SFU NatLangLab

Natural Language Processing

Angel Xuan Chang
angelxuanchang.github.io/nlp-class
adapted from lecture slides from Anoop Sarkar

Simon Fraser University

January 16th, 2020

Natural Language Processing

Angel Xuan Chang
angelxuanchang.github.io/nlp-class
adapted from lecture slides from Anoop Sarkar

Simon Fraser University

January 16th, 2020

Part 1: Probability models of Language

The Language Modeling problem

Setup

- Assume a (finite) vocabulary of words:
$$\mathcal{V} = \{\textit{killer}, \textit{crazy}, \textit{clown}\}$$
- Use $\mathcal{V}$ to construct an infinite set of *sentences*
$$\begin{aligned}\mathcal{V}^+ = \ \{ & \\ & \text{clown, killer clown, crazy clown,} \\ & \text{crazy killer clown, killer crazy clown,} \\ & \dots \\ \} & \end{aligned}$$
- A *sentence* is **defined** as each $s \in \mathcal{V}^+$

The Language Modeling problem

Data

Given a training data set of example sentences $s \in \mathcal{V}^+$

Language Modeling problem

Estimate a probability model:

$$\sum_{s \in \mathcal{V}^+} p(s) = 1.0$$

- p(clown) = 1e-5
- p(killer) = 1e-6
- p(killer clown) = 1e-12
- p(crazy killer clown) = 1e-21
- p(crazy killer clown killer) = 1e-110
- p(crazy clown killer killer) = 1e-127

Why do we want to do this?

Scoring Hypotheses in Speech Recognition

From acoustic signal to candidate transcriptions

Hypothesis	Score
the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

Scoring Hypotheses in Machine Translation

From source language to target language candidates

Hypothesis	Score
we must also discuss a vision .	-29.63
we must also discuss on a vision .	-31.58
it is also discuss a vision .	-31.96
we must discuss on greater vision .	-36.09
⋮	⋮

Scoring Hypotheses in Decryption

Character substitutions on ciphertext to plaintext candidates

Hypothesis	Score
Heopaj, zk ukq swjp pk gjks w oaynap?	-93
Urbcnw, mx hxd fjwc cx twxf j bnlanc?	-92
Wtdepy, oz jzf hlye ez vyzh l dpncpe?	-91
Mjtufo, ep zpv xbou up lopx b tfdsfu?	-89
Nkuvgp, fq aqw ycpv vq mpqy c ugetgv?	-87
Gdnozi, yj tjp rvio oj fijr v nzxmzo?	-86
Czjkve, uf pfl nrek kf befn r jvtivk?	-85
Yvfgra, qb lbh jnag gb xabj n frperg?	-84
Zwghsb, rc mci kobh hc ybck o gsqfsh?	-83
Byijud, te oek mqdj je adem q iushuj?	-77
Jgqrcl, bm wms uylr rm ilmu y qcapcr?	-76
Listen, do you want to know a secret?	-25

Scoring Hypotheses in Spelling Correction

Substitute spelling variants to generate hypotheses

Hypothesis	Score
… stellar and versatile **acress** whose combination of sass and glamour has defined her …	-18920
… stellar and versatile **acres** whose combination of sass and glamour has defined her …	-10209
… stellar and versatile **actress** whose combination of sass and glamour has defined her …	-9801

T9 to English

Grover, King, & Kushler. 1998.
Reduced keyboard disambiguating computer. US Patent 5,818,437

Sequence of numbers to English

Input	Hypothesis	Score
46 04663	GO HOOD	-24
46 04663	GO HOME	-10
843 0746453 06678 07678527 0243373 0460843 096753	?	?

