Article One: Two minutes NLP — Perplexity explained with simple probabilities

Language models, sentence probabilities, and entropy

Photo by Wojciech Then on Unsplash

In general, perplexity is a measurement of how well a probability model predicts a sample. In the context of Natural Language Processing, perplexity is one way to evaluate language models.

A language model is a probability distribution over sentences: it's both able to generate plausible human-written sentences (if it's a good language model) and to evaluate the goodness of already written sentences. Presented with a well-written document, a good language model should be able to give it a higher probability than a badly written document, i.e. it should not be "perplexed" when presented with a well-written document.

Thus, the perplexity metric in NLP is a way to capture the degree of 'uncertainty' a model has in predicting (i.e. assigning probabilities to) text.

Now, let's try to compute the probabilities assigned by language models to some example sentences and derive an intuitive explanation of what perplexity is.

Computing perplexity from sentence probabilities

Suppose we have trained a small language model over an English corpus. The model is only able to predict the probability of the next word in the sentence from a small subset of six words: *"a"*, *"the"*, *"red"*, *"fox"*, *"dog"*, and ".".

Let's compute the probability of the sentence *W,* which is "*a red fox.*".

The probability of a generic sentence *W*, made of the words *w1*, *w2*, up to *wn*, can be expressed as the following:

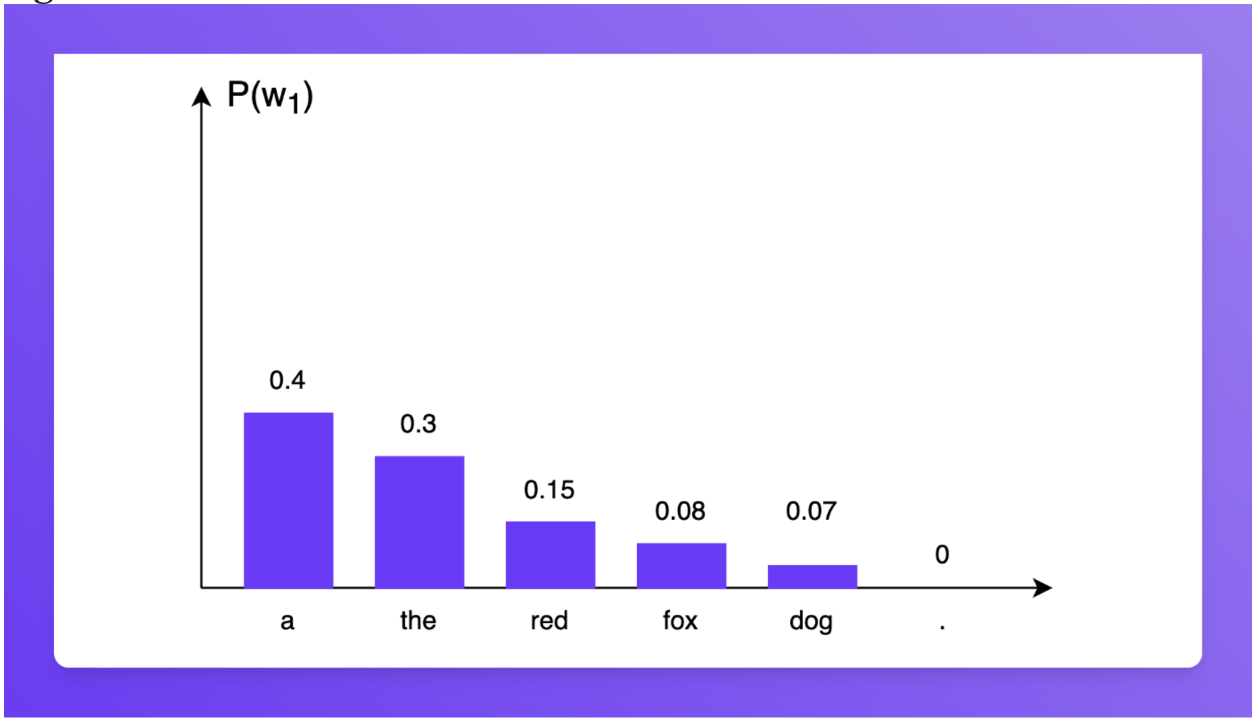
P(W) = P(w1, w2, …, wn)

