

Language modelling with N -grams

Herman Kamper

2023-04, CC BY-SA 4.0

The language modelling problem

N -gram language models

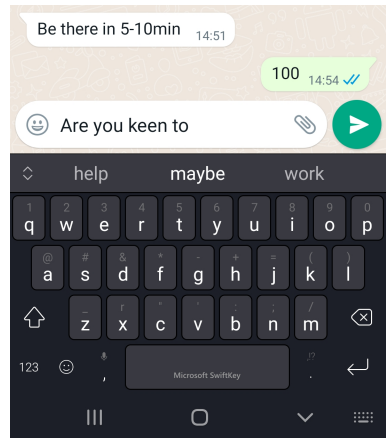
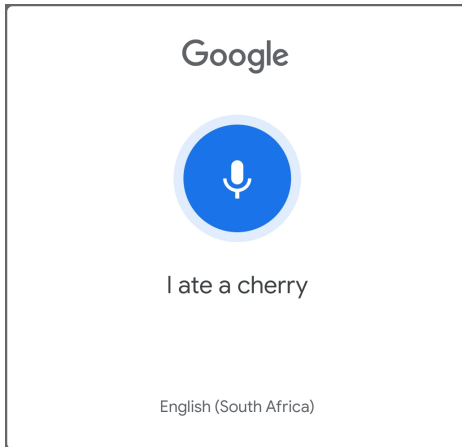
Evaluating language models

Smoothing

- Additive smoothing
- Interpolation and back-off
- Kneser-Ney

Exercises

Speech recognition and autocomplete



Speech recognition: How does Google know I am saying I ate a cherry and not I eight uh Jerry, although these are acoustically almost indistinguishable?¹ One solution would be to select the sentence that is most probable based solely on the written words. We could do this if we knew that:

$$P(\text{I ate a cherry}) > P(\text{I eight uh Jerry})$$

Autocomplete: How does Microsoft SwiftKey know that the next word that I want to type is probably maybe? For this we need to know the probability of an upcoming word:

$$P(\text{maybe} | \text{Are you keen to})$$

¹https://en.wikipedia.org/wiki/Jerry_Gergich

The language modelling problem

Generalizing the speech recognition example, for a sequence of words $w_{1:T} = w_1, w_2, \dots, w_T$, we need to know the probability

$$P(w_1, w_2, \dots, w_T)$$

For the autocomplete example we need

$$P(w_t | w_1, w_2, \dots, w_{t-1})$$

A model that estimates $P(w_{1:T})$ or $P(w_t | w_{1:t-1})$ is called a language model.

How do we calculate $P(w_{1:T})$? We repeatedly apply the product rule $P(A, B) = P(A|B)P(B)$:

$$P(w_{1:T}) =$$

$$= \prod_{t=1}^T P(w_t | w_{1:t-1})$$

We haven't actually made any assumptions yet: The above expansion is exact. So if we knew the true conditional probabilities $P(w_t | w_{1:t-1})$, we would be done!

But we almost never know the true probabilities. So we need a model that estimates these conditional probabilities.²

²It is sometimes worth explicitly distinguishing a true probability $P(x)$ from an estimated probability. I will sometimes use $P_E(x)$ or $P(x; E)$ to indicate that this is a probability from some estimation method E or model with parameters E . Sometimes I am sloppy and will just write $P(x)$ even when it is an estimate. This is because we are almost always talking about an estimate.

Naive language modelling approach

Let's say we need

$$P(\text{will}|\text{On day zero the water in Cape Town})$$

A naive approach would simply be to take a big corpus of training text and count how often will follows On day zero the water in Cape Town:

$$\begin{aligned} P_{\text{MLE}}(\text{will}|\text{On day zero the water in Cape Town}) \\ = \frac{C(\text{On day zero the water in Cape Town will})}{C(\text{On day zero the water in Cape Town})} \end{aligned}$$

where $C(w_{n:m})$ is the count of words $w_{n:m}$ in the training data. This is actually the maximum likelihood estimate (MLE); see [Exercise 1](#).

Sparsity (again!)

Many long sequences of particular words will never occur in the training data. Does this mean their probability should be zero?

We need to make some assumptions!

Towards N -gram language models

Bigram models: We assume that the probability of a given word only depends on the preceding word:

$$P(w_t|w_{1:t-1}) \approx P_{\text{bi}}(w_t|w_{t-1})$$

We still need to actually decide on a way to calculate $P_{\text{bi}}(w_t|w_{t-1})$. One way would be to again use the MLE:

$$P_{\text{MLE}}(w_t|w_{t-1}) = \frac{C(w_{t-1}, w_t)}{C(w_{t-1})}$$

This addresses the sparsity problem since we only need to look at the counts of two words occurring together (instead of very long sequences of words that are unlikely to occur).

