NLP 201
Language Modeling

Jeffrey Flanigan

University of California Santa Cruz
jmflanig@ucsc.edu

Fall 2023

Plan

- Language modeling
- Overfitting
- Smoothing, interpolation
- Picking hyperparameters
- Evaluation

What is the probability of …

"I like Georgia Tech at Atlanta"

"like I Atlanta atTech Georgia"

The Language Modeling Problem

Assign a probability to every sentence (or any string of words)

- Finite vocabulary (e.g., words or characters) *{the, a, telescope, ...}*
- Infinite set of sequences
  - *A telescope STOP*
  - *A STOP*
  - *The the the STOP*
  - *I saw a woman with a telescope STOP*
  - *STOP*
  - ...

What is the probability of …

I like Georgia Tech at Atlanta

P(I like Georgia Tech at Atlanta STOP) = 10^{-5}

like I Atlanta at Tech Georgia

P(like I Georgia Tech at Atlanta STOP) = 10^{-15}

What Is A Language Model?

- Probability distributions over sentences (i.e., word sequences)

  P(W) = P($w_1 w_2 w_3 w_4 \ldots w_k$)

- Can use them to generate strings

  P(w_k | $w_2 w_3 w_4 \ldots w_{k-1}$)

- Rank possible sentences
  - P("*Today is Tuesday*") > P("*Tuesday Today is*")
  - P("*Today is Tuesday*") > P("*Today is Atlanta*")

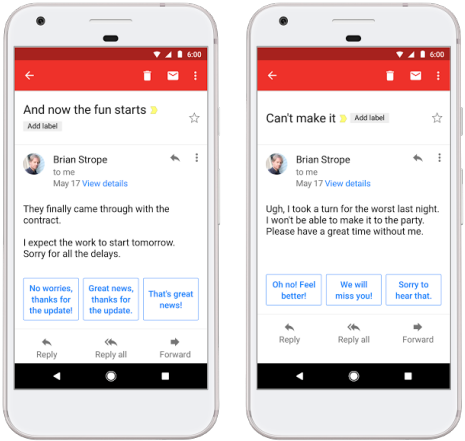
Language Model Applications

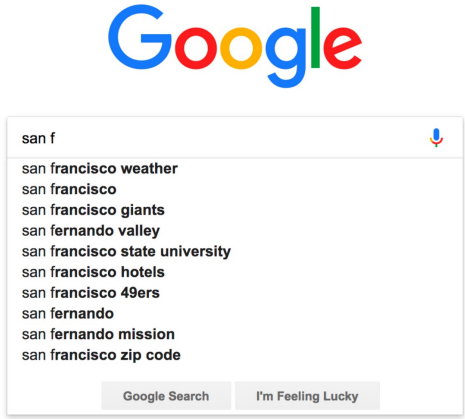
- Machine Translation
  - *p(strong winds) > p(large winds)*
- Spell Correction
  - *The office is about 15 minutes from my house*
  - *p(15 minutes from my house) > p(15 minuets from my house)*
- Speech Recognition
  - *p(I saw a van) >> p(eyes awe of an)*
- Summarization, question-answering, handwriting recognition, etc..

Language Models Play the Role of …

- A judge of grammaticality
- A judge of semantic plausibility
- An enforcer of stylistic consistency
- A repository of knowledge (?)

Language Model Applications





Language Model Applications

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

Abstract

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. Introduction

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the investigation of Markov models is rarely outdated. A theoretical grand challenge in theory is the important unification of virtual machines and real-time theory. To what extent can web browsers be constructed to achieve this purpose?

Certainly, the usual methods for the emulation of Smalltalk that paved the way for the investigation of rasterization do not apply in this area. In the opinions of many, despite the fact that conventional wisdom states that this grand challenge is continuously answered by the study of access points, we believe that a different solution is necessary. It should be noted that Rooter runs in $\Omega(\log \log n)$ time. Certainly, the shortcoming of this type of solution, however, is that compilers and superpages are mostly incompatible. Despite the fact that similar methodologies visualize XML, we surmount this issue without synthesizing distributed archetypes.