Probability models of language

Question

- Given a finite vocabulary set $\mathcal{V}$
- We want to build a probability model $P(s)$ for all $s \in \mathcal{V}^+$
- **But** we want to consider sentences s of each length ℓ separately.
- Write down a new model over $\mathcal{V}^+$ such that $P(s \mid \ell)$ is in the model
- **And** the model should be equal to $\sum_{s \in \mathcal{V}^+} P(s)$.
- Write down the model

$$\sum_{s \in \mathcal{V}^+} P(s) = \ldots$$

Natural Language Processing

Angel Xuan Chang
angelxuanchang.github.io/nlp-class
adapted from lecture slides from Anoop Sarkar

Simon Fraser University

January 16th, 2020

Part 2: *n*-grams for Language Modeling

Language models

n-grams for Language Modeling

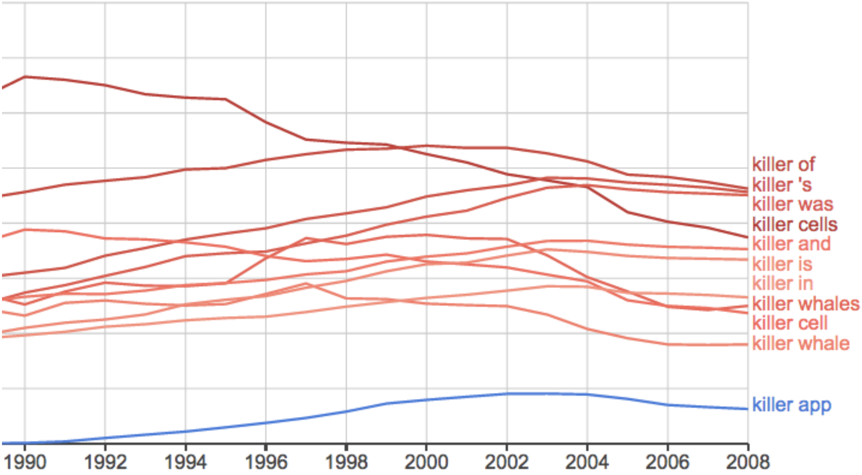
Smoothing n-gram Models
- Handling Unknown Tokens
- Smoothing Counts
  - Add-one Smoothing
  - Additive Smoothing
- Interpolation: Jelinek-Mercer Smoothing
- Backoff Smoothing with Discounting

Evaluating Language Models

Event Space for n-gram Models

n-gram Models

Google n-gram viewer



Learning Language Models

- Directly count using a training data set of sentences: $w_1, \ldots, w_n$:

$$p(w_1, \ldots, w_n) = \frac{c(w_1, \ldots, w_n)}{N}$$

- c is a function that counts how many times each sentence occurs
- N is the sum over all possible $c(\cdot)$ values
- Problem: does not generalize to new sentences unseen in the training data.
- What are the chances you will see a sentence: `crazy killer clown crazy killer`?
- In NLP applications we often need to assign non-zero probability to previously unseen sentences.

Learning Language Models

Apply the Chain Rule: the unigram model

$$\begin{aligned} p(w_1, \ldots, w_n) &\approx p(w_1)p(w_2)\ldots p(w_n) \\ &= \prod_i p(w_i) \end{aligned}$$

Big problem with a unigram language model

p(the the the the the the the) $>$ p(we must also discuss a vision .)

Learning Language Models

Apply the Chain Rule: the bigram model

$$
\begin{aligned}
p(w_1, \ldots, w_n) &\approx p(w_1)p(w_2 \mid w_1) \ldots p(w_n \mid w_{n-1}) \\
&= p(w_1) \prod_{i=2}^{n} p(w_i \mid w_{i-1})
\end{aligned}
$$

Better than unigram

p(the the the the the the the) $<$ p(we must also discuss a vision .)

