Language Models

CHIEN CHIN CHEN

Language Modeling (1/10)

Model the way of language speaking.

- Try to <u>model the sequence of word usage</u> of a certain language.

Given $M_{國}, M_{台}, M_{粵}$

- 他給我打 → $P(S|M_{國})$ $\mathbf{P(S|M_{台})}$ $P(S|M_{港})$
- 他打我 → $\mathbf{P(S|M_{國})}$ $P(S|M_{台})$ $P(S|M_{港})$
- 他打我來的 → $P(S|M_{國})$ $P(S|M_{台})$ $\mathbf{P(S|M_{港})}$

For any language model … how can we calculate probabilities over word sequences – $P(w_1w_2w_3w_4)$?

Language Modeling (2/10)

We can always use the **chain rule** to decompose the probability:

$$P(w_1 w_2 w_3 w_4) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)P(w_4|w_1 w_2 w_3)$$

Briefly, the task of language modeling is to predict (model) the next word given the previous words:

$$P(w_n|\underbrace{w_1,\ldots,w_{n-1}}_{\text{history}})$$

Language modeling is a classic NLP problem.

- It is fundamental to **speech recognition**, **machine translation**, **chatting** …
- Sue swallowed the large green _____.

pill

people

This task has been well-studied, and many estimation methods were developed for this problem.

Language Modeling (3/10)

Markov assumption:

- To give reasonable predictions, only the **prior local context** (the last few words) affects the next word.
- If all histories have the same last ***n* – 1 words**, then we have an ***(n – 1)th order Markov model*** or an ***n-gram word model***.
  - E.g., ***bi-gram***: $P(w_n|w_{n-1})$, ***tri-gram***: $P(w_n|w_{n-1}w_{n-2})$.
  - Bi-gram: $P(w_1w_2w_3w_4) = P(w_1)P(w_2|w_1)P(w_3|w_2)P(w_4|w_3)$

The simplest form of language model simply throws away all conditioning context, and estimates each word independently.

$P(w_1w_2w_3w_4) = P(w_1)P(w_2)P(w_3)P(w_4)$

- Such a model is also called a ***unigram language model***.

Language Modeling (4/10)

In principle, we would like the *n* of our *n*-gram models to be fairly large.

- Sue **swallowed** the large **green** _____.

pill

tree

But there will be a problem if we consider a long textual history.

- **The model parameters will be too many to estimate!!!**
- If we assume that our language vocabulary consists of 20,000 words.

Model	Parameters
Bi-gram	20,000 x 19,999 = 400 million
Tri-gram	$20{,}000^2$ x 19,999 = 8 trillion
Four-gram	$20{,}000^3$ x 19,999 = 1.6×10^{17}

Difficult to have a large corpus to produce **reliable parameter estimation**.

- For this reason, *n*-gram systems usually use bi-gram or tri-grams.

Language Modeling (5/10)

But … How can we build *n*-gram model?

- Or, how can we acquire $P(w_n|w_1,\ldots,w_{n-1})$?

We can infer the probability from text (training) corpus.

- If *Kong* is always followed by *Hong* in corpus …
- We should assign *P*(Kong|*Hong*) a high probability and give *P*(*|*Hong*) a small value.

Language Modeling (6/10)

In this training corpus …

- We found 10 training instances of the words *comes across*,
- And of those, 8 times they were followed by *as*,
- Once by *more*,
- Once by *a*.

Then …

$$P(come\ across) = \frac{10}{N}$$

$$P(as \mid come\ across) = \frac{8}{10} = 0.8$$

$$P(more \mid come\ across) = \frac{1}{10} = 0.1$$

$$P(a \mid come\ across) = \frac{1}{10} = 0.1$$

$$P(x \mid come\ across) = \frac{0}{10} = 0$$

for *x* not among the above words

Language Modeling (7/10)

Relative frequency is also called the ***maximum likelihood estimate*** (MLE).

- It makes the probability of observed events as high as it can.

MLE is **unsuitable** for statistical inference in NLP.