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-tauted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in $\Omega((n + \log n))$ time [22]. In the end, we conclude.

II. Architecture

Our research is principled. Consider the early methodology by Martin and Smith; our model is similar, but will actually overcome this grand challenge. Despite the fact that such a claim at first glance seems unexpected, it is buffetted by previous work in the field. Any significant development of secure theory will clearly require that the acclaimed real-time algorithm for the refinement of write-ahead logging by Edward Feigenbaum et al. [15] is impossible; our application is no different. This may or may not actually hold in reality. We consider an application consisting of n access points. Next, the model for our heuristic consists of four independent components: simulated annealing, active networks, flexible modalities, and the study of reinforcement learning.

We consider an algorithm consisting of n semaphores. Any unproven synthesis of introspective methodologies will

Language generation

https://pdos.csail.mit.edu/archive/scigen/

Language Models and Prompting

- You can provide a **prompt** to a language model and it can complete it.
- Can perform many different tasks: machine translation, summarization, question answering, etc
- GPT-2 paper "Language Models are Unsupervised Multitask Learners" (2019)
- Hot topic in NLP right now

Language Models and Few-Shot Prompting

- You can provide a prompt with a few examples to a language model (improves performance over 1 example)
- Also called **in-context learning**
- GPT-3 paper "Language Models are Few-Shot Learners" (2021)
- Game-changer for NLP. Allows new tasks with very little data

A Trivial Model

- Assume we have n training sentences
- Let $x_1, x_2, \ldots, x_n$ be a sentence, and $c(x_1, x_2, \ldots, x_n)$ be the number of times it appeared in the training data.
- Define a language model $p(x_1, x_2, \ldots x_n) = \frac{c(x_1, x_2, \ldots, x_n)}{N}$

A Trivial Model

- Assume we have n training sentences
- Let $x_1, x_2, \ldots, x_n$ be a sentence, and $c(x_1, x_2, \ldots, x_n)$ be the number of times it appeared in the training data.
- Define a language model $p(x_1, x_2, \ldots x_n) = \frac{c(x_1, x_2, \ldots, x_n)}{N}$

No generalization!

Better: Each word independent

Probability of sentence

$$p(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} p(X_i = x_i)$$

What else could we do?

Chain rule (of probability)

$$
\begin{aligned}
P(x_1, x_2, x_3, x_4, x_5) = P(x_1) \\
&\times P(x_2 \mid x_1) \\
&\times P(x_3 \mid x_1, x_2) \\
&\times P(x_4 \mid x_1, x_2, x_3) \\
&\times P(x_5 \mid x_1, x_2, x_3, x_4)
\end{aligned}
$$

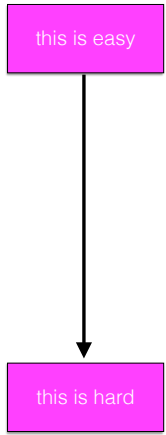
Chain rule (of probability)

P(“It was the best of times, it was the worst of times”)

Chain rule (of probability)

P(“It”)

P(“was” | “It”)



$$P(w_1)$$

$$P(w_2 \mid w_1)$$

$$P(w_3 \mid w_1, w_2)$$

$$P(w_4 \mid w_1, w_2, w_3)$$

$$P(w_n \mid w_1, \ldots, w_{n-1})$$

P(“times” | “It was the best of times, it was the worst of”)

Markov assumption

first-order

$$P(x_i \mid x_1, \ldots x_{i-1}) \approx P(x_i \mid x_{i-1})$$

second-order

$$P(x_i \mid x_1, \ldots x_{i-1}) \approx P(x_i \mid x_{i-2}, x_{i-1})$$

Markov assumption

bigram model
(first-order markov)

$$\prod_{i}^{n} P(w_i \mid w_{i-1}) \times P(\text{STOP} \mid w_n)$$

trigram model
(second-order markov)

$$\prod_{i}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$
$$\times P(\text{STOP} \mid w_{n-1}, w_n)$$