Using our specific sentence *W*, the probability can be extended as the following:

P(“a red fox.”) =

P(“a”) * P(“red” | “a”) * P(“fox” | “a red”) * P(“.” | “a red fox”)

Suppose these are the probabilities assigned by our language model to a generic first word in a sentence:

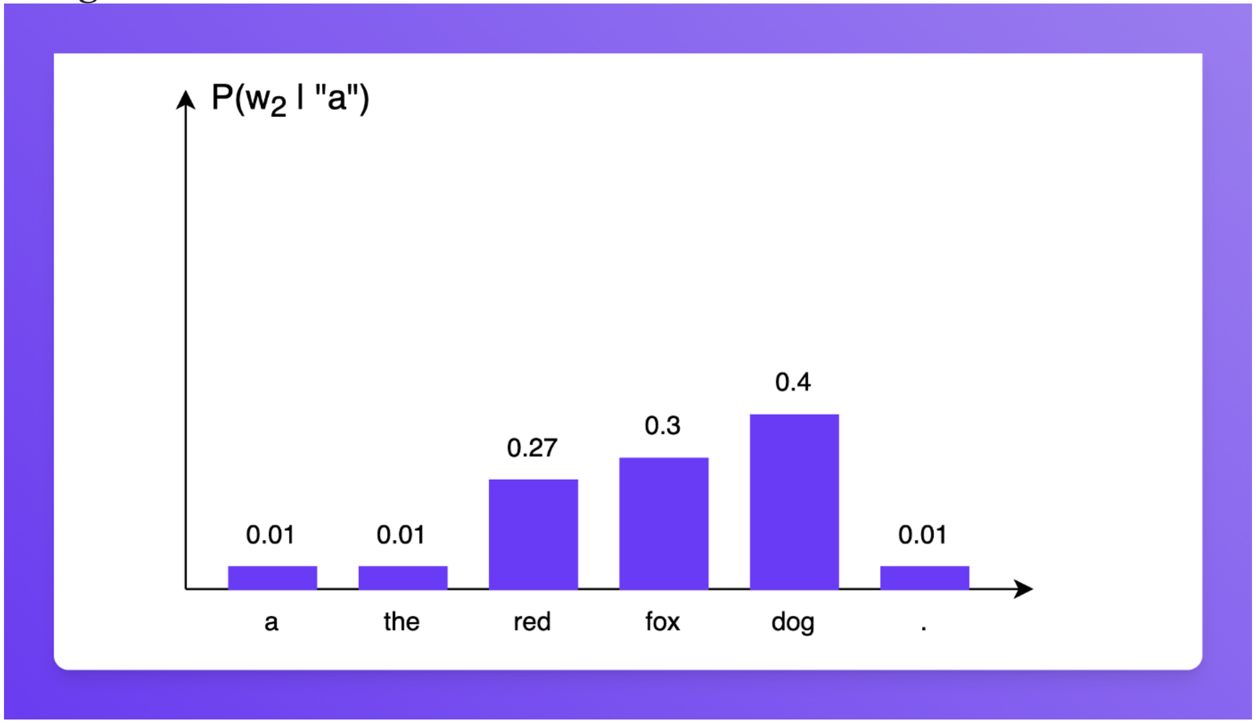


Probabilities assigned by a language model to a generic first word w1 in a sentence. Image by the author.

As can be seen from the chart, the probability of "*a*" as the first word of a sentence is:

P("a") = 0.4

Next, suppose these are the probabilities given by our language model to a generic second word that follows "*a*":