Unigram models: We could define an even simpler model by assuming that every word just occurs independently:

$$P(w_t|w_{1:t-1}) \approx P_{\text{uni}}(w_t)$$

The MLE would be:

$$P_{\text{MLE}}(w_t) = \frac{C(w_t)}{W}$$

where W is the total number of tokens in our training data. This is a really dumb bag-of-words model since it completely ignores context or order. But it might actually be good enough for some applications.

Trigram models: We could increase the complexity from the bigram model by instead looking at two preceding words:

$$P(w_t|w_{1:t-1}) \approx P_{\text{tri}}(w_t|w_{t-1}, w_{t-2})$$

The MLE would be:

$$P_{\text{MLE}}(w_t|w_{t-1}, w_{t-2}) = \frac{C(w_{t-2}, w_{t-1}, w_t)}{C(w_{t-2}, w_{t-1})}$$

N -gram language models

We can generalize the models above by assuming that the probability of a given word depends on the $N - 1$ previous words:

$$P(w_t|w_{1:t-1}) \approx P_{N\text{-gram}}(w_t|w_{t-N+1:t-1})$$

The MLE would be:

$$P_{\text{MLE}}(w_t|w_{t-N+1:t-1}) = \frac{C(w_{t-N+1:t})}{C(w_{t-N+1:t-1})}$$

All N -gram language models make a Markov assumption: We can predict the probability of some future observation by only looking a finite number of observations into the past (we don't have to look all the way back to the start).

So for the models discussed, N would take on these values:

- Unigram: $N = 1$
- Bigram: $N = 2$
- Trigram: $N = 3$

How do we choose N ? This will depend on your application and your training data. There is a trade-off:

- A higher N allows you to capture more context, but then you might run into more sparsity issues. If you have tons of training data, then this is maybe okay.
- A smaller N means that sparsity is less of a problem, but now you are modelling less context.

Modelling the start and end of sentences

Let's say you train a bigram language model and generate some sentences from the model. We get three sentences:

1. galaxy was far away
2. the force was
3. at the start of the

Are these proper sentences? What is missing?

To capture behaviour at the beginning and the end of sentences, we need to augment the input. One way to do this is to add tokens indicating the start and end of the sentences:

1. <s> the galaxy was far away </s>
2. <s> the force was strong with him </s>
3. <s> at the start of the movie he sat down </s>

Concretely, we assume that $w_0 = \text{<s>}$ and $w_{T+1} = \text{</s>}$. So for a bigram model we would have:

$$P(w_{0:T+1}) = P(w_0) \prod_{t=1}^{T+1} P(w_t | w_{t-1}) = \prod_{t=1}^{T+1} P(w_t | w_{t-1})$$

I will mostly still write the product from 1 to T and write $w_{1:T}$, but just remember to deal with the start and end of sentences in this way.

A valid probability distribution

There is another reason to add an end-of-sentence token: Without it we are not defining a proper probability distribution. Without getting too deep into the details here, have a look at the discussion above and see how we dealt with the sequence length T . The answer is: We didn't. But this is a problem, since when you write $P(w_{1:T})$, the words $w_{1:T}$ aren't the only random variables: The length T is also a random variable. And we didn't model this at all! By defining an end-of-sentence symbol, we are implicitly modelling duration. And this then actually results in a valid probability distribution. See [Exercise 2](#).

Start-of-sentence symbol

The start-of-sentence symbol is also important. When we write $P(w_1, w_2, \dots)$, w_1 is specifically the variable for the first word in the sentence, w_2 the second word, and so on. When we make the N -gram assumption, terms for the middle words in a sentence will be the same irrespective of their position, e.g. in a bigram LM, $P(w_5|w_4)$ and $P(w_6|w_5)$ are determined in the same way. But the start of the sentence should still be dealt with specifically. Concretely, a word might have a very high unigram probability, but might not have a high probability for starting a sentence. The start-of-sentence symbol addresses this.

In practice: Use logs

Multiplying together many numbers that are all smaller than one often causes numerical underflow.