Learning Language Models

Apply the Chain Rule: the trigram model

$$
\begin{aligned}
p(w_1, \ldots, w_n) \approx \\
& p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) \ldots p(w_n \mid w_{n-2}, w_{n-1}) \\
& p(w_1)p(w_2 \mid w_1) \prod_{i=3}^{n} p(w_i \mid w_{i-2}, w_{i-1})
\end{aligned}
$$

Better than bigram, but ...

p(we must also discuss a vision .) might be zero because we have not seen p(discuss | must also)

Maximum Likelihood Estimate

Using training data to learn a trigram model

- Let $c(u, v, w)$ be the count of the trigram u, v, w, e.g. $c(crazy, killer, clown)$. $P(u, v, w) = \frac{c(u,v,w)}{\sum_{u,v,w} c(u,v,w)}$
- Let $c(u, v)$ be the count of the bigram u, v, e.g. $c(crazy, killer)$. $P(u, v) = \frac{c(u,v)}{\sum_{u,v} c(u,v)}$
- For any u, v, w we can compute the conditional probability of generating w given u, v:

$$p(w \mid u, v) = \frac{c(u, v, w)}{c(u, v)}$$

- For example:

$$p(clown \mid crazy, killer) = \frac{c(crazy, killer, clown)}{c(crazy, killer)}$$

Number of Parameters

How many probabilities in each n-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many unigram probabilities: $P(x)$ for $x \in \mathcal{V}$?

4

Number of Parameters

How many probabilities in each n-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many bigram probabilities: $P(y|x)$ for $x, y \in \mathcal{V}$?

$4^2 = 16$

Number of Parameters

How many probabilities in each *n*-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many trigram probabilities: $P(z|x, y)$ for $x, y, z \in \mathcal{V}$?

$4^3 = 64$

Number of Parameters

Question

- Assume $|\mathcal{V}| = 50{,}000$ (a realistic vocabulary size for English)
- What is the minimum size of training data in tokens?
  - If you wanted to observe all unigrams at least once.
  - If you wanted to observe all trigrams at least once.

125,000,000,000,000 (125 Ttokens)

Some trigrams should be zero since they do not occur in the language, $P(\textit{the} \mid \textit{the}, \textit{the})$.
But others are simply unobserved in the training data, $P(\textit{idea} \mid \textit{colourless}, \textit{green})$.

Summary: Probabilistic Language Modeling

- Language modeling: Predict probability for sequence of words
- Useful for speech recognition, machine translation, spelling correction, etc.
- To compute the probability of a sequence of words
  - $P(W) = P(w_1, w_2, w_3, \ldots, w_n) = \prod_i P(w_i | w_1, \ldots w_{i-1})$
  - $P(\text{it is very cold today}) =$
    $P(\text{it})P(\text{is}|\text{it})P(\text{very}|\text{it is})P(\text{cold}|\text{it is very})P(\text{today}|\text{it is very cold})$
- Learn probabilities from a corpus of text data by counting
  $P(\text{cold}|\text{it is very}) = \frac{\text{Count(it is very cold)}}{\text{Count(it is very)}}$

Summary: Probabilistic Language Modeling

- What about $P(\text{cold}|\text{it is very very very very})$?
- Lots of unseen sequences! What to do?
- Use the Markov assumption
  Assume that $P(w_i)$ depends only on recent history
  Unigram: $P(w_i|w_1, \ldots w_{i-1}) \approx P(w_i)$
  Bigram: $P(w_i|w_1, \ldots w_{i-1}) \approx P(w_i|w_{i-1})$
  Trigram: $P(w_i|w_1, \ldots w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$
- $P(\text{cold}|\text{it is very very very very}) \approx P(\text{cold}|\text{very very}) = \frac{\text{Count(very very cold)}}{\text{Count(very very)}}$

Bigram Models

- In practice:

$$
\begin{aligned}
&P(\texttt{crazy killer clown}) = \\
&\quad P(\texttt{crazy} \mid \texttt{<s>}) \times P(\texttt{killer} \mid \texttt{crazy}) \times \\
&\quad P(\texttt{clown} \mid \texttt{killer}) \times P(\texttt{</s>} \mid \texttt{clown})
\end{aligned}
$$

