

Language Understanding Systems

Statistical Language Modeling

Evgeny A. Stepanov

SISL, DISI, UniTN & VUI, Inc.

`evgeny.stepanov@unitn.it`

Outline

- ① Corpora and Counting
 - Lexicon
 - Word Counting
- ② N-grams and N-gram Probabilities
 - N-grams
 - N-gram Probabilities
- ③ Language Models
 - OOV
 - Smoothing
 - Perplexity

Section 1

Corpora and Counting

Corpus

Data Set

- <https://github.com/esrel/NL2SparQL4NLU>
 - NL2SparQL4NLU.train.utterances.txt (train.txt)
 - NL2SparQL4NLU.test.utterances.txt (test.txt)
- Format
 - utterance-per-line
 - tokenized

Ex: Create a lexicon file

Input

Text file

Output

List of word

Ex: Create a lexicon file

Input

Text file

Output

List of word

Algorithm

- ① Tokenize as token per line (tr, sed)
- ② Sort (sort)
- ③ Remove duplicates (uniq)

Ex: Create a lexicon file

Input

Text file

Output

List of word

Algorithm

- ① Tokenize as token per line (tr, sed)
- ② Sort (sort)
- ③ Remove duplicates (uniq)

```
cat train.txt | tr ' ' '\n' | sort | uniq
```

Ex: Counting Words in Text

Input

Text file

Output

List of word with frequency counts

Ex: Counting Words in Text

Input

Text file

Output

List of word with frequency counts

Algorithm

- ① Tokenize as token per line (tr, sed)
- ② Sort (sort)
- ③ Count occurrences of each token (uniq -c)

Ex: Counting Words in Text

Input

Text file

Output

List of word with frequency counts

Algorithm

- ① Tokenize as token per line (tr, sed)
- ② Sort (sort)
- ③ Count occurrences of each token (uniq -c)

```
cat train.txt | tr ' ' '\n' | sort | uniq -c
```

Ex: Counting Words in Text

Case insensitive count

Cut-off

- frequency:
 - most frequent
 - rare

Stop words

- Compare the list to the most frequent words
- Remove stop words

english.stop.txt

Section 2

N-grams and N-gram Probabilities

N-grams

N-gram

n-gram is a contiguous sequence of *n* items from a given sequence of text or speech

- cat
- the cat is fat

Unit	1-gram	2-gram	3-gram
	<i>unigram</i>	<i>bigram</i>	<i>trigram</i>
Markov Order	0	1	2
character	c, a, t	ca, at	cat
word	the, cat, is, fat	the cat, cat is, ...	the cat is, ...

Ex: Counting Bigrams

Ex: Counting Bigrams

Algorithm

- ① Tokenize as token per line
- ② Print $word_i$ and $word_{i+1}$ side by side
- ③ Count

Ex: Counting Bigrams

Algorithm

- ① Tokenize as token per line
- ② Print $word_i$ and $word_{i+1}$ side by side
- ③ Count

Solution

```
cat $data | tr ' ' '\n' > word1
tail -n +2 word1 > word2
paste word1 word2 > bigrams
cat bigrams | sort | uniq -c | sort -gr
```


Ex: Counting Trigrams

Extend bigram counting to trigrams.

N-gram Probabilities

Probability of a Sequence: Chain Rule

$$p(w_1, \dots, w_T) = p(w_1) \prod_{i=2}^T p(w_i | w_1, \dots, w_{i-1}) = p(w_1) \prod_{i=2}^T p(w_i | h_i) \quad (1)$$

$$h_i = \{w_1, \dots, w_{i-1}\} \quad (2)$$

N-gram Probabilities

Probability of a Sequence: Chain Rule

$$p(w_1, \dots, w_T) = p(w_1) \prod_{i=2}^T p(w_i | w_1, \dots, w_{i-1}) = p(w_1) \prod_{i=2}^T p(w_i | h_i) \quad (1)$$

$$h_i = \{w_1, \dots, w_{i-1}\} \quad (2)$$

N-gram History

Truncate history to length $n - 1$:

$$p(w_i | w_1, \dots, w_{i-1}) = p(w_i | w_{i-n+1}, \dots, w_{i-1}) \quad (3)$$

N-gram Probabilities – 2

n-grams

- unigram : $p(w_i)$
- bigram : $p(w_i|w_{i-1})$
- trigram : $p(w_i|w_{i-2}, w_{i-1})$

Estimating N-gram Probabilities

Maximum Likelihood Estimation

Count relative frequencies of the words in a corpus.