To deal with this, we almost always calculate probabilities in the log domain:

$$\log(p_1 \cdot p_2 \cdot p_3 \cdot p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

So to calculate the probability of a sentence $w_{1:T}$ with an N -gram model, we would calculate:

$$\begin{aligned}\log P_{N\text{-gram}}(w_{1:T}) &= \log \prod_{t=1}^T P_{N\text{-gram}}(w_t | w_{t-N+1:t-1}) \\ &= \sum_{t=1}^T \log P_{N\text{-gram}}(w_t | w_{t-N+1:t-1})\end{aligned}$$

Evaluating language models: Perplexity

A good language model assigns a high probability to observed sentences (and a low probability to everything else).

Given a test set $w_{1:T}$ of observed words, we can calculate the perplexity of model θ :

$$PP = P_{\theta}(w_{1:T})^{-\frac{1}{T}}$$

The test data $w_{1:T} \sim P(w_{1:T})$ is a sample from the real world (with some unknown distribution). A good language model will have low perplexity.

As usual, we compute perplexity using logs. This reveals how perplexity is a per-word negative log likelihood:

$$\log_2 PP =$$

Foundations in information theory:

- Is related to the cross-entropy H between a model θ and the real-world distribution, estimated from a test sample:

$$PP = 2^{H(w_{1:T}, \theta)}$$

- Can be interpreted as the weighted average branching factor: The number of possible words that can follow any word (on average).
- More on this in the separate note on [entropy and perplexity](#).

Training, validation, testing

We might have some hyperparameters that we need to tune for a language model, and we can use perplexity as a metric to do so. We follow the [standard practice in machine learning](#):

- Train a model (estimate probabilities) on a training set with different settings for the hyperparameters.
- Choose the hyperparameter values that give the best perplexity on validation data.
- After deciding on the final hyperparameter values, report the final perplexity on test data.

Example: Comparing language models

Use perplexity to compare language models trained on a single sentence from Europarl.

Entropy of a trigram language model on a test sentence:

	P_{tri}	$-\log_2 P_{\text{tri}}$
$P_{\text{tri}}(\text{i} \text{<s> <s>})$	0.109	3.197
$P_{\text{tri}}(\text{would} \text{<s> i})$	0.144	2.791
$P_{\text{tri}}(\text{like} \text{i would})$	0.489	1.031
$P_{\text{tri}}(\text{to} \text{would like})$	0.905	0.144
$P_{\text{tri}}(\text{commend} \text{like to})$	0.002	8.794
$P_{\text{tri}}(\text{the} \text{to commend})$	0.472	1.084
$P_{\text{tri}}(\text{rapporteur} \text{commend the})$	0.147	2.763
$P_{\text{tri}}(\text{on} \text{the rapporteur})$	0.056	4.150
$P_{\text{tri}}(\text{his} \text{rapporteur on})$	0.194	2.367
$P_{\text{tri}}(\text{work} \text{on his})$	0.089	3.498
$P_{\text{tri}}(\text{.} \text{his work})$	0.290	1.785
$P_{\text{tri}}(\text{</s>} \text{work .})$	0.99999	0.000014
Average		2.634

Comparison of language models:

Word	Unigram	Bigram	Trigram	Four-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
Average	8.051	4.072	2.634	2.251
Perplexity	265.136	16.817	6.206	4.758

Unseen N -grams

At test time we might see words that doesn't occur in our vocabulary \mathcal{V} . Normally we replace these with `<unk>`.

But we might have problems even with words that are in our vocabulary.

Example: Zero-count trigrams

We have the following counts for words following `denied the` in WSJ Treebank 3:

```
denied the allegations: 5
denied the speculation: 2
denied the rumors:      1
denied the report:      1
```

If these are the counts, then we will assign the following trigram probabilities using maximum likelihood:

$$P_{\text{MLE}}(\text{offer}|\text{denied the}) = 0$$
$$P_{\text{MLE}}(\text{loan}|\text{denied the}) = 0$$

Is this reasonable? Any sentence with the words `denied the offer` will be assigned zero probability.

```
He denied the offer to go to the Supreme Chancellor.
```

Sparsity (again!)

Most N -gram types will actually never occur. E.g. if we have a vocabulary of $|\mathcal{V}| = 50\,000$ words, then there are $(50\,000)^3 = 1.25 \cdot 10^{14}$ unique possible trigrams. At one byte per trigram type, that takes 125 TB of space.