- $P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$
  On unseen data, $c(w_{i-1}, w_i)$ or worse $c(w_{i-1})$ could be zero

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

Natural Language Processing

Angel Xuan Chang
angelxuanchang.github.io/nlp-class
adapted from lecture slides from Anoop Sarkar

Simon Fraser University

January 16th, 2020

Part 3: Smoothing Probability Models

Language models

n-grams for Language Modeling

Smoothing *n*-gram Models
- Handling Unknown Tokens
- Smoothing Counts
  - Add-one Smoothing
  - Additive Smoothing
- Interpolation: Jelinek-Mercer Smoothing
- Backoff Smoothing with Discounting

Evaluating Language Models

Event Space for *n*-gram Models

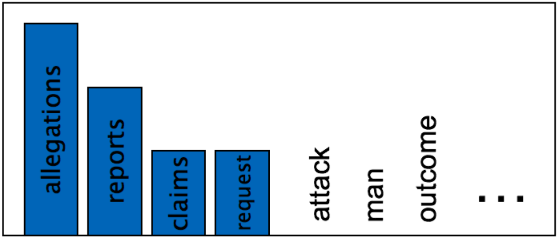
Smoothing Intuition

Taking from the rich and giving to the poor

When we have sparse statistics:

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

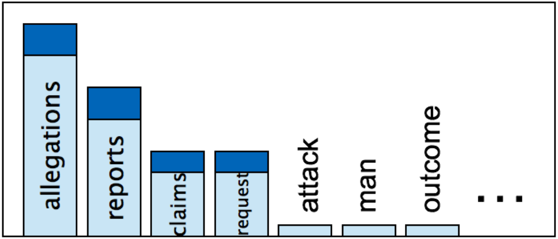
7 total



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



Handling tokens in test corpus unseen in training corpus

Assume closed vocabulary

In some situations we can make this assumption, e.g. our vocabulary is ASCII characters

Interpolate with unknown words distribution

We will call this *smoothing*. We combine the n-gram probability with a distribution over unknown words

$$P_{\mathrm{unk}}(w) = \frac{1}{V_{\mathrm{all}}}$$

V_{all} is an estimate of the vocabulary size including unknown words.

Add an `<unk>` word

Modify the training data L by changing words that appear only once to the `<unk>` token. Since this probability can be an over-estimate we multiply it with a probability $P_{\mathrm{unk}}(\cdot)$.

Smoothing

- **Smoothing** deals with events that have been observed zero times
- Smoothing algorithms also tend to improve the accuracy of the model

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

- Not just unobserved events: what about events observed once?

Add-one Smoothing

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

- Add-one Smoothing (aka Laplace Smoothing):

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

- Let V be the number of words in our vocabulary
  Assign count of 1 to unseen bigrams

Add-one Smoothing

$$P(\texttt{insane killer clown}) =$$
$$P(\texttt{insane} \mid \texttt{<s>}) \times P(\texttt{killer} \mid \texttt{insane}) \times$$
$$P(\texttt{clown} \mid \texttt{killer}) \times P(\texttt{</s>} \mid \texttt{clown})$$

- Without smoothing:

$$P(\texttt{killer} \mid \texttt{insane}) = \frac{c(\texttt{insane, killer})}{c(\texttt{insane})} = 0$$

- With add-one smoothing (assuming initially that `c(insane)` = 1 and `c(insane, killer)` = 0):

$$P(\texttt{killer} \mid \texttt{insane}) = \frac{1}{V+1}$$

Additive Smoothing: (Lidstone 1920, Jeffreys 1948)

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

- Why add 1? 1 is an overestimate for unobserved events.
- Additive Smoothing:

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

- $0 < \delta \leq 1$

Interpolation: Jelinek-Mercer Smoothing

$$P_{ML}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- $P_{JM}(w_i \mid w_{i-1}) = \lambda P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda) P_{ML}(w_i)$
  where, $0 \leq \lambda \leq 1$
- Jelinek and Mercer (1980) describe an elegant form of this **interpolation**:

