CS 679: Natural Language Processing

Lectures #4: Language Model
Smoothing, Interpolation

Thanks to Dan Klein of UC Berkeley for many of the materials used in this lecture.

Announcements

- Project #1
  - Build an interpolated language model
  - Due: Tuesday

- Reading Report #3
  - M&S 6.3-end
  - Due: Thursday

Recap: Language Models

- Why are language models useful?
- Why did I show examples of generated text?
- What do you think are the main challenges in building n-gram language models?

Objectives

- Get comfortable with the process of factoring and smoothing a joint model of a familiar object: text!
- Motivate smoothing of language models
- Dig into Linear Interpolation as a method for smoothing
- Discuss how to train interpolation weights

Markov Chains

- Andrei Andreyevich Markov
  - 1856-1922

Factoring Language Models

- Recall our Bigram model:

\[ P(w_1 \ldots w_n) = \prod P(w_i | w_{i-1}) \]
Local Models

- Recall our Bigram model:

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

Regular Languages?

- N-gram models are (weighted) regular processes
- Why can’t we model language like this?
  - Long-distance dependencies: “The computer which I had just put into the machine room on the fifth floor crashed.”
  - Hierarchical structure among phrases
- Why CAN we often get away with n-gram models?
  - 74% of dependencies in the Penn treebank are between adjacent words!
  - 95% have no more than 4 words intervening

Problem

\[ P(w_1, w_2, \ldots, w_n) = 0 \]

MLE

- How should we estimate the local model?
- Starting point: Maximum Likelihood Estimator
  - a.k.a. Empirical Estimator
  \[ P_{\text{ML}}(w|h) = \frac{c(w, h)}{c(h)} \]
  - For a bigram model: \( h = \text{previous word} \)

Is This Working?

- The game isn’t to pound out fake sentences!
- What we really want to know is:
  - Will our model prefer good sentences to bad ones?
  - Bad ≠ ungrammatical!
  - Bad = unlikely
  - Bad = sentences that our acoustic model really likes but aren’t the correct answer (SR)
  - Bad = sentences that our alignment model really likes but aren’t the correct answer (MT)

Cause: Sparsity

- New words appear all the time:
  - Synaptitute
  - 132,701.03
  - fuzzificational
- New bigrams: even more often
- Trigrams or larger – still worse!
- What was the point of Zipf’s law for us?
- What will we do about it?
Solution: Smoothing

- We often want to make predictions from sparse statistics:

- Smoothing flattens distributions so they generalize better

Two approaches we will explore:
- Interpolation: combine multiple estimates to give probability mass to unseen events
  - Think: Two heads are better than one!
  - Today's lecture
- Discounting: explicitly reserve mass for unseen events
  - Think: Robin Hood – rob from the rich to feed the poor!

Another approach you read about:
- Back-off: we won’t spend time on this

Interpolation

- Idea: two heads are better than one
  - i.e., combine (less sparse) lower-order model with higher-order model to get a more robust higher-order model

$$P'(w_l | w_{l-1}) = f\left(P_k(w_l | w_{l-1}), P(w_l), \frac{1}{|W|}\right)$$

Convex, Linear Interpolation

- Convex:
  - interpolation constants sum to 1
  - $$\forall i, 0 \leq \lambda_i \leq 1$$

- One extreme: General linear interpolation

$$P(w | h_{k-1}) = \lambda \hat{P}(w | h_{k-1}) + (1 - \lambda) P(w | h_{k-1})$$

- One interpolation coefficient per history

Linear Interpolation

- The other extreme: a single global mixing weight
  - Generally not ideal but it works:

$$\hat{P}(w | h_{k-1}) = (1 - \lambda) \hat{P}(w | h_{k-1}) + \lambda P(w | h_{k-1})$$

- Middle ground: different weights for classes of histories defined at other granularities:
  - Bucket histories (and their weights) by count $$k$$:
    - for each bucket, use a single weight $$\lambda(k)$$
  - Bucket histories by average count (better):
    - for a range of buckets $$\text{bucket}_{i-2} \ldots \text{bucket}_{i-4}$$, have a single weight

Example: Linear Interpolation

| history (h) | w  | $$P_1(w|h)$$ | $$P_2(w|h)$$ | $$P_3(w|h)$$ | $$P_4(w|h)$$ | interpolated |
|------------|----|-------------|-------------|-------------|-------------|-------------|
| fail into  | the| 0.30        | 0.5         | 0.030       |             |             |
| a          |    | 0.10        | 0.2         | 0.010       |             |             |
| two        |    | 0.00        | 0.0         | 0.001       |             |             |
| <other>    |    | 0.6         | 0.3         |             |             | 0.959       |
| <UNK>      |    |             |             |             |             |             |

