UF Herbert Wertheim College of Engineering
UNIVERSITY of FLORIDA

CIS 6930 Special Topics in Large Language Models

N-gram Language Models

Language Models (LM)

- LM should have the ability to assign each possible sequence of tokens $w_1, \dots, w_n \in V$ with a meaningful probability: $p(w_1, \dots, w_n)$

$$p(w_1, \dots, w_n)$$

Joint Probability

Chain Rule

$$= \mathrm{p}(w_1) * p(w_2|w_1) * p(w_3|w_1, w_2) * \dots * p(w_n|w_1, w_2, \dots, w_{n-1})$$

E.g. p(cat sat on the mat) = p(cat) * p(sat | cat) * p(on | cat sat) *

p(the | cat sat on) * p(mat | cat sat on the)

Conditional Probability

Language Models (LM)

- LM should have the ability to assign each possible sequence of tokens $w_1, \ldots, w_n \in V$ with a meaningful probability: $p(w_1, \ldots, w_n)$

- LM is a machine learning model that assigns a probability for each possible next word $p(w_n \mid w_1, \ldots, w_{n-1})$ (predicting upcoming words)

- This is how LLM work: (1) LLMs are **trained** to predict words (2) LLMs **generate** text by predicting the next word repeatedly.

N-gram Language Model

- N-gram Language Model – the simplest kind of language model

- What is n-gram: n-gram is a sequence of n words

2-gram (bigram) = two-word sequence of words, e.g. the cat, cat sat…

3-gram (trigram) = three-word sequence of words, e.g. the cat sat …

- N-gram LM is a probabilistic model that can estimate the prob of a word given the **N-1** previous words, and thereby to assign prob for entire sequences

Outline

- Building N-gram Language Model
- Generation from N-gram Language Model
- Evaluating N-gram Language Model
- Smoothing of N-gram Language Model

Building a N-gram Language Model

How to estimate probability

- One way to estimate $p(w|h)$ prob of a word w given some history h is using **relative frequency counts**

$$P(\text{sat}|\text{the cat}) = \frac{\text{count}(\text{the cat sat})}{\text{count}(\text{the cat})}$$

$$P(\text{on}|\text{the cat sat}) = \frac{\text{count}(\text{the cat sat on})}{\text{count}(\text{the cat sat})}$$

MLE: Maximum Likelihood Estimation

Problems of relative frequency counts

- **Not accurate** because new sentences are invented all the time

  Even the entire web is not big enough to give us good estimates for counts of every possible sentences

- **Computational heavy** for sequence with long length n

  With a vocabulary of size V, # sequence of length n = V^n

  Typical English vocabulary V ~ 40k words

  Even sentences of length <= 11 results in more than 4 x 10^{50} sequences

Markov Assumption

- Markov Assumption definition – the future state conditioned on the current state (or recent several states) is independent of past states

- Use only the recent past to predict the next word

1-st order $$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the})$$

2-nd order $$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the})$$

N-gram Language Model

- Instead of computing the prob of word given its entire history, we only use the last **N-1** steps to approximate all the history

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

$$p(w_1, \ldots, w_n) = \prod_i p(w_i|w_{i-N+1}, \ldots, w_{i-1})$$

N-gram Language Model

Unigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i)$$ e.g. P(the) P(cat) P(sat)

Bigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$ e.g. P(the) P(cat | the) P(sat | cat)

$$p(w_i \,|w_{i-1}) = \frac{C(w_{i-1},w_i)}{C(W_{i-1})}$$

Larger N, more accurate and better the LM is, but also higher costs

Practice – Estimating Conditional Probabilities

Consider the following corpus

<s> I like apples </s>

<s> You like strawberries </s>

<s> You like apples </s>

Note: <s> and </s> are starting and ending tokens

What's the bigram probability P(apples | like) ?

(A) 1/3 (B) 2/3 (C) 1/2 (D) 1

Practice – Estimating Conditional Probabilities

Consider the following corpus

<s> I like apples </s>

<s> You like strawberries </s>

<s> You like apples </s>

Note: <s> and </s> are starting and ending tokens

What's the bigram probability P(apples | like) ?