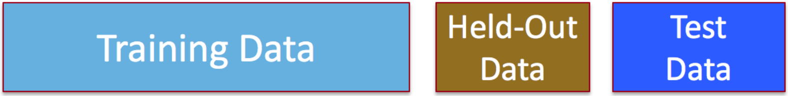
$$P_{JM}(n\text{gram}) = \lambda P_{ML}(n\text{gram}) + (1 - \lambda) P_{JM}(n - 1\text{gram})$$

- What about $P_{JM}(w_i)$?
  For missing unigrams: $P_{JM}(w_i) = \lambda P_{ML}(w_i) + (1 - \lambda) \frac{\delta}{V}$
  $0 < \delta \leq 1$

Interpolation: Finding λ

$$P_{JM}(n\text{gram}) = \lambda P_{ML}(n\text{gram}) + (1-\lambda)P_{JM}(n-1\text{gram})$$

- Deleted Interpolation (Jelinek, Mercer)
  compute λ values to minimize cross-entropy on **held-out** data
  which is deleted from the initial set of training data



- Improved JM smoothing, a separate λ for each w_{i-1}:

$$P_{JM}(w_i \mid w_{i-1}) = \lambda(w_{i-1})P_{ML}(w_i \mid w_{i-1}) + (1-\lambda(w_{i-1}))P_{ML}(w_i)$$

Backoff Smoothing with Discounting

- Absolute Discounting (aka *abs*) (Ney, Essen, Kneser)

$$P_{abs}(y \mid x) = \begin{cases} \frac{c(xy) - D}{c(x)} & \text{if } c(xy) > 0 \\ \alpha(x) P(y) & \text{otherwise} \end{cases}$$

- where $\alpha(x)$ is chosen to make sure that $P_{abs}(y \mid x)$ is a proper probability

$$\alpha(x) = 1 - \sum_y \frac{c(xy) - D}{c(x)}$$

Backoff Smoothing with Discounting

x	$c(x)$	$c(x) - D$	$\frac{c(x)-D}{c(the)}$
the	48		
the,dog	15	14.5	14.5/48
the,woman	11	10.5	10.4/48
the,man	10	9.5	9.5/48
the,park	5	4.5	4.5/48
the,job	2	1.5	1.5/48
the,telescope	1	0.5	0.5/48
the,manual	1	0.5	0.5/48
the,afternoon	1	0.5	0.5/48
the,country	1	0.5	0.5/48
the,street	1	0.5	0.5/48
TOTAL			0.8958
the,UNK	0		0.1042

Web-scale N-grams Smoothing

- "Stupid backoff" (Brants et al, 2007)

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Takeaways

- Predict probability of sequence of words
- Need to handle data sparsity
  use Markov assumption and smoothing
- Later: Neural language models

Use Chain rule and approximate using a neural network

$$p(w_1, \ldots, w_n) \approx \prod_t p(w_{t+1} \mid \underbrace{\phi(w_1, \ldots, w_t)}_{\text{capture history with vector } s(t)})$$

Natural Language Processing

Angel Xuan Chang
angelxuanchang.github.io/nlp-class
adapted from lecture slides from Anoop Sarkar

Simon Fraser University

January 16th, 2020

Part 4: Evaluating Language Models

Language models

n-grams for Language Modeling

Smoothing *n*-gram Models
Handling Unknown Tokens
Smoothing Counts
Add-one Smoothing
Additive Smoothing
Interpolation: Jelinek-Mercer Smoothing
Backoff Smoothing with Discounting

Evaluating Language Models

Event Space for *n*-gram Models

Evaluating Language Models

- So far we've seen the probability of a sentence: $P(w_0, \ldots, w_n)$
- What is the probability of a collection of sentences, that is what is the probability of an unseen test corpus T
- Let $T = s_0, \ldots, s_m$ be a test corpus with sentences s_i
- T is assumed to be separate from the training data used to train our language model $P(s)$
- What is $P(T)$?