Consider the [Google \$N\$ -grams dataset](#), which can be [browsed](#):

- Number of words in vocabulary: 13 588 391
- Five-gram types occurring at least 40 times: 1 176 470 663
- But the possible number of five-gram types: $4.63 \cdot 10^{35}$

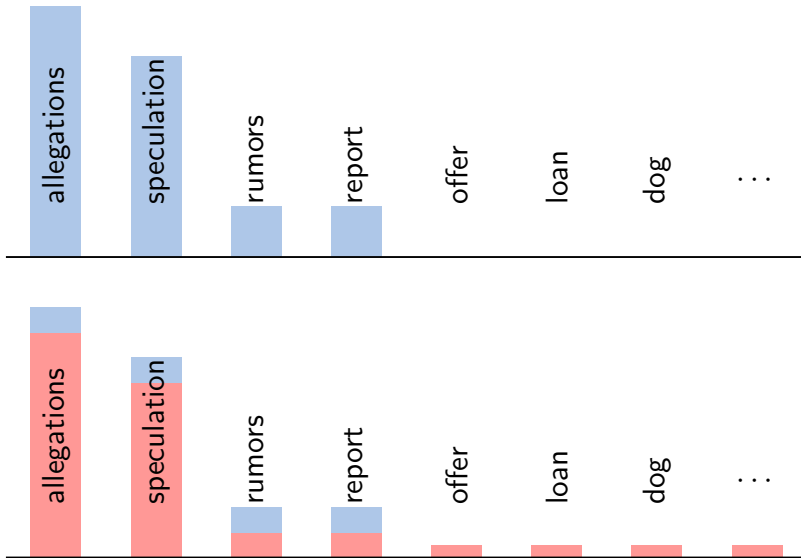
Overfitting

Assigning zero probability to N -grams not seen in the training data is really a form of overfitting to the training data. How do we deal with overfitting in machine learning? Regularization.

Language model smoothing are regularization approaches for N -gram language models. It typically results in higher perplexity on the training data, but lower perplexity on evaluation data.

Additive smoothing

Robin Hood: Steal from the rich and give to the poor.



There are different ways of moving probability mass.

Below I give the smoothing equations for bigrams or trigrams, but in all cases these can be extended to arbitrary N -grams (sometimes with little effort, sometimes a bit more).

Add-one smoothing (Laplace)

Without smoothing, the MLE for bigrams are:

$$P_{\text{MLE}}(w_t|w_{t-1}) = \frac{C(w_{t-1}, w_t)}{C(w_{t-1})}$$

In add-one smoothing we add one to each count and normalize appropriately for all possible bigrams:

$$P_{+1}(w_t|w_{t-1}) = \frac{C(w_{t-1}, w_t) + 1}{C(w_{t-1}) + |\mathcal{V}|}$$

We need the $+|\mathcal{V}|$ in the denominator to ensure that

$$\sum_{w_t \in \mathcal{V}} P_{+1}(w_t|w_{t-1}) = 1$$

Example: In the Berkeley Restaurant Project, we have the following counts:

$$C(\text{want}) = 927$$

$$C(\text{want to}) = 608$$

$$C(\text{want want}) = 0$$

$$|\mathcal{V}| = 1446$$

Estimate the probabilities for $P(\text{to}|\text{want})$ and $P(\text{want}|\text{want})$ with and without add-one smoothing.

$$P_{\text{MLE}}(\text{to}|\text{want}) =$$

$$P_{\text{MLE}}(\text{want}|\text{want}) =$$

$$P_{+1}(\text{to}|\text{want}) =$$

$$P_{+1}(\text{want}|\text{want}) =$$

Problem: We often steal way too much! This is because the $|\mathcal{V}|$ in the denominator can completely overpower $C(w_{t-1})$. In the example above, we had a probability go from 0.6559 to 0.2566!

Add- α smoothing (Lindstone)

In add- α smoothing, we add $0 < \alpha < 1$ to each count and normalize appropriately:

$$P_{+\alpha}(w_t|w_{t-1}) = \frac{C(w_{t-1}, w_t) + \alpha}{C(w_{t-1}) + \alpha|\mathcal{V}|}$$

Tune α on validation data.

Absolute discounting

The additional term in the denominator for add-one or add- α smoothing might still screw things up. We also treat N -grams the same irrespective of how often they occur: Whether an N -gram occurs zero or a million times, we add 1 or α to the numerator and $|\mathcal{V}|$ or $\alpha|\mathcal{V}|$ to the denominator.

Why not treat zero-count N -grams separately from higher-order N -grams? Idea: Steal counts from higher-order N -grams without changing the denominator, and then just distribute the left-over mass between zero-count N -grams.