(A) 1/3 (B) 2/3 (C) 1/2 (D) 1

$$P(\text{apples} \mid \text{like}) = \frac{\text{Count("like apples")}}{\text{Count("like")}} = \frac{2}{3}$$

Practice – Estimating Joint Probabilities

Consider the following corpus

<s> I like apples </s>

<s> You like strawberries </s>

<s> You like apples </s>

Note: <s> and </s> are starting and ending tokens

Using the bigram model, what's the probability of the sentence "<s> I like strawberries </s>"? Ignore the probability of <s>.

(A) 4/9 (B) 1/3 (C) 2/9 (D) 1/9

Practice – Estimating Joint Probabilities

Consider the following corpus

<s> I like apples </s>

<s> You like strawberries </s>

<s> You like apples </s>

Note: <s> and </s> are starting and ending tokens

Using the bigram model, what's the probability of the sentence "<s> I like strawberries </s>"? Ignore the probability of <s>.

(A) 4/9 (B) 1/3 (C) 2/9 (D) 1/9

$$P(\text{<s> I like strawberries </s>}) = \frac{1}{3} \cdot 1 \cdot \frac{1}{3} \cdot 1$$

Scaling in Large N-gram Models – Log Probability

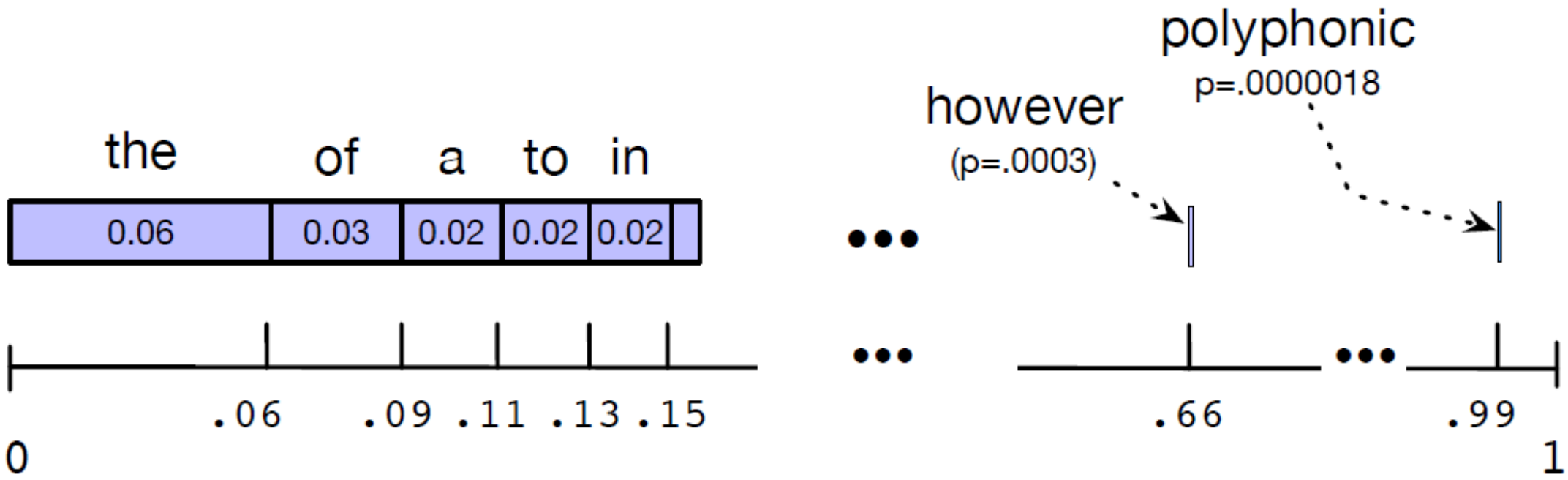
- Probability Numerical Underflow: Because probabilities are <= 1, the more probs we multiply together, the smaller the product becomes

- Log Probability: adding in log space is equivalent to multiplying in linear space, and we get results that are not small

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Generation from N-gram Language Model

Sampling a word from a prob distribution



Generating sequences from bigram LM

Bigram $P(w_1, w_2, \ldots w_n) = \prod_{i=1}^{n} P(w_i | w_{i-1})$

- Generate the first word $w_1 \sim P(w)$
- Generate the second word $w_2 \sim P(w \mid w_1)$
- Generate the third word $w_3 \sim P(w \mid w_2)$
- ...