Evaluating Language Models: Independence assumption

- $T = s_0, \ldots, s_m$ is the text corpus with sentences s_0 through s_m
- $P(T) = P(s_0, s_1, s_2, \ldots, s_m)$ – but each sentence is independent from the other sentences
- $P(T) = P(s_0) \cdot P(s_1) \cdot P(s_2) \cdot \ldots \cdot P(s_m) = \prod_{i=0}^{m} P(s_i)$
- $P(s_i) = P(w_0^{(i)}, \ldots, w_{n_i}^{(i)})$ – which can be any n-gram language model
- A language model is better if the value of $P(T)$ is higher for unseen sentences T, we want to maximize:

$$P(T) = \prod_{i=0}^{m} P(s_i)$$

Evaluating Language Models: Computing the Average

- However, T can be any arbitrary size
- $P(T)$ will be lower if T is larger.
- Instead of the probability for a given T we can compute the *average* probability.
- M is the total number of tokens in the test corpus T:

$$M = \sum_{i=0}^{m} \text{length}(s_i)$$

- The average *log* probability of the test corpus T is:

$$\frac{1}{M} \log_2 \prod_{i=0}^{m} P(s_i) = \frac{1}{M} \sum_{i=0}^{m} \log_2 P(s_i)$$

Evaluating Language Models: Perplexity

- The average *log* probability of the test corpus T is:

$$\ell = \frac{1}{M} \sum_{i=0}^{m} \log_2 P(s_i)$$

- Note that ℓ is a negative number
- We evaluate a language model using *Perplexity* which is $2^{-\ell}$

Evaluating Language Models

Question

Show that:

$$2^{-\frac{1}{M}\log_2 \prod_{i=0}^{m} P(s_i)} = \frac{1}{\sqrt[M]{\prod_{i=0}^{m} P(s_i)}}$$

Evaluating Language Models

Question

What happens to $2^{-\ell}$ if any n-gram probability for computing $P(T)$ is zero?

Evaluating Language Models: Typical Perplexity Values

From 'A Bit of Progress in Language Modeling' by Chen and Goodman

Model	Perplexity
unigram	955
bigram	137
trigram	74

Evaluating Language Models: Typical Perplexity Values

From 'One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling' by Chelba+ (Google)

Model	**Num. Params [billions]**	**Training Time [hours]**	**Training Time [CPUs]**	**Perplexity**
Interpolated KN 5-gram, 1.1B n-grams (KN)	1.76	3	100	67.6
Katz 5-gram, 1.1B n-grams	1.74	2	100	79.9
Stupid Backoff 5-gram (SBO)	1.13	0.4	200	87.9
Interpolated KN 5-gram, 15M n-grams	0.03	3	100	243.2
Katz 5-gram, 15M n-grams	0.03	2	100	127.5
Binary MaxEnt 5-gram (n-gram features)	1.13	1	5000	115.4
Binary MaxEnt 5-gram (n-gram + skip-1 features)	1.8	1.25	5000	107.1
Hierarchical Softmax MaxEnt 4-gram (HME)	6	3	1	101.3
Recurrent NN-256 + MaxEnt 9-gram	20	60	24	58.3
Recurrent NN-512 + MaxEnt 9-gram	20	120	24	54.5
Recurrent NN-1024 + MaxEnt 9-gram	20	240	24	51.3

Natural Language Processing

Angel Xuan Chang
angelxuanchang.github.io/nlp-class
adapted from lecture slides from Anoop Sarkar

Simon Fraser University

January 16th, 2020

Part 5: Event space in Language Models

Trigram Models

- The trigram model:
  $P(w_1, w_2, \ldots, w_n) =$
  $P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times P(w_4 \mid w_2, w_3) \times$
  $\ldots P(w_i \mid w_{i-2}, w_{i-1}) \ldots \times P(w_n \mid w_{n-2}, \ldots, w_{n-1})$
- Notice that the length of the sentence n is variable
- What is the event space?