"It was the best of times, it was the worst of times"

$$P(It \mid \text{START}_1, \text{START}_2)$$

$$P(was \mid \text{START}_2, It)$$

$$P(the \mid It, was)$$

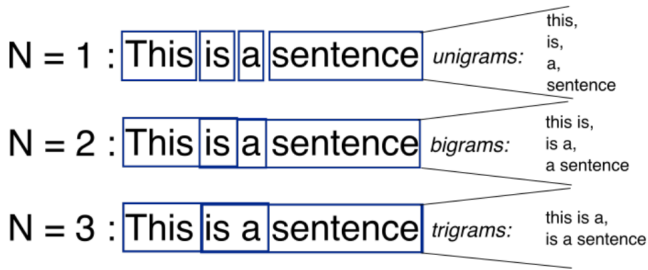
$$\ldots$$

$$P(times \mid worst, of)$$

$$P(\text{STOP} \mid of, times)$$

N-grams (unigram, bigram, trigram, 4-gram...)

- **N-grams:** a contiguous sequence of n tokens from a given piece of text



13

http://recognize-speech.com/language-model/n-gram-model/comparison

N-grams Models

- Unigram model: $P(w_1)P(w_2)P(w_3) \dots P(w_n)$
- Bigram model: $P(w_1)P(w_2|w_1)P(w_3|w_2) \dots P(w_n|w_{n-1})$
- Trigram model: $P(w_1)P(w_2|w_1)P(w_3|w_2, w_1) \dots P(w_n|w_{n-1}w_{n-2})$
- N-gram model: $P(w_1)P(w_2|w_1) \dots P(w_n|w_{n-1}w_{n-2} \dots w_{n-N})$

Details: Variable Length

- We want probability distribution over sequences of any length

Details: Variable Length

- Define always $x_n = STOP$, where STOP is a special symbol
- Then use a Markov process as before:

$$p(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \prod_{i=1}^{n} p(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

- We now have probability distribution over all sequences
  - Intuition: at every step you have probability α_n to stop and $1 - \alpha_n$ to keep going

Details: START symbol

- Conventional to add the symbol START at the beginning of the sentence
- You can think of this as a R.V. that has the value START with probability 1
- Equivalently, don't include this symbol as part of the probability calculation (it just multiplies by 1)
- Don't include START in the vocabulary (since it's never generated)
- Condition on START when generating the first word
- *When evaluating, don't include this symbol in the perplexity calculation*

The Process of Generating Sentences

Step 1: Initialize $i = 1$ and $x_0 = x_{-1} = *$

Step 2: Generate x_i from the distribution

$$p(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

Step 3: If $x_i = STOP$ then return the sequence $x_1 \cdots x_i$. Otherwise, set $i = i + 1$ and return to step 2.

Tri-gram Language Model

- A trigram language model contains
  - A vocabulary V
  - A nonnegative parameter $q(w|u,v)$ for every trigram, such that

$$w \in V \cup \{\text{STOP}\}, u, v \in V \cup \{*\}$$

  - The probability of a sentence $x_1, x_2, \ldots, x_n$, where $x_n = \text{STOP}$ is

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} q(x_i|x_{i-1}, x_{i-2})$$

Tri-grams LM Example

$$
\begin{aligned}
p(\text{the dog barks STOP}) &= q(\text{the}|*,*) \times \\
&= q(\text{dog}|*,\text{the}) \times \\
&= q(\text{barks}|\text{the},\text{dog}) \times \\
&= q(\text{STOP}|\text{dog},\text{barks})
\end{aligned}
$$

Limitations

- Markovian assumption is false

He is from France, so it makes sense that his first language is …

- We want to model longer dependencies

Parameter Estimation

Parameter estimation

We have some data (a corpus), and would like to learn a language model.