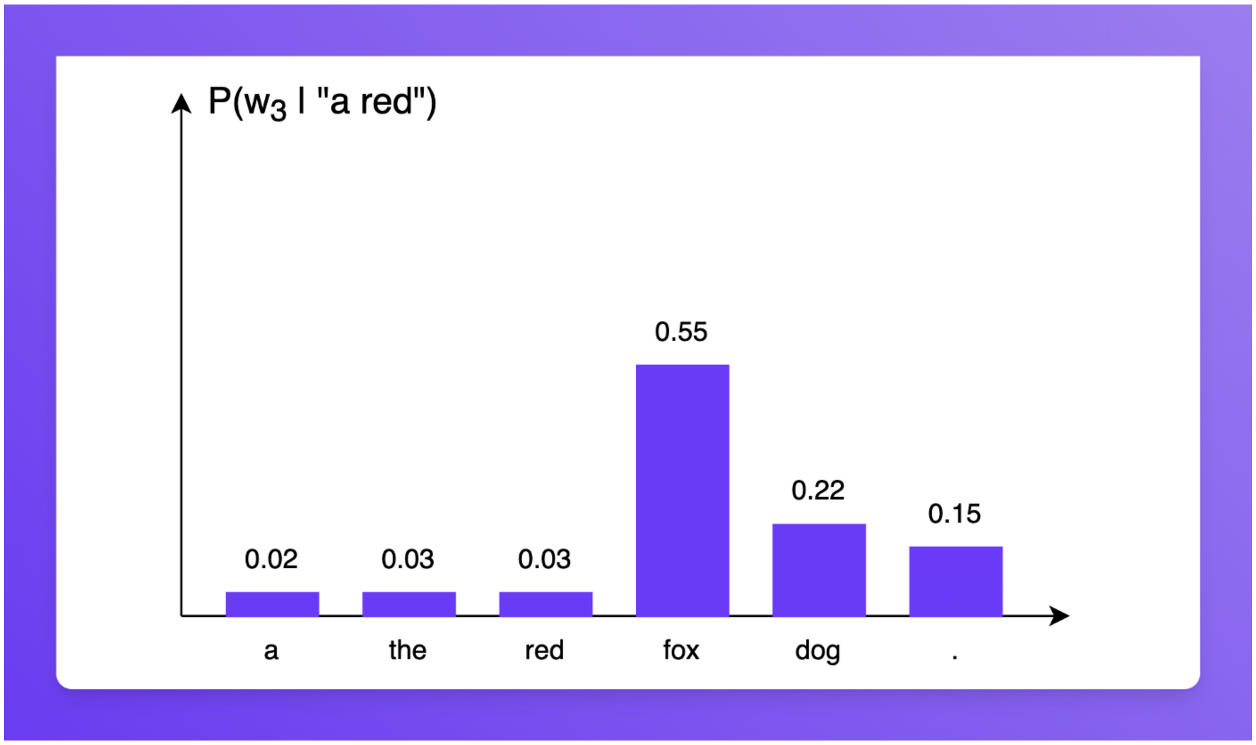


Probabilities assigned by a language model to a generic second word w2 in a sentence. Image by the author.

