Language Modelling: Smoothing and Model Complexity

COMP-599

Sept 14, 2016

Announcements

A1 has been released

- Due on Wednesday, September 28th

Start code for Question 4:

- Includes some of the package import statements that you'll need.
- Includes code to read the files (and deal with annoying Unicode issues)

Outline

Review of last class

Justification of MLE probabilistically

Overfitting and unseen data

Dealing with unseen data: smoothing and regularization

Language Modelling

Predict the next word given some context

Mary had a little _____

- *lamb* GOOD
- *accident* GOOD?
- *very* BAD
- *up* BAD

N-grams

Make a **conditional independence assumption** to make the job of learning the probability distribution easier.

- Context = the previous N-1 words

Common choices: N is between 1 and 3

Unigram model

$$P(w_N|C) = P(w_N)$$

Bigram model

$$P(w_N|C) = P(w_N|w_{N-1})$$

Trigram model

$$P(w_N|C) = P(w_N|w_{N-1}, w_{N-2})$$

Deriving Parameters from Counts

Simplest method: count N-gram frequencies, then divide by the total count

e.g.,

Unigram: P(*cats*) = Count(*cats*) / Count(all words in corpus)

Bigram: P(*cats* | *the*) = Count(*the cats*) / Count(*the*)

Trigram: P(*cats* | *feed the*) = ?

These are the **maximum likelihood estimates** (**MLE**).

Basic Information Theory

Consider some random variable X, distributed according to some probability distribution.

We can define information in terms of how much certainty we gain from knowing the value of X.

Rank the following in terms of how much information we gain by knowing its value:

- Fair coin flip
- An unfair coin flip where we get tails ¾ of the time
- A very unfair coin that always comes up heads

Likely vs Unlikely Outcomes

Observing a likely outcome – less information gained

Intuition: you kinda knew it would happen anyway

- e.g., observing the word *the*

Observing a rare outcome: more information gained!

Intuition: it's a bit surprising to see something unusual!

- e.g., observing the word *armadillo*

Formal definition of information in bits:

$$I(x) = \log_2\left(\frac{1}{P(x)}\right)$$

Minimum number of bits needed to communicate some outcome *x*

Entropy

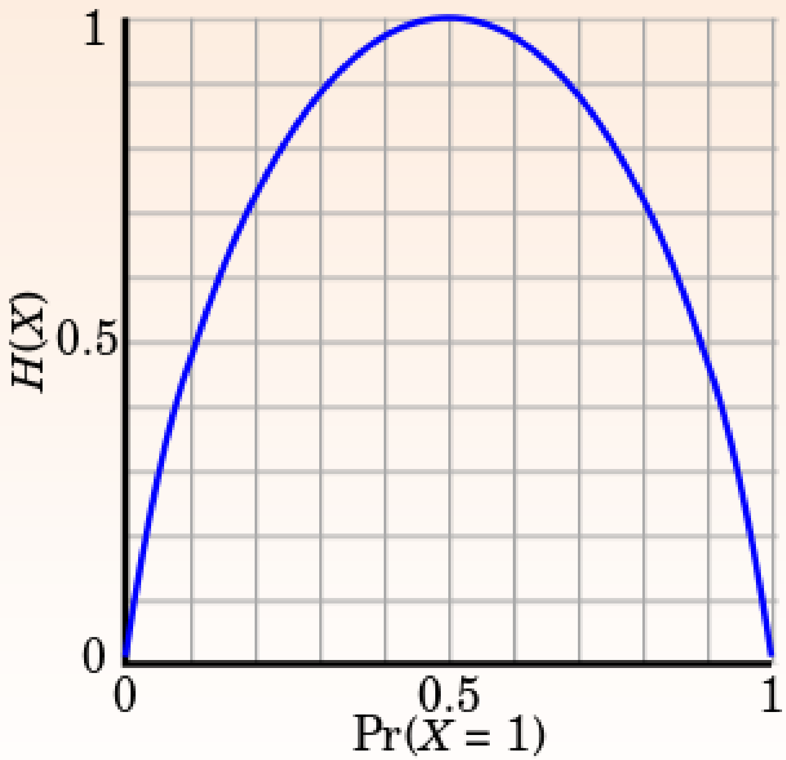
The expected amount of information we get from observing a random variable.

Let a discrete random variable be drawn from distribution *p* take on one of k possible values with probabilities p_1 ... p_k

$$
\begin{aligned}
H(p) &= \sum_{i=1}^{k} p_i I(x_i) \\
&= \sum_{i=1}^{k} p_i \log_2 \frac{1}{p_i} \\
&= -\sum_{i=1}^{k} p_i \log_2 p_i
\end{aligned}
$$

Entropy Example

Plot of entropy vs. coin toss "fairness"



Maximum fairness = maximum entropy

Completely biased = minimum entropy

Image source: Wikipedia, by Brona and Alessio Damato

Cross Entropy

Entropy is the minimum number of bits needed to communicate some message, *if we know what probability distribution the message is drawn from*.