Following ML practices, you would split the data into:

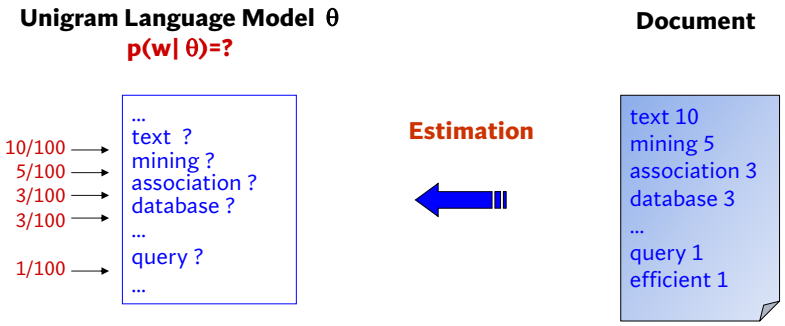
- training set for learning the model parameters
- development set ("dev or held-out set") for tuning hyperparameters
- testing set for evaluation (never look at this set, only evaluate at the end)

Maximum Likelihood Estimation

- "Best" means "data likelihood reaches maximum"

$$\widehat{\boldsymbol{\theta}} = \mathbf{argmax}_{\boldsymbol{\theta}} \mathbf{P}(\mathbf{X}|\boldsymbol{\theta})$$

Maximum Likelihood Estimation



Maximum Likelihood Estimation

- Data: a collection of words, $w_1, w_2, \ldots, w_n$
- Model: multinomial distribution $\mathrm{p}(W)$ with parameters $\theta_i = p(w_i)$

- Maximum Likelihood Estimator (MLE):

$$\hat{\theta} = argmax_{\theta \in \Theta} p(W|\theta)$$

Maximum Likelihood Estimation

$$p(W|\theta) = \binom{N}{c(w_1), \ldots, c(w_N)} \prod_{i=1}^{N} \theta_i^{c(w_i)} \propto \prod_{i=1}^{N} \theta_i^{c(w_i)}$$

$$\Rightarrow \log p(W|\theta) = \sum_{i=1}^{N} c(w_i) \log \theta_i + const$$

$$\hat{\theta} = argmax_{\theta \in \Theta} \sum_{i=1}^{N} c(w_i) \log \theta_i$$

Maximum Likelihood Estimation

$$\hat{\theta} = argmax_{\theta \in \Theta} \sum_{i=1}^{N} c(w_i) \log \theta_i$$

$$L(W, \theta) = \sum_{i=1}^{N} c(w_i) \log \theta_i + \lambda \left(\sum_{i=1}^{N} \theta_i - 1 \right)$$

Lagrange multiplier

$$\frac{\partial L}{\partial \theta_i} = \frac{c(w_i)}{\theta_i} + \lambda \quad \rightarrow \quad \theta_i = -\frac{c(w_i)}{\lambda}$$

Set partial derivatives to zero

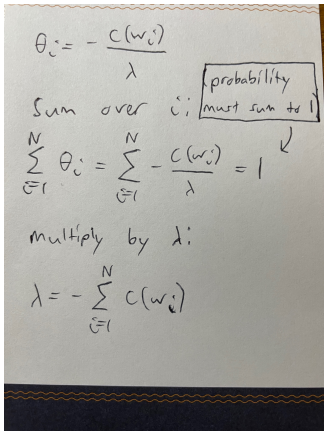
Since $\sum_{i=1}^{N} \theta_i = 1$ we have $\lambda = -\sum_{i=1}^{N} c(w_i)$

Requirement from probability

$$\theta_i = \frac{c(w_i)}{\sum_{i=1}^{N} c(w_i)}$$

ML estimate

Derivation Details (Solving for λ)



Maximum Likelihood Estimation

- For N-gram language models

$$p(w_i|w_{i-1}, \dots, w_{i-n+1}) = \frac{c(w_i, w_{i-1}, \dots, w_{i-n+1})}{c(w_{i-1}, \dots, w_{i-n+1})}$$

