Evaluation of Language Models

Smoothing

Evaluating Language Model

Does it prefer good sentences to bad sentences?

Assign higher probability to real (or frequently observed) sentences than ungrammatical (or rarely observed) ones

Evaluating Language Model

Does it prefer good sentences to bad sentences?

Assign higher probability to real (or frequently observed) sentences than ungrammatical (or rarely observed) ones

Training and Test Corpora

- Parameters of the model are trained on a large corpus of text, called **training set**.
- Performance is tested on a disjoint (held-out) **test data** using an **evaluation metric**

Two ways of evaluating a Language Model — extrinsic and intrinsic

Extrinsic evaluation of N-grams models

Comparison of two models, A and B

- Use each model for one or more tasks: *spelling corrector, speech recognizer, machine translation*
- Get accuracy values for A and B
- Compare accuracy for A and B

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and . . .
- The president of India is . . .
- I wrote a . . .

How well does a Language Model work in predicting the next word?

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and . . .
- The president of India is . . .
- I wrote a . . .

Unigram model doesn't work for this game. **since it does not use the context (previous words)**

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and ...
- The president of India is ...
- I wrote a ...

Unigram model doesn't work for this game.

A better model of text

is one which assigns a higher probability to the actual word

Perplexity

The best language model is one that best predics an unseen test set

Perplexity ($PP(W)$)

Perplexity is the inverse probability of the test data, normalized by the number of words:

Perplexity

The best language model is one that best predics an unseen test set

Perplexity ($PP(W)$)

Perplexity is the inverse probability of the test data, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

Perplexity

The best language model is one that best predics an unseen test set

Perplexity ($PP(W)$)

Perplexity is the inverse probability of the test data, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

Applying chain Rule

$$PP(W) = \left(\prod \frac{1}{P(w_i | w_1 \dots w_{i-1})} \right)^{\frac{1}{N}}$$

Perplexity

The best language model is one that best predics an unseen test set

Perplexity ($PP(W)$)

Perplexity is the inverse probability of the test data, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

Applying chain Rule

$$PP(W) = \left(\prod \frac{1}{P(w_i|w_1 \ldots w_{i-1})} \right)^{\frac{1}{N}}$$

For bigrams

$$PP(W) = \left(\prod \frac{1}{P(w_i|w_{i-1})} \right)^{\frac{1}{N}}$$

This expression will be different for different LMs.

Example: A Simple Scenario

- Consider a sentence consisting of N random digits

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

$$
\begin{aligned}
PP(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \left(\left(\frac{1}{10}\right)^N\right)^{-\frac{1}{N}}
\end{aligned}
$$

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

$$
\begin{aligned}
PP(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \left(\left(\frac{1}{10}\right)^N\right)^{-\frac{1}{N}} \\
&= \left(\frac{1}{10}\right)^{-1} \\
&= 10
\end{aligned}
$$

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

LM trained over the training set

Perplexity computed over the test set

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

What do these values indicate?

In other words, how to interpret the value of perplexity?

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Unigram perplexity: 962?

The model is as confused on test data as if it had to choose uniformly and independently among 962 possibilities for each word.

Once we have built a Language Model, what can we do?

One interesting application - generate sentences / word sequences

The Shannon Visualization Method

Use the language model to generate word sequences

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram (<s>,w) as per its probability

There is a bigram (<s>, w) for every word w in the vocabulary, and a probability for each such bigram.

One word w will be sampled according to these probabilities.

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram (<s>,w) as per its probability
- Choose a random bigram (w,x) as per its probability

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram (<s>,w) as per its probability
- Choose a random bigram (w,x) as per its probability
- And so on until we choose </s>

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram (<s>,w) as per its probability
- Choose a random bigram (w,x) as per its probability
- And so on until we choose </s>

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

Shakespeare as Corpus

- N = 884,647 tokens, V = 29,066
- Shakespeare produced 300,000 bigram types out of $V^2 = 844$ million possible bigrams.

Approximating Shakespeare

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

Problems with simple MLE estimate: zeros

In Shakespeare's works, only 300,000 bigrams out of possible 844 million bigrams have non-zero probability.

So, even for a large corpus, most bigrams have probability zero.

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Data

- ... denied the offer
- ... denied the loan

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Data

- ... denied the offer
- ... denied the loan

Zero probability n-grams

- P(offer | denied the) = 0
- The test set will be assigned a probability 0
- And the perplexity can't be computed

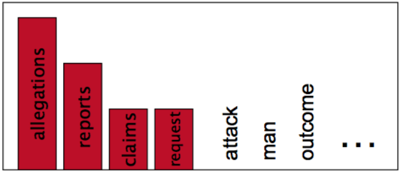
Language Modeling: Smoothing

Language Modeling: Smoothing

With sparse statistics

P(w | denied the)
3 allegations
2 reports
1 claims
1 request

7 total



Probabilities: 3/7, 2/7, 1/7, 1/7

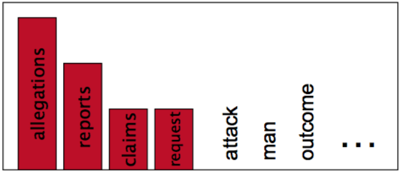
But it is not a good idea to assign probability zero to all other words in the vocabulary.

Smoothing: transfer some probability mass from these words (for which probability is already non-zero) to other words as well. There are several ways to do smoothing.

Language Modeling: Smoothing

With sparse statistics

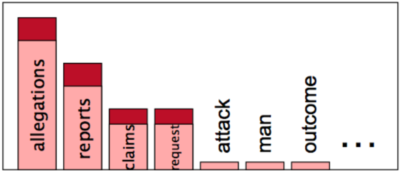
P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



Steal probability mass to generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total

Distribute the probability mass of 2/7 among the other words



Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time that we actually did

Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time that we actually did
- Just add one to all the counts!

Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time that we actually did
- Just add one to all the counts!
- MLE estimate for bigram: $P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$

Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time that we actually did
- Just add one to all the counts!
- MLE estimate for bigram: $P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$
- Add-1 estimate: $P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)+1}{c(w_{i-1})+V}$

Reconstituted counts as effect of smoothing

Effective bigram count ($c^*(w_{n-1}w_n)$)

$$\frac{c^*(w_{n-1}w_n)}{c(w_{n-1})} = \frac{c(w_{n-1}w_n)+1}{c(w_{n-1})+V}$$

Comparing with bigrams: Restaurant corpus

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Comparing with bigrams: Restaurant corpus

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

Add k to the count of each n-gram

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

Let kV = m

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

Unigram prior smoothing:

$$P_{UnigramPrior}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

Unigram prior smoothing:

$$P_{UnigramPrior}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

A good value of k or m?

Can be optimized on held-out set

There are several advanced smoothing algorithms:

Good-Turing
Kneser-Ney

Basic idea - reallocate the probability mass of n-grams that have been seen (in the training data) to the n-grams that were never seen