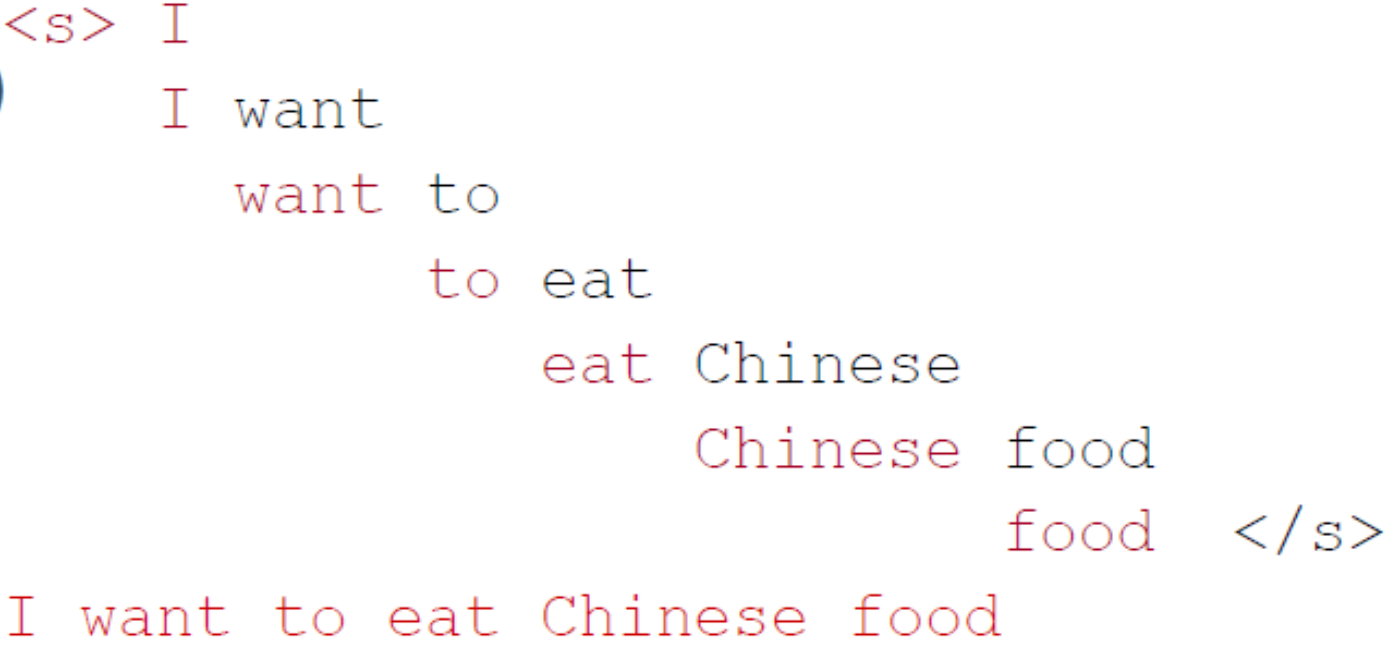
Generating sequences from bigram LM

Choose a random bigram (<s>, w)
according to its probability p(w|<s>)

Now choose a random bigram (w, x)
according to its probability p(x|w)

And so on until we choose </s>

Then string the words together



Generating sequences from trigram LM

Trigram $P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$

- Generate the first word $w_1 \sim P(w)$
- Generate the second word $w_2 \sim P(w \mid w_1)$
- Generate the third word $w_3 \sim P(w \mid w_1, w_2)$
- Generate the fourth word $w_4 \sim P(w \mid w_2, w_3)$
- …