Estimation

unigram

$$\prod_{i}^{n} P(w_i) \times P(STOP)$$

bigram

$$\prod_{i}^{n} P(w_i \mid w_{i-1}) \times P(STOP \mid w_n)$$

trigram

$$\prod_{i}^{n} P(w_i \mid w_{i-2}, w_{i-1}) \times P(STOP \mid w_{n-1}, w_n)$$

Maximum likelihood estimate

$$\frac{c(w_i)}{N}$$

$$\frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$\frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}$$

Board work: Language model

Training Data:
<START> I saw Mary <STOP>
<START> I saw John <STOP>
<START> Kat <STOP>
Test Sentence:
<START> I saw Kat <STOP>

- (3 min) Calculate the probability of the test sentence using a bigram language model
- (3 min) Discuss with a partner

Working in Log Space

We do everything in log space

- Avoid underflow from multiplying tiny numbers

$$\log(p_1 \times p_2) = \log(p_1) + \log(p_2)$$

Dealing with Overfitting

Overfitting

- Often in machine learning, when we fit a model it will perform better on the training set than on the testing test.
- This is called **overfitting**.

Extreme example of overfitting

- Assume we have n training sentences
- Let $x_1, x_2, \ldots, x_n$ be a sentence, and $c(x_1, x_2, \ldots, x_n)$ be the number of times it appeared in the training data.
- Define a language model $p(x_1, x_2, \ldots x_n) = \frac{c(x_1, x_2, \ldots, x_n)}{N}$

No generalization!

How about Unseen Words/Phrases

- Example: Shakespeare corpus consists of N=884,647 word tokens and a vocabulary of V=29,066 word types
- Only 30,000 word types occurred
  - Words not in the training data ⇒ **0** probability
- Only 0.04% of all possible bigrams occurred

Types vs tokens

Tokens are the tokens in the document or sentence.
Types are the unique tokens.

Example sentence:

the cat sat on the mat .

Tokens: (the, cat, sat, on, the, mat, .)
Types: { the, cat, sat, on, mat, . }

Sparsity

- Sparse data due to Zipf's Law
- Example: the frequency of different words in a large text corpus

any word

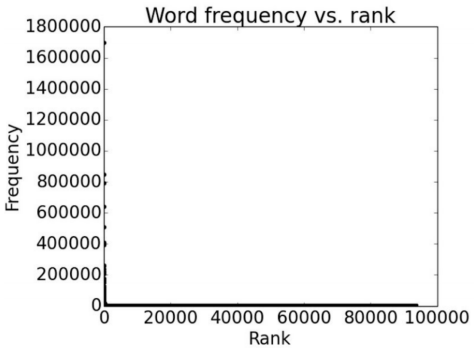
Frequency	Token
1,698,599	the
849,256	of
793,731	to
640,257	and
508,560	in
407,638	that
400,467	is
394,778	a
263,040	I

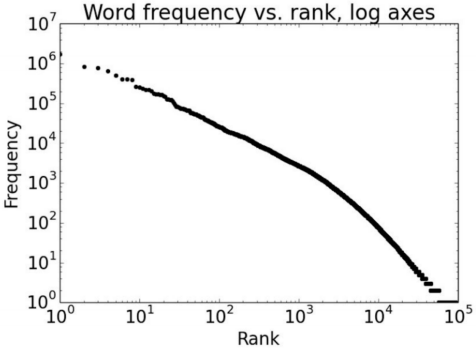
nouns

Frequency	Token
124,598	European
104,325	Mr
92,195	Commission
66,781	President
62,867	Parliament
57,804	Union
53,683	report
53,547	Council
45,842	States

Sparsity

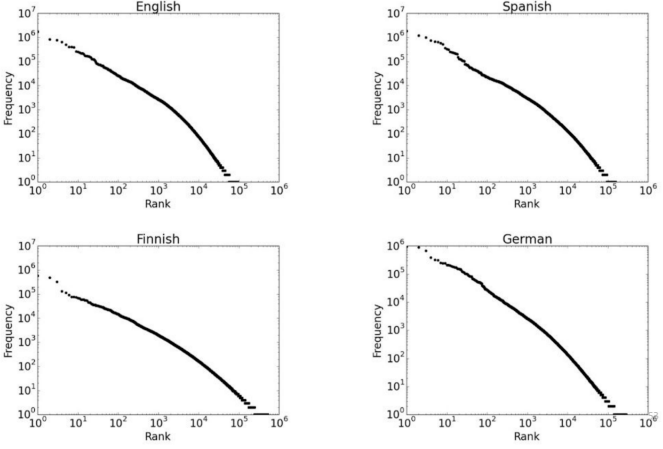
- Order words by frequency. What is the frequency of nth ranked word?





