Basic statistical language modeling

CS 759/859 Natural Language Processing Lecture 11

Samuel Carton, University of New Hampshire

Last lecture

Intermediate representations

Word vector models

- Word2Vec
    - CBOW
    - Skip-gram
- GloVe

Word vectors in classification

- Padding
- Collation
- Centroids

Probability review

Random variables

A random variable X can take different values depending on chance

Notation:

- p(X = x) is the probability that r.v. X takes value x
  - p(x) is shorthand for the same
- p(X) is the distribution over values X can take (a function)

Example: flipping a coin; P(X = heads) = P(X=tails) = 0.5

Discrete distributions

A discrete distribution enumerates the values a random variable can take and how likely each one is

Examples:

p(flipping a coin) = [0.5, 0.5]

p(rolling a die) = [.167, .167, .167, .167, .167, .167]

p(flipping a rigged coin) = [0.25, 0.75]

p(rolling a weighted die) = [0.1, 0 .1, 0.1,0 .1, 0.1, 0.5]

How does **entropy** relate to the values in the discrete distribution?

Joint probability and product rule

The **joint probability** of two random variables X and Y describes the total chance they take on a particular pair of values: $p(X = x, Y = y)$

If X and Y are **independent**, then $p(X = x, Y = y) = p(X = x) * p(Y = y)$

Example: two coin flips X and Y. $p(X = \text{heads}, Y = \text{heads}) = 0.5 * 0.5 = 0.25$

Conditional probability

If X and Y are **dependent**, then you have to think of the probability of X given Y: $p(X = x \mid Y = y)$

In this case, the joint probability of X and Y is $p(Y=y) * p(X = x \mid Y = y)$

Example: Weather is 50% sunny and 50% cloudy; I am 25% likely to run when sunny and 10% likely when rainy.

P(run|sunny) = **.25**

P(run, sunny) = 0.5 * 0.25 = **0.125**

P(run) = P(run, sunny) + P(run, cloudy) = 0.5 * 0.25 + 0.5 * 0.1 = **0.175**

How does **mutual information** relate to dependence versus independence?

Chain rule

If we generalize to N joint random variables, we end up with the **chain rule**

$$
\begin{aligned}
P(X_1, X_2, \ldots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_2, X_1)....P(X_n|X_1, ...X_{n-1}) \\
&= P(X_1)\prod_{i=2}^{n} P(X_i|X_1 \ldots X_{i-1})
\end{aligned}
$$

Conditional probability table

With two variables X and Y, we can summarize their joint distribution with a **conditional probability table**

Each cell is P(X=column | y = row)

Example:

		X	
		Run	Don't run
Y	Sunny	0.25	0.75
	Cloudy	0.1	0.9

Maximum-likelihood probabilistic modeling

Whenever we build a probabilistic model of some phenomenon, we are deciding to fit it within some probabilistic form, and then finding the parameters for the model that make the data **most likely**.

Example:

- Model: the weather is sunny with probability p, and cloudy with probability 1-p
- Data: 10 days; [sunny, cloudy, sunny, sunny, cloudy, sunny, sunny, cloudy, sunny, sunny]
- MLE estimate of p:

Maximum-likelihood probabilistic modeling

Whenever we build a probabilistic model of some phenomenon, we are deciding to fit it within some probabilistic form, and then finding the **most likely** parameters of our model to fit the data.

Example:

- Model: the weather is sunny with probability p, and cloudy with probability 1-p
- Data: 10 days; [sunny, cloudy, sunny, sunny, cloudy, sunny, sunny, cloudy, sunny, sunny]
- MLE estimate of p: 0.7

How did we do that?

Probabilistic NLP terminology

Statistical model (i.e. generative story): A hypothesis about how the data was generated

- E.g. "Every day is either sunny or rainy based on probability *p*, independent of previous day"
- Always an abstraction!