Evaluating N-gram Language Model

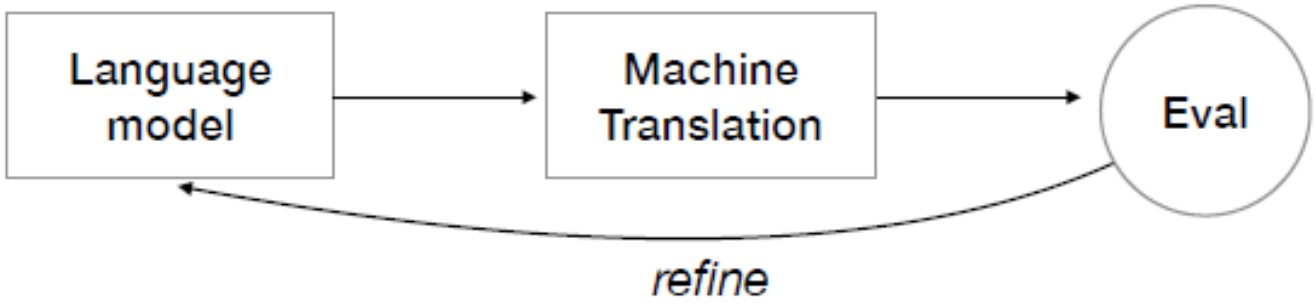
Two types of evaluation for LM

- Extrinsic Evaluation

Train LM -> apply to downstream task -> evaluate the task

Higher task performance -> better model

Problem: expensive, time consuming, indirect feedback



Two types of evaluation for LM

- Intrinsic Evaluation

Measures the quality of a LM independent of any application

How well a LM estimates the probability of sequences

How to evaluate: training set, testing set, development set, metric

Intrinsic Evaluation – split datasets

- **Training set:** is the data we use to learn the parameter of model

For the N-gram LM, training data is the data from which we get the n-gram counts to estimate n-gram probability

- **Testing set:** is a set of unseen data that is not overlapping with training set, that we use to evaluate the model

If a LM assigns a higher probability to the test set, that means it more accurately predicts the test set, then it is a better model.

- **Development set:** used to tune hyper-parameter

Intrinsic Evaluation – metric

- Recall: if a LM assigned a higher probability to the test set, it is better

- Question: Can we use probability as the evaluation metric?

We do not use the raw probability as the metric, because the prob of a test set depends on the number of words/tokens – the prob gets smaller if the test text is longer.

- Metric: **Perplexity** (a metric normalized by sequence length, per-word)

Intrinsic Evaluation – perplexity

- Perplexity has an inverse relation with probability
- Better LM = higher estimated probability on test set = lower perplexity