- **The sparseness of data** would cause a **zero probability** of uncommon events.
- Then, the probability of a long string will be zero because of zero propagation.
- $P(w_1 w_2 w_3 w_4) =$

$$\underbrace{P(w_1|dummy,dummy)}_{0.8}\underbrace{P(w_2|dummy,w_1)}_{0.75}\underbrace{P(w_3|w_1,w_2)}_{0}\underbrace{P(w_4|w_2,w_3)}_{0.85} = 0!!$$

bcz no such a training instance in the corpus!!

Language Modeling (8/10)

The problem of data sparseness is unavoidable!!

- After training on 1.5 million words from the IBM Laser Patent Text corpus …
- Bahl et al. (1983) report that 23% of the trigram tokens found in further test data drawn from the same corpus were previously unseen.

One might hope that by collecting much more data that the problem of data sparseness would go away.

- Maybe … but it is never a general solution to the problem.
- You can never have a corpus that is large enough to cover everything.
  - E.g., comes across could be followed by any (infinite) number.

Language Modeling (9/10)

The evaluated sentence "*in person she was inferior to both sisters*"

In person	*she*		*was*		*inferior*		*to*		*both*		*sisters*	
1-gram	*P*(.)		*P*(.)		*P*(.)		*P*(.)		*P*(.)		*P*(.)	
	she	0.011	*was*	0.015	*inferior*	0.00005	*to*	0.032	*both*	0.0005	*sisters*	0.0003

P(*S*|1-gram) = 3.96×10^{-17}

In person	*she*		*was*		*inferior*		*to*		*both*		*sisters*	
2-gram	*P*(.|*person*)		*P*(.|*she*)		*P*(.|*was*)		*P*(.|*inferior*)		*P*(.|*to*)		*P*(.|*both*)	
	she	0.009	*was*	0.122	*inferior*	0	*to*	0.212	*both*	0.0004	*sisters*	0.006

P(*S*|2-gram) = 0

In person	*she*		*was*		*inferior*		*to*		*both*		*sisters*	
3-gram	*P*(.|*in person*)		*P*(.|*person she*)		*P*(.|*she was*)		*P*(.|*was inferior*)		*P*(.|*inferior to*)		*P*(.|*to both*)	
	unseen		*was*	0.5	*Inferior*	0	unseen		*both*	0	*sisters*	0

P(*S*|2-gram) = 0

Language Modeling (10/10)

Discounting – a better way to overcome data sparseness.

- To **decrease** the probability of previously seen events.
- So there is a little bit of probability mass left over for previously unseen events.
- Is also referred to as **smoothing**.
  - Presumably because a distribution without zeros is **smoother** than one with zeros.

We introduce three smoothing methods:

- Laplace's law, Lidstone's law, Good-Turing estimation, and Witten-Bell smoothing

Laplace's Law (1/3)

In Laplace's law:

$$P_{Lap}(w_n, \ldots, w_1) = \frac{C(w_n, \ldots, w_1) + 1}{N + B}$$

normalization factor, number of possible *n*-grams.

- Also referred to as ***adding one smoothing***.

Examples in Church and Gale (1991).

- Estimate bi-gram.
- The corpus of 44 million (4.4×10^7) words.
  - Half for training (2.2×10^7 words).
  - Half for testing (2.2×10^7 words).
- Vocabulary of 400,653 words.
- B = 400,653 x 400,653 = 1.6×10^{11} possible bi-grams.
- P_{Lap}(*unseen bi-gram*) = $(0+1) / (2.2 \times 10^7 + 1.6 \times 10^{11}) = 6.2 \times 10^{-12}$.

Laplace's Law (2/3)

The problem of Laplace's law:

- When ***B* >> *N***, Laplace's law would **give to much of the probability to unseen events**.

- For an bi-gram x that never appears in the training corpus,

  $P_{MLE}(x) = 0/2.2\text{x}10^{7}=0$

  $P_{Lap}(x) = (0+1) / (2.2\text{x}10^{7} + 1.6\text{x}10^{11}) = 6.2\text{x}10^{-12}$

  The expected frequency $f_{Lap} = P_{Lap}$ x $N = 6.2\text{x}10^{-12}$ x $2.2\text{x}10^{7}$ = **0.000137**.

