Language Understanding Systems

Statistical Language Modeling

Evgeny A. Stepanov

SISL, DISI, UniTN
stepanov@disi.unitn.it

Outline

Signals & Interactive Systems

Section 1

Corpora and Counting

Corpus

Data Set

- 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
```

Signals & Interactive Systems

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)

Signals & Interactive Systems

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
```

Signals & Interactive Systems

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**
	unigram	*bigram*	*trigram*
Markov Order	0	1	2
character word	c, a, t the, cat, is, fat	ca, at the cat, cat is, ...	cat 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
```

Signals & Interactive Systems

Ex: Counting Trigrams

Extend bigram counting to trigrams.

N-gram Probabilities

Probability of a Sequence: Chain Rule

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

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

Signals & Interactive Systems

N-gram Probabilities

Probability of a Sequence: Chain Rule

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

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

N-gram History

Truncate history to length $n - 1$:

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

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;

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

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

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:

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

Signals & Interactive Systems

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

$$\langle s \rangle \textit{ the cat is fat } \langle /s \rangle$$

Ex: *Sentence Tags*

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

Signals & Interactive Systems LAB

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

$$p(\langle s\rangle, the, cat, is, fat, \langle/s\rangle) =$$
$$= p(the|\langle s\rangle) * p(cat|the) * p(is|cat) * p(fat|is) * p(\langle/s\rangle|fat) =$$
$$= 0.25 * 0.10 * 0.20 * 0.05 * 0.15 = 0.0000375$$

The dog is fat

$$p(\langle s\rangle, the, dog, is, fat, \langle/s\rangle) =$$
$$= p(the|\langle s\rangle) * p(dog|the) * p(is|dog) * p(fat|is) * p(\langle/s\rangle|fat) =$$
$$= 0.25 * 0.08 * 0.16 * 0.05 * 0.15 = 0.000024$$

Signals & Interactive Systems

LM Problems

Unseen Words: 'cow'

$p(\langle s\rangle, the, cow, is, fat, \langle /s\rangle) =$

$= p(the|\langle s\rangle) * p(cow|the) * p(is|cow) * p(fat|is) * p(\langle /s\rangle|fat) =$

$= 0.25 * 0.00 * 0.00 * 0.05 * 0.15 = 0.00$

Unseen N-grams

$p(\langle s\rangle, the, cat, is, dog, \langle /s\rangle) =$

$= p(the|\langle s\rangle) * p(cat|the) * p(is|cat) * p(dog|is) * p(\langle /s\rangle|dog) =$

$= 0.25 * 0.10 * 0.20 * 0.00 * 0.00 = 0.00$

Signals & Interactive Systems

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
- $p(a) > p(b)$
- $log(p(a)) > log(p(b))$
- $log(a * b) = log(a) + log(b)$
- $p(a) * p(b) \rightarrow log(p(a)) + log(p(b))$

Signals & Interactive Systems

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*

Signals & Interactive Systems LAB

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';

Signals & Interactive Systems

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!

Signals & Interactive Systems

Laplace (Add One) Smoothing

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

Bigram: V – bigram vocabulary size

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

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

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

Signals & Interactive Systems

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 $V = \{w : c(w) > 0\} \cup \{UNK\}$

Signals & Interactive Systems

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

Signals & Interactive Systems

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).

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

- N – number of words in test set;

Ex: Calculate Perplexity (Optional)

Calculate Perplexity for *test.txt*