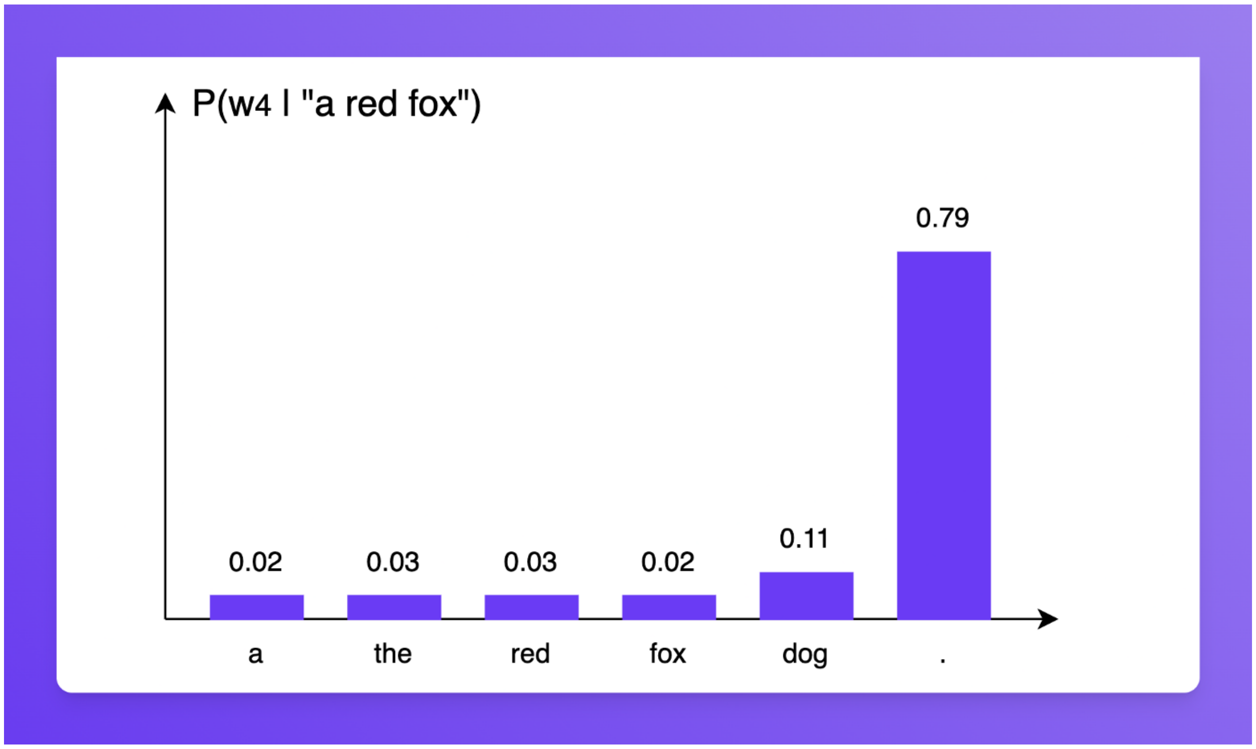
The probability of "*red*" as the second word in the sentence after "*a*" is:

P("red" | "a") = 0.27

Similarly, these are the probabilities of the next words:



Probabilities assigned by a language model to a generic third word w3 in a sentence. Image by the author.



Probabilities assigned by a language model to a generic fourth word w4 in a sentence. Image by the author.

Finally, the probability assigned by our language model to the whole sentence “*a red fox.*” is:

P(“a red fox.”) =

P(“a”) * P(“red” | “a”) * P(“fox” | “a red”) * P(“.” | “a red fox”)

= 0.4 * 0.27 * 0.55 * 0.79

= 0.0469

It would be nice to compare the probabilities assigned to different sentences to see which sentences are better predicted by the language model. However, since the probability of a sentence is obtained from a product of probabilities, the longer is the sentence the lower will be its probability (since it’s a product of factors with values smaller than one). We should find a way of measuring these sentence probabilities, without the influence of the sentence length.

This can be done by normalizing the sentence probability by the number of words in the sentence. Since the probability of a sentence is obtained by multiplying many factors, we can average them using the geometric mean.

Let’s call *Pnorm(W)* the normalized probability of the sentence *W*. Let *n* be the number of words in *W*. Then, applying the geometric mean:

Pnorm(W) = P(W) ^ (1 / n)

Using our specific sentence “*a red fox.*”:

Pnorm(“a red fox.”) = P(“a red fox”) ^ (1 / 4) = 0.465

Great! This number can now be used to compare the probabilities of sentences with different lengths. The higher this number is over a well-written sentence, the better is the language model.

So, what does this have to do with perplexity? Well, perplexity is just the reciprocal of this number.

Let’s call *PP(W)* the perplexity computed over the sentence *W*. Then:

PP(W) = 1 / Pnorm(W)

= 1 / (P(W) ^ (1 / n))

= (1 / P(W)) ^ (1 / n)

Which is the formula of perplexity. Since perplexity is just the reciprocal of the normalized probability, the lower the perplexity over a well-written sentence the better is the language model.

Let’s try computing the perplexity with a second language model that assigns equal probability to each word at each prediction. Since the language models can predict six words only, the probability of each word will be 1/6.

P(“a red fox.”) = (1/6) ^ 4 = 0.00077

Pnorm(“a red fox.”) = P(“a red fox.”) ^ (1/4) = 1/6

PP(“a red fox”) = 1 / Pnorm(“a red fox.”) = 6

…which, as expected, is a higher perplexity than the one produced by the well-trained language model.