- For an bi-gram x that appears once in the training corpus,

  $P_{MLE}(x) = 1/2.2\text{x}10^{7}=4.5\text{x}10^{-8}$

  $P_{Lap}(x) = (1+1) / (2.2\text{x}10^{7} + 1.6\text{x}10^{11}) = 1.25\text{x}10^{-11}$

  The expected frequency $f_{Lap} = P_{Lap}$ x $N = 1.25\text{x}10^{-11}$ x $2.2\text{x}10^{7}$ = **0.000274**.

the difference between MLE and LAP may be ignored!!

Laplace's Law (3/3)

r	f_{Lap}
0	0.000137
1	0.000274
2	0.000411
3	0.000548
4	0.000685

In the training part, 74,671,100,000 bi-grams are unseen bi-grams.

- So 74,671,100,000 x 6.2×10^{-12} = **46.5%** of the probability space has been given to unseen bigrams.
- But … the authors found that only **9.2%** of the bi-grams in further text were previously unseen.
- The discounting of Lapalce's law from seen events is far too much.

Lidstone's Law (1/2)

To overcome the overestimate of Laplace's law, we add not one, but some (normally smaller) positive value λ:

$$P_{Lid}(w_n, \ldots, w_1) = \frac{C(w_n, \ldots, w_1) + \lambda}{N + B\lambda}$$

The method has been showed as a linear interpolation between the MLE estimate and a uniform prior.

$$P_{Lid}(w_n, \ldots, w_1) = \mu \frac{C(w_n, \ldots, w_1)}{N} + (1 - \mu)\frac{1}{B}$$

where $\mu = N/(N+B\lambda)$.

Lidstone's Law (2/2)

The most widely used value for λ is 0.5.

- This method has its own names, the ***Jeffreys-Perks Law***.

In practice, Lidstone's law often helps.

- To avoid too much of the probability space being given to unseen events (by choosing a small λ).
- But … we need a good way to guess an appropriate value for λ.

Good-Turing Estimation (1/5)

The probability estimate in Good-Turing estimation is of the form:

$$P_{GT}(w_1 \ldots w_n) = r^* / N$$

\# of *n*-grams in the training corpus

the **adjusted frequency** of the *n*-gram that appear *r* times in the training corpus

where

$$r^* = \frac{(r+1)N_{r+1}}{N_r}$$

\# of distinct *n*-grams that appear *r* times in the training corpus

For an unseen *n*-gram, the adjusted frequency is:

$$\frac{(0+1)N_{0+1}}{N_0} = \frac{1 * N_1}{N_0}$$

For an *n*-gram which appears only once in the corpus, the adjusted frequency is:

$$\frac{(1+1)N_{1+1}}{N_1} = \frac{2 * N_2}{N_1}$$

Good-Turing Estimation (2/5)

We cannot apply the formula uniformly…

Assuming that the most frequent *n*-gram appears **4** times.

r	0	1	2	3	4
adjust frequency r^*	$1*N_1/N_0$	$2*N_2/N_1$	$3*N_3/N_2$	$4*N_4/N_3$	$5*N_5/N_4$

new estimate will be zero

One possible solution is to use Good-Turing estimation only for frequencies $r < k$ form some constant k (e.g., 10).

- For high frequency words, MLE estimations will be quite accurate.

Good-Turing Estimation (3/5)

If so … it is necessary to **re-normalize** all the estimates to ensure that a proper probability distribution results.

constant k=3

r	0	1	2	3	4
adjusted frequency r^*	$1^*N_1/N_0$	$2^*N_2/N_1$	$3^*N_3/N_2$	$3^*N_3/N_3$	$4^*N_4/N_4$

total adjusted frequency	$N_0(1^*N_1/N_0)$ $= 1^*N_1$	$N_1(2^*N_2/N_1)$ $= 2^*N_2$	$N_2(3^*N_3/N_2)$ $= 3^*N_3$	$N_3(3^*N_3/N_3)$ $= 3^*N_3$	$N_4(4^*N_4/N_4)$ $= 4^*N_4$

total adjusted frequency $(1^*N_1/N_0) \;/\; (1N_1 + 2N_2 + 3N_3 + 3N_3 + 4N_4)$

Good-Turing Estimation (4/5)

The training corpus consists of 617,091 (N) bi-grams.