Parameters: the specific numbers associated with the model

- E.g. the specific *p* we choose for sunny vs cloudy
- Often denoted as ϴ

Observations: the real-world data we use to determine the parameters of the model

- E.g. we saw 7 sunny says and 3 cloudy days, so we think *p* is 0.7

Probabilistic NLP terminology

Likelihood: The probability of a given outcome or outcomes given a parameterized model

- E.g. if we think *p* is 0.7, then what is P(weather=cloudy)
- E.g. if we think *p* is 0.7, what is P(weather1=cloudy, weather2=sunny, weather3=sunny…)

Maximum likelihood estimation: Modeling paradigm where we choose θ to maximize likelihood of observed data

- So if we see a particular sequence of 7 sunny days and 3 cloudy days…
  - Likelihood if p=0.7 is: $0.7^7 * 0.3^3 = 0.0022$
  - Likelihood if p=0.5 is: $0.5^{10} = 0.0010$

MLE with conditional probability

Model:

- the weather is sunny with probability p and cloudy with probability 1-p
- I run with probability r_s when it is sunny and probability r_c when it is cloudy

Data: 10 runs

sunny	cloudy	cloudy	sunny	cloudy	sunny	cloudy	sunny	sunny	cloudy
run	no run	no run	no run	no run	no run	no run	run	no run	run

Counts:

	Run	Don't run
Sunny	2	4
Cloudy	1	3

$p =$
$r_s =$
$r_c =$

MLE with conditional probability

Model:

- the weather is sunny with probability p and cloudy with probability 1-p
- I run with probability r_s when it is sunny and probability r_c when it is cloudy

Data: 10 runs

sunny	cloudy	cloudy	sunny	cloudy	sunny	cloudy	sunny	sunny	cloudy
run	no run	no run	no run	no run	no run	no run	run	no run	run

Counts:

	Run	Don't run
Sunny	2	4
Cloudy	1	3

$p = 2+4 / (2+4+1+3) = 6/10 = .6$
$r_s = 2 / (2+4) = 2/6 = 0.33$
$r_c = 1 / (1+3) = 1/4 = 0.25$

Probabilistic language modeling

Unigram model

Basic idea: probability of a given word w depends **only** on its overall frequency within the corpus

- So the probability of a given text is the product of the individual word probabilities

$$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$$

Similar to bag-of-words in that it doesn't respect word order, but bag-of-words isn't explicitly probabilistic

Bigram model

Basic idea: the probability of word i depends **only** on word i-1

Example: “I am” is more likely than “I is”

$$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$$

What can we do with a language model?

Two main things:

1. Generate new text
2. Assess the likelihood of existing text

Manually creating an N-gram language model

Code description

- Use manual token counting to generate a conditional probability table (CPT) representing a bigram language model

Notebook headings

- Manual calculation example
  - Generating a CPT
    - The corpus
    - Preprocessing
    - Counting
    - Counts to probabilities

Generating text

Once we have a MLE estimate model, we can **generate** text by just sampling from our model one word at a time

We can randomly sample, or take the most probable word at each step

We can stop after N tokens, or when we hit some stopping condition (like a [STOP] token, or a ".")

Manually generating text

Code description

- Using our CPT to generate new text sequences

Notebook headings

Generating text

Assessing text likelihood

Given our model, we can calculate the likelihood that a given text was produced by the model.

Example: Bigram model

$$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$$

p("this is a sentence .") = p(this|[START])p(is|this)p(a|is)p(sentence|a)p(.|sentence)

Note that this is just the chain rule of conditional probability in action.

Log-likelihood

Consider trying to actually calculate the likelihood of a sequence:

p("this is a sentence .") = p(this|[START])p(is|this)p(a|is)p(sentence|a)p(.|sentence)

= 0.01 * 0.03 * 0.1 * 0.0001 * 0.003 = 0.000000000009

Taking a product of a bunch of small numbers while quickly become very small and strain the limits of variable precision, leading to underflow.

So, we typically instead calculate the **log-likelihood** of texts, by calculating the **sum of the logs** of the token-token probabilities.

logL("this is a sentence .")