Perplexity and Entropy

Perplexity can be computed also starting from the concept of Shannon entropy. Let’s call *H(W)* the entropy of the language model when predicting a sentence *W*. Then, it turns out that:

PP(W) = 2 ^ (H(W))

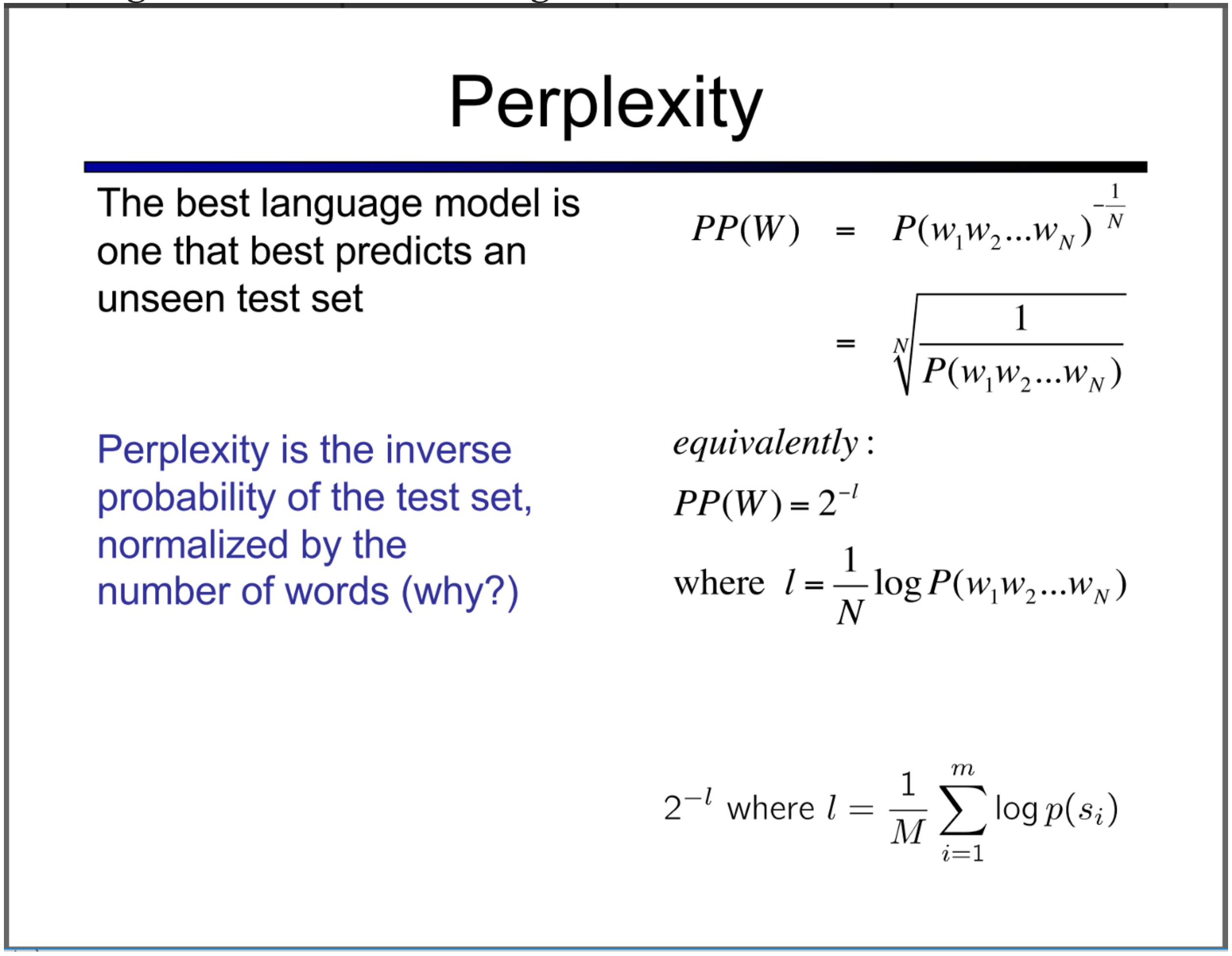
This means that, when we optimize our language model, the following sentences are all more or less equivalent:

- We are maximizing the normalized sentence probabilities given by the language model over well-written sentences.
- We are minimizing the perplexity of the language model over well-written sentences.
- We are minimizing the entropy of the language model over well-written sentences.