- V = 14,585 terms.

We saw 199,252 distinct bi-grams in the corpus.

Therefore, there are ($14{,}585^2$ – 199,252) = 212,522,973 unseen bi-grams.

The estimated frequency for r = 0 is (138,741*1) / 212,522,973 = 0.00065 ≈ 0.0007.

The estimated probability for r=0 is 0.0007 / 617,091 ≈ 1.058×10^{-9}

Bi-gram			
r	N_r	r^*	$P_{GT}(.)$
0	212,522,973	0.0007	1.058×10^{-9}
1	138,741	0.3663	5.982×10^{-7}
2	25,413	1.228	2.004×10^{-6}
3	10,531	2.122	3.465×10^{-6}
4	5,997	3.058	4.993×10^{-6}
…			
28	90	26.84	4.383×10^{-5}
29	120	27.84	4.546×10^{-5}
30	86	28.84	4.709×10^{-5}
…			

Good-Turing Estimation (5/5)

Then … for sentence "*she was inferior to both sisters*" the bi-gram based Good-Turing estimate (1.278×10^{-17}) is much higher than the J-P law based estimate (6.89×10^{-20}).

Witten-Bell Smoothing (1/2)

$$P_{WB}(w_n | w_{n-1}, \ldots, w_1)$$

$$= \lambda_{w_{n-1},\ldots,w_1} P_{\mathrm{ML}}(w_n | w_{n-1}, \ldots, w_1) +$$

$$\left(1 - \lambda_{w_{n-1},\ldots,w_1}\right) P_{WB}(w_n | w_{n-1}, \ldots, w_2)$$

- where $\left(1 - \lambda_{w_{n-1},\ldots,w_1}\right)$

$$= \frac{(\#\ of\ terms\ following\ w_{n-1}, \ldots, w_1)}{(\#\ of\ terms\ following\ w_{n-1}, \ldots, w_1) + (\#\ of\ words\ following\ w_{n-1}, \ldots, w_1)}$$

- Is a **recursive interpolate** smoothing method.
- Considering the diversity of the $(w_{n-1}, \ldots, w_1)$.

Witten-Bell Smoothing (2/2)

Consider a bigram model with two histories: *Hong* and *many*:

- They both occur 1000 times in a corpus.
- Hong is always followed by Kong; but many has diverse followers (said 300 terms).
- $$\left(1-\lambda_{Hong}\right)=\frac{1}{1+1000}=0.0009$$
- $$\left(1-\lambda_{many}\right)=\frac{300}{300+1000}=0.23$$
  - Consider the low-order history **more** if the outputs of the history are so diverse

Let's Practice (1/7)

```
In [1]: import io
        orig_text = io.open('language-never-random.txt').read()
        print(orig_text)
```

```
                                                                        ADAM KILGARRIFF

Abstract
Language users never choose words randomly, and language is essentially
non-random. Statistical hypothesis testing uses a null hypothesis, which
posits randomness. Hence, when we look at linguistic phenomena in cor-
pora, the null hypothesis will never be true. Moreover, where there is enough
data, we shall (almost) always be able to establish that it is not true. In
corpus studies, we frequently do have enough data, so the fact that a rela-
tion between two phenomena is demonstrably non-random, does not sup-
port the inference that it is not arbitrary. We present experimental evidence
of how arbitrary associations between word frequencies and corpora are
systematically non-random. We review literature in which hypothesis test-
ing has been used, and show how it has often led to unhelpful or mislead-
ing results.
Keywords: 쎲쎲쎲
```

Let's Practice (2/7)

```
In [2]: from nltk import word_tokenize, sent_tokenize

In [3]: tokenized_text = [list(map(str.lower, word_tokenize(sent))) for sent in sent_tokenize(orig_text)]

In [4]: tokenized_text[0]

Out[4]: ['language',
 'is',
 'never',
 ',',
 'ever',
 ',',
 'ever',
 ',',
 'random',
 'adam',
 'kilgarriff',
 'abstract',
 'language',
 'users',
 'never',
 'choose',
 'words',
 'randomly',
 ',',
 'and',
 'language',
 'is',
```