Cross entropy is for when we don't know.

e.g., language is drawn from some true distribution, the language model we train is an approximation of it

$$H(p,q) = -\sum_{i=1}^{k} p_i \log_2 q_i$$

p: "true" distribution

q: model distribution

Estimating Cross Entropy

When evaluating our LM, we assume the test data is a good representative of language drawn from *p*.

Original:

$$H(p,q) = -\sum_{i=1}^{k} p_i \log_2 q_i$$

Estimate:

$$H(p,q) = -\frac{1}{N}\log_2 q(w_1 \ldots w_N)$$

True language distribution, which we don't have access to.

Language model under evaluation

Size of test corpus in number of tokens

The words in the test corpus

Perplexity

Cross entropy gives us a number in bits, which is sometimes hard to read. Perplexity makes this easier.

$$\text{Perplexity}(p, q) \; = \; 2^{H(p,q)}$$

Warm-Up Exercise

Evaluate the given unigram language models using perplexity:

A B C B B

Model 1	**Model 2**
P(A) = 0.3	P(A) = 0.4
P(B) = 0.4	P(B) = 0.5
P(C) = 0.3	P(C) = 0.1

$$\text{Perplexity}(p,q) = 2^{H(p,q)}$$

$$H(p,q) = -\frac{1}{N}\log_2 q(w_1 \ldots w_N)$$

What is Maximum Likelihood?

This way of computing the model parameters corresponds to maximizing the likelihood of (i.e., the probability of generating) the training corpus.

Assumption: words (or N-grams) are random variables that are drawn from a categorical probability distribution i.i.d. (independently, and identically distributed)

Categorical Random Variables

1-of-K discrete outcomes, each with some probability

e.g., coin flip, die roll, draw a word from a language model

Probability of a training corpus, $C = x_1, x_2, \ldots x_N$:

K = 2: $P(C;\theta) = \prod_{n=1}^{N} P(x_n;\theta)$

$$= \theta^{N_1}(1-\theta)^{N_0}$$

Can similarly extend for K > 2

Notes:

- When K=2, it is called a Bernoulli distribution
- Sometimes incorrectly called a multinomial distribution, which is something else

Maximizing Quantities

Calculus to the rescue!

Take derivative and set to 0.

Trick: maximize the log likelihood instead (math works out better)

MLE Derivation for a Bernoulli

Maximize the log likelihood:

$$\log P(C;\theta) = \log(\theta^{N_1}(1 - \theta)^{N_0})$$

$$= N_1 \log\theta + N_0 \log(1 - \theta)$$

$$\frac{d}{d\theta}\log P(C;\theta) = \frac{N_1}{\theta} - \frac{N_0}{1-\theta} = 0$$

$$\frac{N_1}{\theta} = \frac{N_0}{1 - \theta}$$

Solve this to get:

$$\theta = \frac{N_1}{N_0 + N_1}$$

Or,

$$\theta = \frac{N_1}{N}$$

MLE Derivation for a Categorical

The above generalizes to the case where K > 2.

Do the derivation!

Parameters are now $\theta_0, \theta_1, \theta_2, \dots, \theta_{K-1}$

Counts are now $N_0, N_1, N_2, \dots, N_{K-1}$

Note: Need to add a constraint that $\sum_{i=0}^{K-1} \theta_i = 1$ to ensure that the parameters specify a probability distribution.

Use the method of Lagrange multipliers

Steps

1. Gather a large, representative training corpus
2. Learn the parameters from the corpus to build the model
3. **Once the model is fixed, use the model to evaluate on testing data**

Overfitting

MLE often gives us a model that is too good of a fit to the training data. This is called **overfitting**.

- Words that we haven't seen
- The probabilities of the words and N-grams that we have seen are not representative of the true distribution.

But when testing, we evaluate the LM *on unseen data*. Overfitting lowers performance.

Out Of Vocabulary (OOV) Items

Suppose we train a LM on the WSJ corpus, which is about economic news in 1987 – 1989. What probability would be assigned to *Grexit*?

In general, we know that there will be many words in the test data that are not in the training data, no matter how large the training corpus is.