= log(p(this|[START]))+log(p(is|this))+log(p(a|is)p(sentence|a))+log(p(.|sentence))

= log(0.01) + log(0.03) + log(0.1) + log(0.0001) * log(0.003)

= -4.6 + -3.5 + -2.3 + -9.2 + -5.8

= -25.4

Perplexity

Perplexity is a metric for comparing the success of two language models over a given corpus. It consists mostly of calculating the average log-likelihood of the corpus text, for that model.

$$\text{Average NLL} = -\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|w_1, w_2, \ldots, w_{i-1})$$

where N is the total number of words in the test dataset, and $P(w_i|w_1, w_2, \ldots, w_{i-1})$ is the probability of word w_i given the previous words $w_1, w_2, \ldots, w_{i-1}$.

$$\text{Perplexity} = e^{\text{Average NLL}}$$

https://blog.uptrain.ai/decoding-perplexity-and-its-significance-in-llms/

Log-likelihood and perplexity

Code description

- Using our CPT to assess the likelihood of sequences of text, both one token at a time and all together.
- Example of calculating overall perplexity of the model

Notebook headings

Assessing the likelihood of text

One token at a time

Over a whole sequence

Perplexity

Smoothing

Any model bigger than a unigram model suffers from **sparsity** issues that make certain sequences impossible, and screws with all the math

Example: “was delightful” is an impossible bigram in our toy corpus because those two words never happen to occur together, even though they very much could.

```
review_0 = "The film was a delight--I was riveted."
review_1 = "It's the most delightful and riveting movie."
review_2 = "It was a terrible flick, the worst I have ever seen."
review_3 = "I have a feeling the film was recut poorly."

reviews = [review_0, review_1, review_2, review_3]
```

Solution: add some **smoothing** to the model which makes any bigram possible (if not likely)

Smoothing

Code description

- Example of simple Laplace Smoothing over our model

Notebook headings

Smoothing

Case study: Identifying email author

I put together a case study of how you can use a language model, by creating a dataset of 2000 emails written by each of 10 people, drawn from the Enron Email Dataset.

My goal is: given an anonymous email like "you are a huge jerk and I hate you", could I use my language model(s) to identify who was most likely to have written it, and why?

My basic workflow is:

- Train a language model on the whole corpus
- Train a language model on each known individual
- Assess whose personal language model was most likely to have generated the anonymous email
- Compare individual word likelihoods with the global model to try to explain why that person was more likely.

NLTK & Anonymous email case study

Code description

- Downloading and preprocessing Enron Email Dataset
- Showing how to use NLTK to extract N-grams
- Training bigram language models for each distinct author in the corpus
- Using a max-likelihood approach to identify which author wrote a nasty anonymous email

Notebook headings

- NLTK example
  - Read/preprocess Enron dataset
    - Read the Enron emails dataset
    - Preprocess the Enron data
  - Basic NLTK language modeling functionality
  - Build bigram language models for the Enron data
    - Build global model and person-specific models
    - Analyze likelihoods of new text

N-gram models

Unigram model: $$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$$

Bigram model: $$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \,|\, w^{(i-1)})$$

Trigram model: $$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \,|\, w^{(i-1)}, w^{(i-2)})$$

…and so on. But what's the problem? What stops us from conditioning on every previous token?

N-gram models

Unigram model: $$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$$

Bigram model: $$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \,|\, w^{(i-1)})$$

Trigram model: $$P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \,|\, w^{(i-1)}, w^{(i-2)})$$

…and so on. But what's the problem? What stops us from conditioning on every previous token?

- data sparsity
- # of parameters

Concluding thoughts

Probabilistic modeling of text: count word occurrences and normalize to conditional probability distributions

Differing context sizes

- Unigrams: words occur independently
- Bigrams: words depend **only** on previous word
- Trigrams: words depend on previous two words
- etc.

Two important tasks

- Generate new text
- Assess text likelihood under model

Discussion question: how to do classification with these abilities?