N – total number of words in a corpus;

$c(x)$ – number of occurrences of x ;

$$p(w_i) = \frac{c(w_i)}{N} \quad (4)$$

$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \quad (5)$$

Ex: Calculating Probabilities

Unigrams

Calculate (automatically) probability of each token in the corpus

Ex: Calculating Probabilities

Unigrams

Calculate (automatically) probability of each token in the corpus

Calculate N-gram probabilities of:

- $p(\text{the}|\text{of})$
- $p(\text{the}|\text{is})$
- $p(\text{play}|\text{the})$
- all n-gram probabilities of words after 'italy', i.e. $p(*|\text{italy})$

N-grams: Improvement

For consistent probabilistic model

- add beginning and end of sentence tags ($\langle s \rangle$ & $\langle /s \rangle$)
- treat them as other words

$\langle s \rangle$ *the cat is fat* $\langle /s \rangle$

Ex: Sentence Tags

- Add sentence tags;
- Re-extract bigrams and re-calculate their probabilities (automatically).

Section 3

Language Models

Language Models

Grammars vs. Probabilistic LMs

- **Grammars:** boolean decision that string is legal
- **LMs:** probability that string is legal (*more useful for NLP*)

LM as Automaton

LM – a probabilistic automaton to generate sentences.

Algorithm (for Bigram Model)

- ① $w_{i-1} = \langle s \rangle$;
- ② **while** $w_i \neq \langle /s \rangle$
 - ① stochastically get new word w.r.t. $p(w_i|w_{i-1})$

Probability of a String (Examples)

The cat is fat

$$\begin{aligned} p(\langle s \rangle, the, cat, is, fat, \langle /s \rangle) &= \\ &= p(the|\langle s \rangle) * p(cat|the) * p(is|cat) * p(fat|is) * p(\langle /s \rangle|fat) = \\ &= 0.25 * 0.10 * 0.20 * 0.05 * 0.15 = 0.0000375 \end{aligned}$$

The dog is fat

$$\begin{aligned} p(\langle s \rangle, the, dog, is, fat, \langle /s \rangle) &= \\ &= p(the|\langle s \rangle) * p(dog|the) * p(is|dog) * p(fat|is) * p(\langle /s \rangle|fat) = \\ &= 0.25 * 0.08 * 0.16 * 0.05 * 0.15 = 0.000024 \end{aligned}$$

LM Problems

Unseen Words: 'cow'

$$\begin{aligned} p(\langle s \rangle, the, cow, is, fat, \langle /s \rangle) &= \\ &= p(the|\langle s \rangle) * p(cow|the) * p(is|cow) * p(fat|is) * p(\langle /s \rangle|fat) = \\ &= 0.25 * 0.00 * 0.00 * 0.05 * 0.15 = 0.00 \end{aligned}$$

Unseen N-grams

$$\begin{aligned} p(\langle s \rangle, the, cat, is, dog, \langle /s \rangle) &= \\ &= p(the|\langle s \rangle) * p(cat|the) * p(is|cat) * p(dog|is) * p(\langle /s \rangle|dog) = \\ &= 0.25 * 0.10 * 0.20 * 0.00 * 0.00 = 0.00 \end{aligned}$$

Underflow Problem

Small Values

- Probabilities are usually small (< 0)
- Multiplying many of those may cause *underflow* problem