The stop symbol

- Let $\mathcal{V} = \{a, b\}$ and the language L be $\mathcal{V}^*$
- Consider a unigram model: $P(a) = P(b) = 0.5$
- So strings in this language L are:

$$\begin{array}{rl} a \text{ stop} & 0.5 \\ b \text{ stop} & 0.5 \\ aa \text{ stop} & 0.5^2 \\ bb \text{ stop} & 0.5^2 \\ & \vdots \end{array}$$

- The sum over all strings in L should be equal to 1:

$$\sum_{w \in L} P(w) = 1$$

- But $P(a) + P(b) + P(aa) + P(bb) = 1.5$!!

The `stop` symbol

- What went wrong?
  We need to model variable length sequences
- Add an explicit probability for the stopsymbol:

$$P(a) = P(b) = 0.25$$

$$P(\text{stop}) = 0.5$$

- $P(\text{stop}) = 0.5$, $P(a \text{ stop}) = P(b \text{ stop}) = 0.25 \times 0.5 = 0.125$, $P(aa \text{ stop}) = 0.25^2 \times 0.5 = 0.03125$ (now the sum is no longer greater than one)

The stop symbol

- With this new stop symbol we can show that $\sum_w P(w) = 1$
  Notice that the probability of any sequence of length n is $0.25^n \times 0.5$
  Also there are 2^n sequences of length n

$$\sum_w P(w) =$$

$$\sum_{n=0}^{\infty} 2^n \times 0.25^n \times 0.5$$

$$\sum_{n=0}^{\infty} 0.5^n \times 0.5 = \sum_{n=0}^{\infty} 0.5^{n+1}$$

$$\sum_{n=1}^{\infty} 0.5^n = 1$$

The $\texttt{stop}$ symbol

- With this new $\texttt{stop}$ symbol we can show that $\sum_w P(w) = 1$
  Using $p_s = P(\texttt{stop})$ the probability of any sequence of length n is $p(n) = p(w_1, \ldots, w_{n-1}) \times p_s(w_n)$

$$\begin{aligned}
\sum_w P(w) &= \sum_{n=0}^{\infty} p(n) \sum_{w_1,\ldots,w_n} p(w_1, \ldots, w_n) \\
&= \sum_{n=0}^{\infty} p(n) \sum_{w_1,\ldots,w_n} \prod_{i=0}^{n} p(w_i)
\end{aligned}$$

$$\sum_{w_1,\ldots,w_n} \prod_i p(w_i) =$$

$$\sum_{w_1} \sum_{w_2} \ldots \sum_{w_n} p(w_1)p(w_2)\ldots p(w_n) = 1$$

The `stop` symbol

$$\sum_{w_1}\sum_{w_2}\dots\sum_{w_n} p(w_1)p(w_2)\dots p(w_n) \quad = \quad 1$$

$$\begin{aligned}
\sum_{n=0}^{\infty} p(n) &= \sum_{n=0}^{\infty} p_s(1-p_s)^n \\
&= p_s \sum_{n=0}^{\infty}(1-p_s)^n \\
&= p_s \frac{1}{1-(1-p_s)} = p_s \frac{1}{p_s} = 1
\end{aligned}$$

Acknowledgements

Many slides borrowed or inspired from lecture notes by Anoop Sarkar, Dan Jurafsky, Michael Collins, Chris Dyer, Kevin Knight, Chris Manning, Philipp Koehn, Adam Lopez, Graham Neubig, Richard Socher and Luke Zettlemoyer from their NLP course materials.

All mistakes are my own.

A big thank you to all the students who read through these notes and helped me improve them.