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

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain Rule

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Bigram LM

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

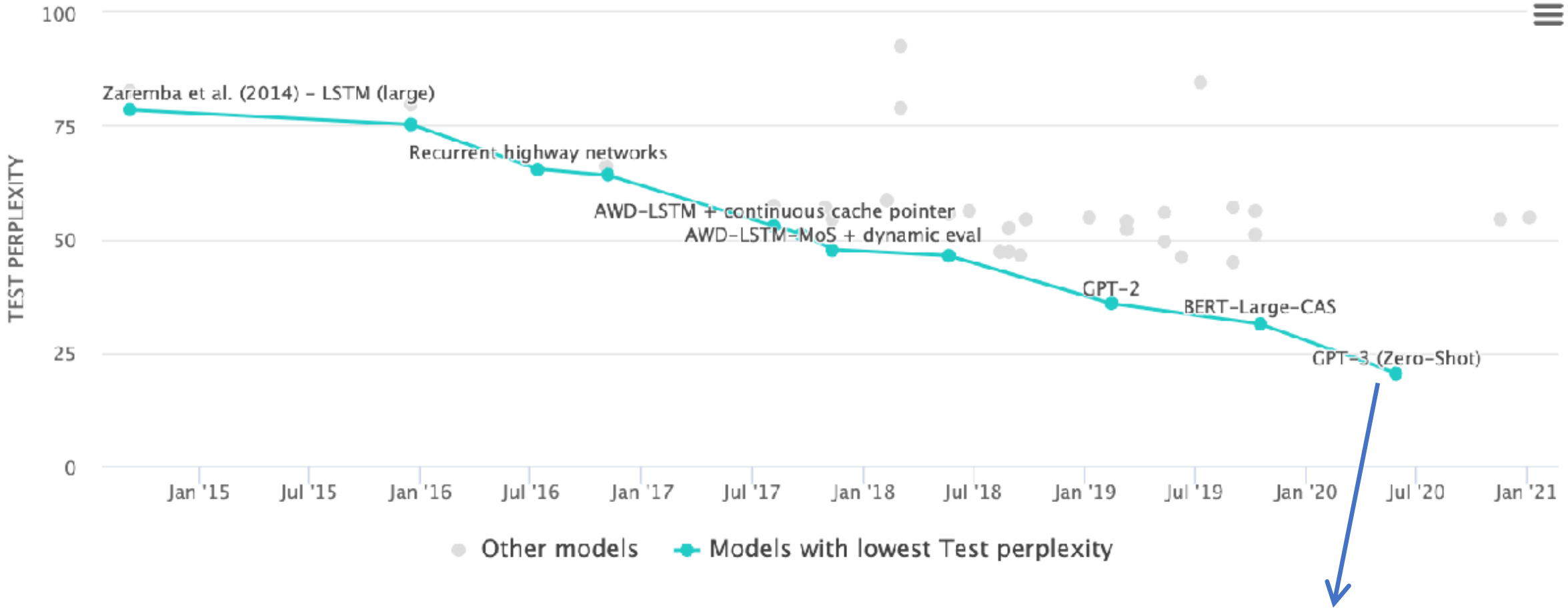
Intrinsic Evaluation – perplexity

- Example: unigram, bigram, trigram models on 38M words from the *Wall Street Journal* news paper, their perplexity on test corpus (1.5M words)

	Unigram	Bigram	Trigram
Perplexity	962	170	109

Trigram model gives us more information about word sequence, has a better idea of what words might come next, thus assigns a higher prob

Intrinsic Evaluation – perplexity



GPT-3 175B: perplexity = 20.5

Smoothing of N-gram Language Model

Problem of N-gram Language Model

- Not all n-grams in the test set will be observed in training data

Example: Training set is Google News, Testing set is Shakespeare

P(affray | voice doth us) = 0

- Problem of a particular n-gram never occurs in the training but appears in the test set:

(1) Underestimate the probability of words sequences that might occur

(2) The prob of the whole test set will also be 0, perplexity no sense

Smoothing

Smoothing is to make sure the probabilities for all the possible n-gram are non-zero in our model

- Laplace Smoothing: Adding a small amount to all probabilities
- Interpolation: Combining different granularities of n-gram LM
- Backoff: Backing off to lower n-gram

Smoothing – Laplace Smoothing

Adding a small amount to all probabilities

Max Likelihood Estimate for Bigram

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

After Laplace Smoothing

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha \mid V \mid}$$

Smoothing – Linear Interpolation

Combining different granularities of n-gram LM to estimate probability

$$\hat{P}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) \quad \text{Trigram}$$
$$+\lambda_2 P(w_i|w_{i-1}) \quad \text{Bigram}$$
$$+\lambda_3 P(w_i) \quad \text{Unigram}$$

$$\sum_i \lambda_i = 1$$

Process: (1) estimate n-gram prob on training set (2) tune hyper-parameter (lambdas) to maximize prob on dev set (3) evaluate on test set

Smoothing – Backoff

If the n-gram we need has zero counts, we approximate it by backing off to the (n-1) gram, until we reach a history that has non-zero counts.

Suppose you want:

P(pancakes| delicious soufflé)

If the trigram probability is 0, use the bigram

P(pancakes| soufflé)

If the bigram probability is 0, use the unigram

P(pancakes)


Thank you!
UF
Herbert Wertheim
College of Engineering
UNIVERSITY of FLORIDA
LEADING THE CHARGE, CHARGING AHEAD

References

Speech and Language Processing (3rd ed. draft)

Dan Jurafsky and James H. Martin

Chapter 3