Article 2: Perplexity Intuition (and its derivation)

Never be perplexed again by perplexity.

You might have seen something like this in an NLP class:



Perplexity

The best language model is one that best predicts an unseen test set

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Perplexity is the inverse probability of the test set, normalized by the number of words (why?)

equivalently :

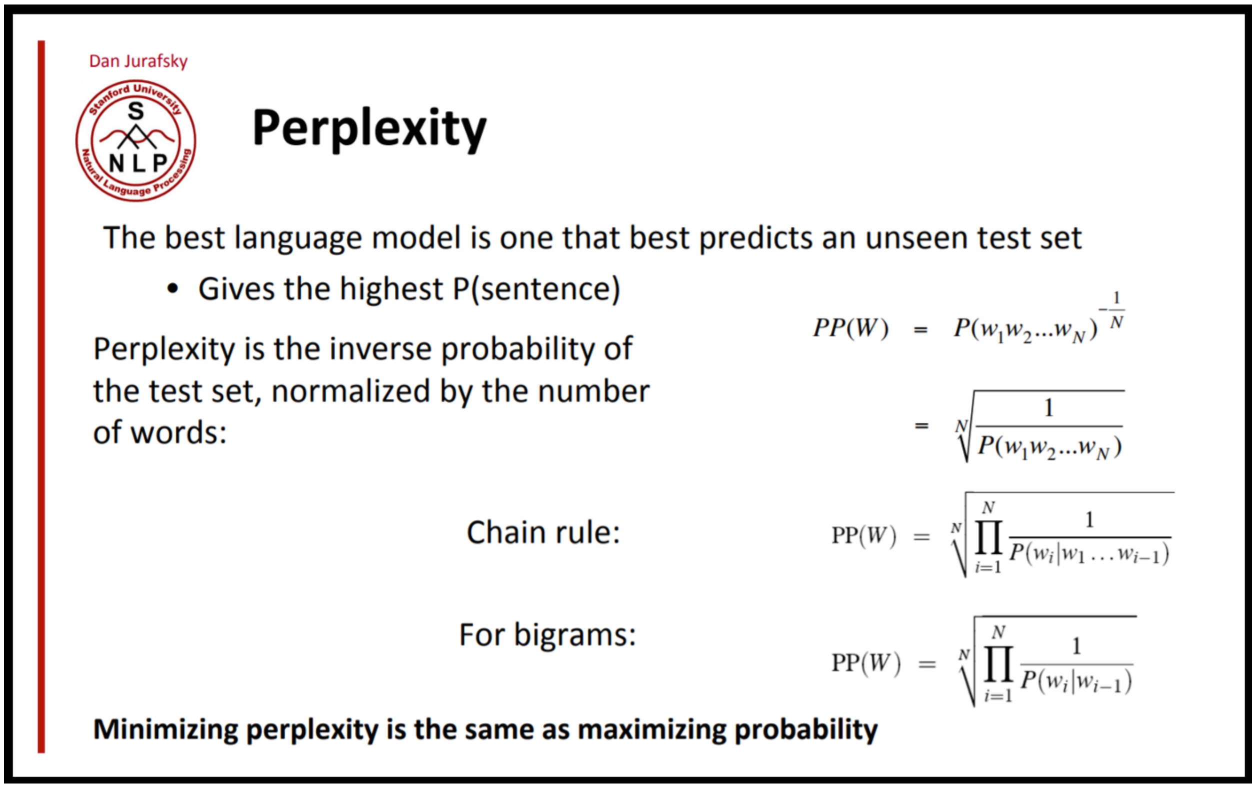
$$PP(W) = 2^{-l}$$

$$\text{where } l = \frac{1}{N} \log P(w_1 w_2 \dots w_N)$$

$$2^{-l} \text{ where } l = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)$$

A slide from Dr. Luke Zettlemoyer's NLP class

Or


Dan Jurafsky

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

A slide of CS 124 at Stanford (Dr. Dan Jurafsky)

During the class, we don't really spend time to derive the perplexity. Maybe perplexity is a basic concept that you probably already know? This post is for those who don't.