Absolute discounting steals a constant $0 < d < 1$ from each higher-order N -gram type:

$$P_{\text{abs}}(w_t|w_{t-1}) = \begin{cases} \frac{C(w_{t-1}, w_t) - d}{C(w_{t-1})} & \text{if } C(w_{t-1}, w_t) > 0 \\ p_0 & \text{if } C(w_{t-1}, w_t) = 0 \end{cases}$$

To figure out what p_0 is, we use the property that probabilities need to sum to 1. The math gets quite hairy! See [Exercise 3](#).

In the general discounting case, we could make d a function of the N -gram count, which would allow you to e.g. steal much more from frequent N -grams and maybe less from N -grams occurring once or twice. But in practice absolute discounting is pretty good already.

Interpolation and back-off

Additive smoothing ignores a useful source of information: Less context can help when there isn't sufficient evidence for higher-order context.

Example: Drinking beer in Scotland

We have a corpus with the following trigram counts:

Scottish beer was:	20
Scottish beer can:	16
Scottish beer awards:	7
Scottish beer brands:	3

The following trigrams are never seen:

Scottish beer drinkers:	0
Scottish beer eaters:	0

How would the following estimates differ, given this data? Is this reasonable?

$$P_{\text{MLE}}(\text{drinkers}|\text{Scottish, beer})$$

$$P_{\text{MLE}}(\text{eaters}|\text{Scottish, beer})$$

But do you think the two bigrams below will have the same count?

beer drinkers
beer eaters

Higher- and lower-order N -gram models have different strengths:

- Higher-order models are more sensitive to context but are based on sparse counts.
- Lower-order models have more limited context but are based on reasonable counts.

We look at two approaches to combine models of different orders.

Interpolation

Use a mixture of models. For the trigram case:

$$\begin{aligned} P_{\text{int}}(w_t|w_{t-2}, w_{t-1}) &= \lambda_1 P_1(w_t) \\ &\quad + \lambda_2 P_2(w_t|w_{t-1}) \\ &\quad + \lambda_3 P_3(w_t|w_{t-2}, w_{t-1}) \end{aligned}$$

For instance:

$$\begin{aligned} P_{\text{int}}(\text{drinkers}|\text{Scottish beer}) &= \lambda_1 P_{\text{uni}}(\text{drinkers}) \\ &\quad + \lambda_2 P_{\text{bi}}(\text{drinkers}|\text{beer}) \\ &\quad + \lambda_3 P_{\text{tri}}(\text{drinkers}|\text{Scottish beer}) \end{aligned}$$

We need

$$\sum_{k=1}^K \lambda_k = 1$$

so that

$$\sum_{w_t \in \mathcal{V}} P_{\text{int}}(w_t|w_{t-2}, w_{t-1}) = 1$$

The λ 's are optimized on validation data.

You can combine interpolation with additive smoothing techniques.

Back-off

In back-off we use the highest-order model if the count is not zero, otherwise we back off to a lower-order model.

We need to discount the higher-order models in order to spread mass to the lower-order models. And then we need to make sure the probabilities sum to 1.

Back-off N -gram model:

$$P_{\text{BO}}(w_t|w_{t-N+1:t-1}) = \begin{cases} P_d(w_t|w_{t-N+1:t-1}) & \text{if } C(w_{t-N+1:t}) > 0 \\ \alpha(w_{t-N+1:t}) P_{\text{BO}}(w_t|w_{t-N+2:t-1}) & \text{if } C(w_{t-N+1:t}) = 0 \end{cases}$$

where

- $P_d(w_t|w_{t-N+1:t-1})$ is some discounted N -gram model.
- The back-off weights $\alpha(w_{t-N+1:t})$ are such that the probability sum to 1.

The math for the α 's gets quite hairy! See J&M2.

You can combine back-off with additive smoothing techniques. The math for the α 's gets quite hairy! See J&M1.

Kneser-Ney smoothing

In the Europarl corpus:

- York occurs 477 times. As frequent as foods, indicates and providers.
- This leads to a relatively high unigram estimate for $P(\text{York})$.
- But, York almost always follows New (473 times).
- So in unseen bigram contexts, York should have a low probability. But when we back-off or interpolate, the unigram probability for York will be high! I.e. the $P(\text{York}|\text{<not New>})$ is too high.

Kneser-Ney smoothing takes diversity of histories into account.