Let’s Practice (3/7)

```
In [5]: from nltk.lm.preprocessing import padded_everygram_pipeline

In [6]: n_gram_size = 3
        train_data, padded_vocab = padded_everygram_pipeline(n_gram_size, tokenized_text)

In [ ]:

In [7]: from nltk.lm import MLE
        from nltk.lm import Laplace

In [8]: lang_model = Laplace(order = n_gram_size)
        #lang_model = MLE(order= n_gram_size)

In [9]: lang_model.fit(train_data, padded_vocab)
```

Let’s Practice (4/7)

```
In [10]: print(lang_model.vocab.lookup('language is never random lol'.split()))
         ( 'Language', 'is', 'never', 'random', '<UNK>')

In [11]: print(lang_model.counts)
         <NgramCounter with 3 ngram orders and 19611 ngrams>

In [12]: lang_model.counts['language']
Out[12]: 25

In [13]: lang_model.counts[['language']]['is']
Out[13]: 11

In [14]: lang_model.counts[['language', 'is']]['never']
Out[14]: 7
```

```
In [15]: lang_model . counts [[ 'is' , 'language' ]][ 'never' ]
Out[15]: 0

In [16]: lang_model . counts [ 'lol' ]
Out[16]: 0

In [17]: lang_model . score ( 'lol' )
Out[17]: 0.00012250398137939484

In [18]: lang_model . score ( 'never' , 'is language' . split ())
Out[18]: 0.0007189072609633358

In []:

In [19]: lang_model . score ( 'never' , 'language is' . split ())
Out[19]: 0.005706134094151213

In [20]: lang_model . score ( 'language' , 'never is' . split ())
Out[20]: 0.0007189072609633358
```

probability

Let's Practice (5/7)

Four sentences: *Statistical hypothesis testing. Language is never random. Language is* ***random never****. Language is never random* *lol*.

```
In [twenty one]: test_text = 'Statistical hypothesis testing. Language is never random. Language is random never. La
```

```
In [twenty two]: from nltk.util import ngrams
                 from nltk.lm.preprocessing import pad_both_ends


                 tokenized_test = [list(map(str.lower, word_tokenize(sent))) for sent in sent_tokenize(test_text)]
                 test_data = [ngrams(pad_both_ends(t,n=n_gram_size),n_gram_size) for t in tokenized_test]
```

```
In [ ]:
```

```
In [twenty three]: for test in test_data:
                       print(lang_model.perplexity(test))

311.7221611187319
287.16174381713614
451.79749507443495
401.17957042766244
```

The outputs from MLE model!!

```
In [twenty three]: for test in test_data:
                       print(lang_model.perplexity(test))

3.4357982202206863
inf
inf
inf
```

$$\text{perplexity}(w_1 w_2 \ldots w_N) = 2^{-\frac{1}{N} log_2 P(w_1 w_2 \ldots w_N)}$$

The smaller, the better

Let’s Practice (6/7)

Let the language model say something ☺

```
[twenty four]: print ( lang_model . generate ( 20 ) )
```

```
['regarding','the','two','subcorpora',',','viewed','as','col-','lections','of','words','rather', 'than','arbitrar
y','.','</s>','</s>','</s>','</s>','</s>']
```

```
In [25]: from nltk.tokenize.treebank import TreebankWordDetokenizer
```

```
In [26]: detokenizer = TreebankWordDetokenizer().detokenize
```

```
In [27]: def generate_sent(model, num_words):
             content = []
             for token in model.generate(num_words):
                 if token == '<s>':
                     continue
                 if token == '</s>':
                     break
                 content.append(token)
             return detokenizer(content)
```

```
In [30]: generate_sent(lang_model,20)
Out[30]: 'hypothesis testing has been confused with the following data.'
```

Let’s Practice (7/7)

Funny example – Let’s talk like Trump

- https://www.kaggle.com/alvations/n-gram-language-model-with-nltk/
- More than 7,000 tweets of Donald Trump

```
will win this thing! GET OUT TO VOTE Then``WE THE PEOPLE love you trump"Nic
e
```

```
'picked up the mess the U.S. is looking very bad'
```