Underflow Problem

Small Values

- Probabilities are usually small (< 0)
- Multiplying many of those may cause *underflow* problem

Solution

- Use the sum of the probabilities' logs instead of product
- $p(a) > p(b)$
- $\log(p(a)) > \log(p(b))$
- $\log(a * b) = \log(a) + \log(b)$
- $p(a) * p(b) \rightarrow \log(p(a)) + \log(p(b))$

Out-Of-Vocabulary (OOV) Rate

- **OOV Rate:** % of word tokens in test data that are not contained in the lexicon (vocabulary)

Out-Of-Vocabulary (OOV) Rate

- **OOV Rate:** % of word tokens in test data that are not contained in the lexicon (vocabulary)
- Empirically each OOV word results in 1.5 - 2 extra errors (> 1 due to the loss of contextual information)

Ex: *Calculate OOV Rate*

Calculate OOV Rate for *test.txt*

Out-Of-Vocabulary (OOV) Words

- How to handle words (in test set) that were never seen in the training data?
- Train a language model with specific token (e.g. 'UNK') for unknown words!

Approaches

- ① Define a vocabulary and replace all other words in training with 'UNK';
- ② Replace the first occurrence of each word with 'UNK';

Data Sparseness

Problem

- Many rare, but possible combinations are not present in training data;
- They have 0 probability, thus
- Whole sequence gets 0 probability;

Data Sparseness

Problem

- Many rare, but possible combinations are not present in training data;
- They have 0 probability, thus
- Whole sequence gets 0 probability;

Solution: Smoothing

- Add some probability to unseen events;
- Remove some probability from seen events – **discounting**;
- Joint probability distribution sums to 1!

Laplace (Add One) Smoothing

Imaginary training data: all possible n-gram combinations occur once.

Bigram: V – bigram vocabulary size

$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V} \quad (6)$$

N-gram: V – total number of possible $(N-1)$ -grams

$$p(w_i|w_{i-N+1}^{i-1}) = \frac{c(w_{i-N+1}^{i-1}, w_i) + 1}{c(w_{i-N+1}^{i-1}) + V} \quad (7)$$

Ex: OOV + Smoothing

Build LM, such that:

- 2-gram
- Case insensitive
- Beginning and end of sentence tags
- Unknown words
- Add-One Smoothing

Ex: OOV + Smoothing

Build LM, such that:

- 2-gram
 - Case insensitive
 - Beginning and end of sentence tags
 - Unknown words
 - Add-One Smoothing
-
- Calculate probabilities of sentences in:
 - *test.txt*

Add-One Smoothing – Cont.d

$$p(w_i | w_{i-N+1}^{i-1}) = \frac{c(w_{i-N+1}^{i-1}, w_i) + 1}{c(w_{i-N+1}^{i-1}) + V} \quad (8)$$

Typically, we assume $V = \{w : c(w) > 0\} \cup \{UNK\}$

Add-One Smoothing – Cont.d

$$p(w_i | w_{i-N+1}^{i-1}) = \frac{c(w_{i-N+1}^{i-1}, w_i) + 1}{c(w_{i-N+1}^{i-1}) + V} \quad (8)$$

Typically, we assume $V = \{w : c(w) > 0\} \cup \{UNK\}$

Add-one smoothing is a worst choice

Smoothing Methods

- Additive smoothing
- Good-Turing estimate
- Jelinek-Mercer smoothing (interpolation)
- Katz smoothing (backoff)
- Witten-Bell smoothing
- Absolute discounting
- Kneser-Ney smoothing

Good Tutorial

<http://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf>

LM Evaluation: Perplexity

- Measures how well model fits test data;
- Probability of test data;
- Weighted average branching factor in predicting the next word (lower is better).

$$PPL = \sqrt[N]{\frac{1}{p(w_1, w_2, \dots, w_N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^N p(w_i | w_{i-N+1})}} \quad (9)$$

- N – number of words in test set;

Ex: Calculate Perplexity (Optional)

Calculate Perplexity for *test.txt*