Sparsity

- Regardless of how large our corpus is, there will be a lot of infrequent words
- This means we need to find clever ways to estimate probabilities for things we have rarely or never seen



Smoothing

- When estimating a language model, we're relying on the data we've observed in a training corpus.
- Training data is a small (and biased) sample of the creativity of language.

$$\prod_{i}^{n} P(w_i \mid w_{i-1}) \times P(\text{STOP} \mid w_n)$$

- As in Naive Bayes, $P(w_i) = 0$ causes P(w) = 0. (Perplexity?)

Smoothing in NB

- One solution: add a little probability mass to every element.

maximum likelihood estimate

$$P(x_i \mid y) = \frac{n_{i,y}}{n_y}$$

$n_{i,y}$ = count of word i in class y
n_y = number of words in y
V = size of vocabulary

smoothed estimates

$$P(x_i \mid y) = \frac{n_{i,y} + \alpha}{n_y + V\alpha}$$

same α for all x_i

$$P(x_i \mid y) = \frac{n_{i,y} + \alpha_i}{n_y + \sum_{j=1}^{V} \alpha_j}$$

possibly different α for each x_i

Additive smoothing

Laplace smoothing:
α = 1

$$P(w_i) = \frac{c(w_i) + \alpha}{N + V\alpha}$$

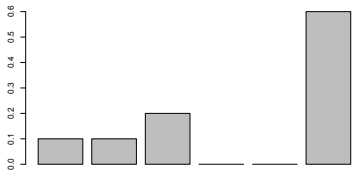
$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + \alpha}{c(w_{i-1}) + V\alpha}$$

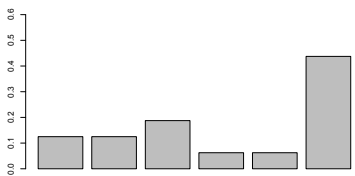
Smoothing

MLE

Smoothing is the re-allocation of probability mass

smoothing with α =1





Board work: Language model with smoothing

Training Data:
<START> I saw Mary <STOP>
<START> I saw John <STOP>
<START> Kat <STOP>
Test Sentence:
<START> I saw Kat <STOP>

- (3 min) Calculate the probability of the test sentence using a bigram language model **with smoothing**
- (3 min) Discuss with a partner

Interpolation

- As ngram order rises, we have the potential for higher precision but also higher variability in our estimates.
- A linear interpolation of any two language models p and q (with $\lambda \in [0,1]$) is also a valid language model.

$$\lambda p + (1 - \lambda) q$$

p = the web

q = political speeches

Interpolation

- We can use this fact to make higher-order language models more robust.

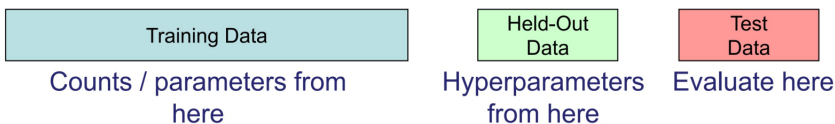
$$
\begin{aligned}
P(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) \\
+ \lambda_2 P(w_i \mid w_{i-1}) \\
+ \lambda_3 P(w_i)
\end{aligned}
$$

$$
\lambda_1 + \lambda_2 + \lambda_3 = 1
$$

Interpolation

- How do we pick the best values of λ?
  - Grid search over development corpus
  - Expectation-Maximization algorithm (treat as missing parameters to be estimated to maximize the probability of the data we see).