In general, perplexity is a measurement of **how well a probability model predicts a sample**. In the context of Natural Language Processing, perplexity is one way to **evaluate language models**.

But why is perplexity in NLP defined the way it is?

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

If you look up **the perplexity of a discrete probability distribution** in Wikipedia:

$$2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

from https://en.wikipedia.org/wiki/Perplexity

where **H(p) is the entropy of the distribution p(x)** and *x* is a random variable over all possible events.

In the previous post, we derived **H(p)** from scratch and intuitively showed **why entropy is the average number of bits that we need to encode the information.** If you don't understand **H(p)**, please read this ⇩ before reading further.

The intuition behind Shannon's Entropy
[WARNING: TOO EASY!]

Now we agree that H(p) =-Σ p(x) log p(x).

Then, perplexity is just an exponentiation of the entropy!

Yes. Entropy is the average number of bits to encode the information contained in a random variable, so the exponentiation of the entropy should be **the total amount of all possible information,** or more precisely, the weighted average number of choices a random variable has**.**

For example, **if the average sentence in the test set could be coded in 100 bits, the model perplexity is 2^{100} per sentence.**

Let's confirm that the definition in Wikipedia matches to the one in the slides.

$$\underset{\text{(of our model } q\text{)}}{\text{Perplexity}} = 2^{\text{entropy}} = 2^{\text{avg. \# of bits}}$$

$$= 2^{-\sum_{i=1}^{\overset{\text{tot \# of words}}{N}} \underbrace{p(\overset{\text{x can be any R.V. In NLP, it's each word.}}{x_i})}_{\text{real dist.}} \cdot \underbrace{\log_2 q(x_i)}_{\text{our prediction!}}}$$

$$= e^{-\sum_{i=1}^{N} p(x_i) \cdot \ln q(x_i)}$$

$$\xrightarrow{\text{We assume all words have the same frequency.}}$$

$$= e^{-\sum_{i=1}^{N} \frac{1}{N} \cdot \ln q(x_i)}$$

$$= q(x_1)^{-\frac{1}{N}} \cdot q(x_2)^{-\frac{1}{N}} \cdots q(x_N)^{-\frac{1}{N}}$$

$$= \prod_{i=1}^{N} q(x_i)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{q(x_1)\, q(x_2) \cdots q(x_n)}} \quad ▨$$

Where

p : A probability distribution that we want to model. A training sample is drawn from **p** and it's unknown distribution.

q : A proposed probability model. Our prediction.

We can evaluate our prediction ***q*** *by testing against samples drawn from* ***p.*** *Then* ***it's basically calculating the cross-entropy****. In the derivation above, we assumed all words have the same probability (1 / # of words) in* ***p****.*

Remarks

- When **q(x) = 0**, the perplexity will be **∞**. In fact, this is one of the reasons why the concept of smoothing in NLP was introduced.
- If we use a uniform probability model for **q** (simply 1/N for all words), the perplexity will be equal to the vocabulary size.
- The derivation above is for illustration purpose only in order to reach the formula in UW/Stanford slides. In both slides, it assumes that we are calculating the perplexity of the entire corpus using a unigram model and there is no duplicated word. (It assumes the # of total words (N) is the same as the number of unique words.) Also, it assumes all words have the same probability 1/N. These are not realistic assumptions.

Takeaway

- Less entropy (or less disordered system) is favorable over more entropy. Because predictable results are preferred over randomness. This is why people say **low perplexity is good and high perplexity is bad since the perplexity is the exponentiation of the entropy** (and you can safely think of the concept of perplexity as entropy).

- A language model is a probability distribution over sentences. And the best language model is one that best predicts an unseen test set.
- **Why do we use perplexity instead of entropy?**
  If we think of perplexity as a **branching factor** (the weighted average number of choices a random variable has), **then that number is easier to understand than the entropy.** I found this surprising because I thought there will be more profound reasons. I asked Dr. Zettlemoyer if there is any other reason other than easy interpretability. His answer was “I think that is it! **It is largely historical since lots of other metrics would be reasonable to use as well!”**