently. So for two independent words w_1 and w_2 (not necessarily sequential), we have: $P(S|w_1, w_2) = \frac{P(w_1|S)P(w_2|S)}{P(w_1|S)P(w_2|S)+(1-P(w_1|S))(1-P(w_2|S))}$
- Detailed derivation at `http://www.paulgraham.com/naivebayes.html` and `https://www.mathpages.com/home/kmath267/kmath267.htm`

Naïve Bayes spam filter

- A classical application of the BOW model
- Two classes: Spam (S) vs ham (H, non-spam)
- Based on Bayes theorem, given one word w, the chances that a message is spam: $P(S|w) = \frac{P(S,w)}{P(w)} = \frac{P(w|S)P(S)}{P(w|H)P(H)+P(w|S)P(S)}$
- Usually we further assume that $P(H) = P(S)$. Thus $P(S|w) = \frac{P(w|S)}{P(w|H)+P(w|S)}$. Takeaway: all you need is $P(w|H)$ and $P(w|S)$ which can be obtained via counting.
- Naïve Bayes spam filters assume that words appear in an email indepdently. So for two independent words w_1 and w_2 (not necessarily sequential), we have: $P(S|w_1, w_2) = \frac{P(w_1|S)P(w_2|S)}{P(w_1|S)P(w_2|S)+(1-P(w_1|S))(1-P(w_2|S))}$
- Detailed derivation at `http://www.paulgraham.com/naivebayes.html` and `https://www.mathpages.com/home/kmath267/kmath267.htm`

Naïve Bayes spam filter

- A classical application of the BOW model
- Two classes: Spam (S) vs ham (H, non-spam)
- Based on Bayes theorem, given one word w, the chances that a message is spam: $P(S|w) = \frac{P(S,w)}{P(w)} = \frac{P(w|S)P(S)}{P(w|H)P(H)+P(w|S)P(S)}$
- Usually we further assume that $P(H) = P(S)$. Thus $P(S|w) = \frac{P(w|S)}{P(w|H)+P(w|S)}$. Takeaway: all you need is $P(w|H)$ and $P(w|S)$ which can be obtained via counting.
- Naïve Bayes spam filters assume that words appear in an email indepdently. So for two independent words w_1 and w_2 (not necessarily sequential), we have: $P(S|w_1, w_2) = \frac{P(w_1|S)P(w_2|S)}{P(w_1|S)P(w_2|S)+(1-P(w_1|S))(1-P(w_2|S))}$
- Detailed derivation at `http://www.paulgraham.com/naivebayes.html` and `https://www.mathpages.com/home/kmath267/kmath267.htm`

Naïve Bayes spam filter

- A classical application of the BOW model
- Two classes: Spam (S) vs ham (H, non-spam)
- Based on Bayes theorem, given one word w, the chances that a message is spam: $P(S|w) = \frac{P(S,w)}{P(w)} = \frac{P(w|S)P(S)}{P(w|H)P(H)+P(w|S)P(S)}$
- Usually we further assume that $P(H) = P(S)$. Thus $P(S|w) = \frac{P(w|S)}{P(w|H)+P(w|S)}$. Takeaway: all you need is $P(w|H)$ and $P(w|S)$ which can be obtained via counting.
- Naïve Bayes spam filters assume that words appear in an email indepdently. So for two independent words w_1 and w_2 (not necessarily sequential), we have: $P(S|w_1, w_2) = \frac{P(w_1|S)P(w_2|S)}{P(w_1|S)P(w_2|S)+(1-P(w_1|S))(1-P(w_2|S))}$
- Detailed derivation at `http://www.paulgraham.com/naivebayes.html` and `https://www.mathpages.com/home/kmath267/kmath267.htm`

TF-IDF

- How to create a feature vector to detect sentences about a topic?
- Given a search term, e.g., “matlab”, and a bunch of documents, how to rank all documents that match the query?
- Intuition 1: If a document has lots occurences of “matlab” then the document is strongly about MATLAB.
- Intuition 2: However, if all documents contain “matlab”, then the importance of “matlab” in ranking results should reduce.
- TF: Term frequency: the frequency of a term in a document.
- IDF: Inverse document frequency: ratio of total number of documents and the number of documents containing the term.
- TF-IDF: TF $\times$ IDF.

TF-IDF

- How to create a feature vector to detect sentences about a topic?
- Given a search term, e.g., “matlab”, and a bunch of documents, how to rank all documents that match the query?
- Intuition 1: If a document has lots occurences of “matlab” then the document is strongly about MATLAB.
- Intuition 2: However, if all documents contain “matlab”, then the importance of “matlab” in ranking results should reduce.
- TF: Term frequency: the frequency of a term in a document.
- IDF: Inverse document frequency: ratio of total number of documents and the number of documents containing the term.
- TF-IDF: TF $\times$ IDF.

TF-IDF

- How to create a feature vector to detect sentences about a topic?
- Given a search term, e.g., "matlab", and a bunch of documents, how to rank all documents that match the query?
- Intuition 1: If a document has lots occurences of "matlab" then the document is strongly about MATLAB.
- Intuition 2: However, if all documents contain "matlab", then the importance of "matlab" in ranking results should reduce.
- TF: Term frequency: the frequency of a term in a document.
- IDF: Inverse document frequency: ratio of total number of documents and the number of documents containing the term.
- TF-IDF: TF $\times$ IDF.

TF-IDF

- How to create a feature vector to detect sentences about a topic?
- Given a search term, e.g., “matlab”, and a bunch of documents, how to rank all documents that match the query?
- Intuition 1: If a document has lots occurences of “matlab” then the document is strongly about MATLAB.
- Intuition 2: However, if all documents contain “matlab”, then the importance of “matlab” in ranking results should reduce.
- TF: Term frequency: the frequency of a term in a document.
- IDF: Inverse document frequency: ratio of total number of documents and the number of documents containing the term.
- TF-IDF: TF $\times$ IDF.

TF-IDF

- How to create a feature vector to detect sentences about a topic?
- Given a search term, e.g., “matlab”, and a bunch of documents, how to rank all documents that match the query?
- Intuition 1: If a document has lots occurences of “matlab” then the document is strongly about MATLAB.
- Intuition 2: However, if all documents contain “matlab”, then the importance of “matlab” in ranking results should reduce.
- TF: Term frequency: the frequency of a term in a document.
- IDF: Inverse document frequency: ratio of total number of documents and the number of documents containing the term.
- TF-IDF: TF $\times$ IDF.

TF-IDF

- How to create a feature vector to detect sentences about a topic?
- Given a search term, e.g., “matlab”, and a bunch of documents, how to rank all documents that match the query?
- Intuition 1: If a document has lots occurences of “matlab” then the document is strongly about MATLAB.
- Intuition 2: However, if all documents contain “matlab”, then the importance of “matlab” in ranking results should reduce.
- TF: Term frequency: the frequency of a term in a document.
- IDF: Inverse document frequency: ratio of total number of documents and the number of documents containing the term.
- TF-IDF: TF $\times$ IDF.

TF-IDF

- How to create a feature vector to detect sentences about a topic?
- Given a search term, e.g., “matlab”, and a bunch of documents, how to rank all documents that match the query?
- Intuition 1: If a document has lots occurences of “matlab” then the document is strongly about MATLAB.
- Intuition 2: However, if all documents contain “matlab”, then the importance of “matlab” in ranking results should reduce.
- TF: Term frequency: the frequency of a term in a document.
- IDF: Inverse document frequency: ratio of total number of documents and the number of documents containing the term.
- TF-IDF: TF $\times$ IDF.