- Neologisms, typos, parts of the text in foreign languages, etc.
- Remember Zipf's law and the long tail

Smoothing

Training corpus does not have all the words

- Add a special UNK symbol for unknown words

Estimates for infrequent words are unreliable

- Modify our probability distributions

Smoothe the probability distributions to shift probability mass to cases that we haven’t seen before or are unsure about

MAP Estimation

Smoothing means we are no longer doing MLE. We now have some **prior belief** about what the parameters should be like: **maximum a posteriori** inference

MLE:

Find θ^{MLE} s.t. $P(X; \theta^{MLE})$ is maximized

MAP:

Find θ^{MAP} s.t. $P(X; \theta^{MAP})P(\theta^{MAP})$ is maximized

Add-δ Smoothing

Modify our estimates by adding a certain amount to the frequency of each word. (sometimes called **pseudocounts**)

e.g., unigram model

$$P(w) = \frac{\text{Count}(w) + \delta}{|Lexicon| * \delta + |Corpus|}$$

- **Pros**: simple
- **Cons**: not the best approach; how to pick δ? Depends on sizes of lexicon and corpus

When δ = 1, this is called **Laplace discounting**

Exercise

Suppose we have a LM with a vocabulary of 20,000 items.

In the training corpus, we see donkey 10 times.

- Of these, in 5 times it was followed by the word kong.
- In the other 5 times, it was followed by another word.

What is the MLE estimate of P(*kong*|*donkey*)?

What is the Laplace estimate of P(*kong*|*donkey*)?

Interpolation

In an N-gram model, as N increases, data sparsity (i.e., unseen or rarely seen events) becomes a bigger problem.

In an **interpolation**, use a lower N to mitigate the problem.

Simple Interpolation

e.g., combine trigram, bigram, unigram models

$$\hat{P}(w_t|w_{t-2}, w_{t-1}) = \lambda_1 P^{MLE}(w_t|w_{t-2}, w_{t-1}) + \lambda_2 P^{MLE}(w_t|w_{t-1}) + \lambda_3 P^{MLE}(w_t)$$

Need to set $\sum_i \lambda_i = 1$ so that the overall sum is a probability distribution

How to select λ_i? We will see shortly…

Good-Turing Smoothing

A more sophisticated method of modelling OOV items

Remember Zipf's lessons

- We shouldn't adjust all words uniformly.
- The frequency of a word type is related to its rank—we should be able to model this!
- Unseen words should behave a lot like words that only occur once in a corpus (**hapax legomenon**; pl., **hapax legomena**)
- Words that occur a lot should behave like other words that occur a lot.

Count of Counts

Let's build a histogram to count how many word-types occur a certain number of times in the corpus.

Word frequency	# word-types with that frequency
1	f_1 = 3993
2	f_2 = 1292
3	f_3 = 664
…	…

- For some word in bin f_c , that word occurred *c* times in the corpus; *c* is the numerator in the MLE.
- **Idea**: re-estimate c using f_{c+1}

Good-Turing Smoothing Defined

Let N be total number of observed word-tokens, w_c be a word that occurs c times in the training corpus.

$$N = \sum_i f_i \times i$$

$$P(UNK) = f_1 / N$$ ← Note that this is for **all** OOV words

Then: $c^* = \frac{(c+1) f_{c+1}}{f_c}$

$$P(w_c) = c^* / N$$ ← Note that this is for **one** word that occurs c times

Example:

Let N be 100,000.

Word frequency	# word-types
1	f_1 = 3,993
2	f_2 = 1,292
3	f_3 = 664
…	…

P(UNK) = 3993 / 100000
= 0.03993
(for all unknown words)

c_1^* = 2 * 1292 / 3993
= 0.647

c_2^* = 3 * 664 / 1292
= 1.542

Good-Turing Refinement

In practice, we need to do something a little more:

At higher values of c, f_{c+1} is often 0.

Solution: Estimate f_c as a function of c

- We'll assume that a linear relationship exists between $\log c$ and $\log f_c$
- Use linear regression to learn this relationship:

  $\log f_c^{LR}$ = a $\log c$ + b
- For lower values of c, we continue to use f_c; for higher values of c , we use our new estimate f_c^{LR}.

Exercises

Suppose we have the following counts:

Word	ship	pass	camp	frock	soccer	mother	tops
Freq	8	7	3	2	1	1	1

Give the MLE and Good-Turing estimates for the probabilities of:

- any unknown word
- *soccer*
- *camp*

Model Selection

We now have very many slightly different versions of the model (with different **hyperparameters**). How to decide between them?

Use a **development / validation set**

Procedure:

1. Train one or more models on the training set
2. Test (repeatedly, if necessary) on the dev/val set; choose a final hyperparameter setting/model
3. Test the final model on the final testing set (once only)

Steps 1 and 2 can be structured using cross-validation

Model Complexity Trade-Offs

In general, there is a trade-off between:

- model expressivity; i.e., what trends you could capture about your data with your model
- how well it generalizes to data

If you use a highly expressive model (e.g., high values of N in N-gram modelling), it is much easier to overfit, and you need to do smoothing. OTOH, if your model is too weak, your performance will suffer as well.