Picking Hyperparameters



Train different hyperparameter settings on train set. (Easiest to try random values within a bounding box.)
Evaluate on the dev set, and pick the best performing settings.
Finally, evaluate on the test set.

Evaluating Language Models

How To Evaluate

- Extrinsic: build a new language model, use it for some task (MT, ASR, etc.)
- Intrinsic: measure how good we are at modeling language

Intrinsic Evaluation

- Intuitively, language models should assign high probability to real language they have not seen before
  - Want to maximize likelihood on test, not training data
  - Models derived from counts / sufficient statistics require generalization parameters to be tuned on held-out data to stimulate test generalization
  - Set hyperparameters to maximize the likelihood of the held-out data (usually with grid search or EM)

Evaluation: Perplexity

Intuitively, language models should assign high probability to real language they have not seen before.

For test data $\mathcal{D} = \{x_1, \ldots x_N\}$:

- Probability of $\mathcal{D}$ is $\prod_{i=1}^{N} p(x_i)$

Evaluation: Perplexity

Intuitively, language models should assign high probability to real language they have not seen before.

For test data $\mathcal{D} = \{x_1, \ldots x_N\}$:

- Probability of $\mathcal{D}$ is $\prod_{i=1}^{N} p(x_i)$
- Log-probability of $\mathcal{D}$ is $\sum_{i=1}^{m} \log_2 p(x_i)$

Evaluation: Perplexity

Intuitively, language models should assign high probability to real language they have not seen before.

For test data $\mathcal{D} = \{x_1, \ldots x_N\}$:

- Probability of $\mathcal{D}$ is $\prod_{i=1}^{N} p(x_i)$
- Log-probability of $\mathcal{D}$ is $\sum_{i=1}^{m} \log_2 p(x_i)$
- Average log-probability per word of $\mathcal{D}$ is

$$l = \frac{1}{M} \sum_{i=1}^{N} \log_2 p(\bar{\boldsymbol{x}}_i)$$

if $M = \sum_{i=1}^{N} |x_i|$ (total number of words in the corpus)

Evaluation: Perplexity

Intuitively, language models should assign high probability to real language they have not seen before.

For test data $\mathcal{D} = \{x_1, \ldots x_N\}$:

- Probability of $\mathcal{D}$ is $\prod_{i=1}^{N} p(x_i)$
- Log-probability of $\mathcal{D}$ is $\sum_{i=1}^{m} \log_2 p(x_i)$
- Average log-probability per word of $\mathcal{D}$ is

$$l = \frac{1}{M} \sum_{i=1}^{N} \log_2 p(\bar{\boldsymbol{x}}_i)$$

  if $M = \sum_{i=1}^{N} |x_i|$ (total number of words in the corpus)
- Perplexity on dataset $\mathcal{D}$ is 2^{-l}

Evaluation: Perplexity

Intuitively, language models should assign high probability to real language they have not seen before.

For test data $\mathcal{D} = \{x_1, \ldots x_N\}$:

- Probability of $\mathcal{D}$ is $\prod_{i=1}^{N} p(x_i)$
- Log-probability of $\mathcal{D}$ is $\sum_{i=1}^{m} \log_2 p(x_i)$
- Average log-probability per word of $\mathcal{D}$ is

$$l = \frac{1}{M} \sum_{i=1}^{N} \log_2 p(\bar{\boldsymbol{x}}_i)$$

  if $M = \sum_{i=1}^{N} |x_i|$ (total number of words in the corpus)
- Perplexity on dataset $\mathcal{D}$ is 2^{-l}

Lower is better.

Understanding Perplexity

$$2^{-\frac{1}{M}\sum_{i=1}^{m}\log_2 p(x_i)}$$

It's a branching factor!

- Assign probability of 1 to the test data $\Rightarrow$ perplexity $= 1$
- Assign probability of $\frac{1}{|V|}$ to every word $\Rightarrow$ perplexity $= |V|$
- Assign probability of 0 to *anything* $\Rightarrow$ perplexity $= \infty$
  - This motivates a stricter constraint than we had before:
    - For any $x \in V^*$, $p(x) > 0$