We count the number of bigram types ending in w_t :

$$N(\bullet, w_t) = |\{w_{t-1} : C(w_{t-1}, w_t) > 0\}|$$

Instead of using the unigram MLE

$$P_{\text{MLE}}(w_t) = \frac{C(w_t)}{\sum_{w \in \mathcal{V}} C(w)} = \frac{C(w_t)}{W}$$

we use the unique history counts

$$P_{\text{KN}}(w_t) = \frac{N(\bullet, w_t)}{\sum_{w \in \mathcal{V}} N(\bullet, w)} = \frac{N(\bullet, w_t)}{N(\bullet, \bullet)}$$

To do this we need to get some extra mass from somewhere, and we get this by combining absolute discounting with interpolation:

$$P_{\text{KN}}(w_t|w_{t-1}) = \frac{\max\{C(w_{t-1}, w_t) - d, 0\}}{C(w_{t-1})} + \lambda(w_{t-1})P_{\text{KN}}(w_t)$$

In the higher-order case we will have even more λ 's. The math for the λ 's gets quite hairy! See J&M3.

N-gram language models today

In many NLP applications today there is a move to neural language models (later).

But *N*-gram language models are still used!

- Still gives a strong baseline in automatic speech recognition.
- Easy to incorporate into a decoder.
- If you are searching for specific keywords, can easily push up their mass (and discount the rest).
- Normally use a large context e.g. five-gram.

N-gram language modelling toolkits:

- [SRILM](#)
- [KenLM](#)

Exercises

Exercise 1: Why are the normalized counts the MLE?

The note states that for an unsmoothed bigram model, the MLE is

$$P_{\text{MLE}}(w_t|w_{t-1}) = \frac{C(w_{t-1}, w_t)}{C(w_{t-1})}$$

Say we have a training set of a single observed word sequence $w_{1:T}$ and we fit a unsmoothed bigram language model. Prove that the above equation is indeed the MLE.

Hint: What are the model parameters? Define these as $\theta_{j|k} = P_{\text{bi}}(w_t = j|w_{t-1} = k)$ and minimize the negative log likelihood of the parameters with the constraint that $\sum_j \theta_{j|k} = 1$.

Exercise 2: Why we need an end-of-sentence symbol³

We are going to be a bit formal. Let's say we have a language with a vocabulary $\mathcal{V} = \{a, b, \langle s \rangle, \langle /s \rangle\}$. The set of possible sentence in this language is denoted as \mathcal{V}^+ . For a language model to give a valid probability distribution, we need:

$$\sum_{w_{1:T} \in \mathcal{V}^+} P_{\theta}(w_{1:T}) = 1$$

where $w_{1:T}$ denotes a particular sentence in this language.

Given the following training corpus without an end-of-sentence token:

$\langle s \rangle$ a a
 $\langle s \rangle$ a b
 $\langle s \rangle$ b a
 $\langle s \rangle$ b b

the set of all possible sentences would be

$$\mathcal{V}^+ = \{\langle s \rangle a, \langle s \rangle b, \langle s \rangle a a, \langle s \rangle a b, \langle s \rangle b a, \langle s \rangle b b, \langle s \rangle a a a, \dots\}$$

Train an unsmoothed bigram language model on the above training corpus. Show that this language model does not give a valid probability distribution over all sentence lengths.

Now repeat the above question but on the following training corpus, which do include an end-of-sentence symbol $\langle /s \rangle$:

$\langle s \rangle$ a a $\langle /s \rangle$
 $\langle s \rangle$ a b $\langle /s \rangle$
 $\langle s \rangle$ b a $\langle /s \rangle$
 $\langle s \rangle$ b b $\langle /s \rangle$

For this case show that we do indeed get a language model that gives a valid probability distribution.

³Adapted from Exercise 3.5 in J&M3.

Exercise 3: Absolute discounting

In bigram absolute discounting, prove that zero-count bigrams with history word w_{t-1} are all assigned the probability

$$p_0 = \frac{1}{C_0} \sum_{r>0} \left[C_r \cdot \frac{d}{C(w_{t-1})} \right]$$

where $C_r = |\{v : C(w_{t-1}, v) = r\}|$ are the number of bigram types with history word w_{t-1} occurring exactly r times.

Acknowledgements

Some of the content in this note is based on:

- The NLP course of Jan Buys at the University of Cape Town.
- The NLP course of Sharon Goldwater at the University of Edinburgh.

References

S. F. Chen and J. Goodman, “An empirical study of smoothing techniques for language modeling,” *Computer Speech and Language*, 1999.

M. Collins, “[Language modelling](#),” *Columbia University*, 2013.