Question: using the following weights
- $$\lambda_{\text{fall into}} = 0.1$$
- $$\lambda_{\text{fail into}} = 0.5$$
- $$\lambda_{\text{fail into}} = 0.4$$

How do you compute the combined, interpolated probabilities?
Example: Linear Interpolation

| history | w | \( P_3(w|h) \) | \( P_2(w|h) \) | \( P_1(w|h) \) | Interpolated |
|---------|---|-------------|-------------|-------------|-------------|
| fall into | the | 0.30 | 0.5 | 0.030 | 0.272 |
| a | | 0.10 | 0.2 | 0.010 |
| two | | 0.00 | 0.0 | 0.001 |
| <other> | <OOV> | 0.6 | 0.3 | 0.959 |

\[
\hat{p}(w|h) = \lambda_1 P_1(w|h) + \lambda_2 P_2(w|h) + \lambda_3 P_3(w|h)
\]

Question: using the following weights
- \( \lambda_1 \), "fall into" = 0.1
- \( \lambda_2 \), "fall into" = 0.5
- \( \lambda_3 \), "fall into" = 0.4

How do you compute the combined, interpolated probabilities?

Learning the Weights

- How?

\[
\hat{p}(w|w_{-1}) = \lambda_1 p(w_1|w_{-1}) + \lambda_2 p(w_2|w_{-1}) + \lambda_3 \hat{p}(w)
\]

Tuning on Held-Out Data

- Important tool for getting models to generalize:

```
Training Data  Held-Out Data  Test Data
```

\[
P_{\text{LIN}}(\hat{a} | w_{-1}) = (1 - \lambda) \hat{p}(w | w_{-1}) + \lambda \hat{p}(w)
\]

Wisdom

- "A cardinal sin in Statistical NLP is to test on your training data."
  - Manning & Schuetze, p. 206
- Corollary: "You should always eyeball the training data – you want to use your human pattern-finding abilities to get hints on how to proceed. You shouldn’t eyeball the test data – that’s cheating …"
  - M&S, p. 207

Likelihood of the Data

- We want the joint probability of a data set \( w \) (containing \( S \) sentences):
  \[
P(w) = \prod_s P(w_s)
\]
- Use our model \( M(w) \) (local model \( \hat{L} \)), trained on the training set:
  \[
P_{\hat{L}}(w) = \prod_s \prod_h \hat{p}_h(w_s|m_{s-1})
\]
- Take the logarithm: (why?)
  \[
  \log p(w) = \log \prod_s \prod_h \hat{p}_h(w_s|m_{s-1})
  \]
- Distribute the log: Inward:
  \[
  \log P_{\hat{L}}(w) = \sum_s \sum_h \log \hat{p}_h(w_s|m_{s-1})
  \]
- Compare models using the Log likelihood function:
  \[
  \mathcal{L}(w|\theta) = \sum_s \sum_h \log \hat{p}_h(w_s|m_{s-1})
  \]

Maximizing the Likelihood

- Situation: we have a small number of parameters \( \lambda_1 \ldots \lambda_k \) that control the degree of smoothing in our model
- Goal: set them to maximize the (log-)likelihood of held-out data:
  \[
  \mathcal{L}_{\text{held-out}}(M(\lambda_1 \ldots \lambda_k)) = \sum_s \sum_h \log \hat{p}_h(w_s|m_{s-1})
  \]
  \[\lambda_1 \ldots \lambda_k = \arg \max_{\lambda_1 \ldots \lambda_k} \mathcal{L}_{\text{held-out}}(M(\lambda_1 \ldots \lambda_k))\]
- Method: use any optimization technique
  - line search – easy, OK
  - EM (to be discussed later in this course)
**Tuning on Held-Out Data**

- Important tool for getting models to generalize:
  - Training Data
  - Held-Out Data
  - Test Data

\[
P_{LIN(\lambda)}(w | w_{-1}) = (1 - \lambda) \hat{P}(w | w_{-1}) + \lambda \hat{P}(w)
\]

**What’s Next**

- Upcoming lectures:
  - Metrics
  - Discounting strategies
  - Reserving mass for Unknown Word (UNK) and Unseen n-grams
    - i.e., “Open Vocabulary”