slidenumbers: true
footer: A. Ferrara. **Language models**. *Part 2: Basic notions and evaluation. email, corse website, slack, github

Information Retrieval

[fit]Language models

Part 2: Basic notions and evaluation. Prof. Alfio Ferrara

Master Degree in Computer Science

Master Degree in Data Science and Economics

A data-driven approach

With language models, we aim at building a computational model of the language of an entity X, where X can be almost any entity in a domain (e.g., the language of *politics*, the language of *sports*, the language of *John Smith*)

One of the main advantages of language models is that they are entirely **data-driven**

This means that in order to define the entity X we just need to collect a **corpus** of text documents that is representative of the entity X (e.g., collections of newspaper articles about *politics* or *sports*, emails send by *John Smith*)

The potential drawback of such an approach is that we do not actually model the **language of** X but instead the language of the **documents about** X, which makes it crucial the choice of the corpus used for defining X

[fit] n-gram language models

[fit] An example from the Cornell Movie-Dialogs Corpus

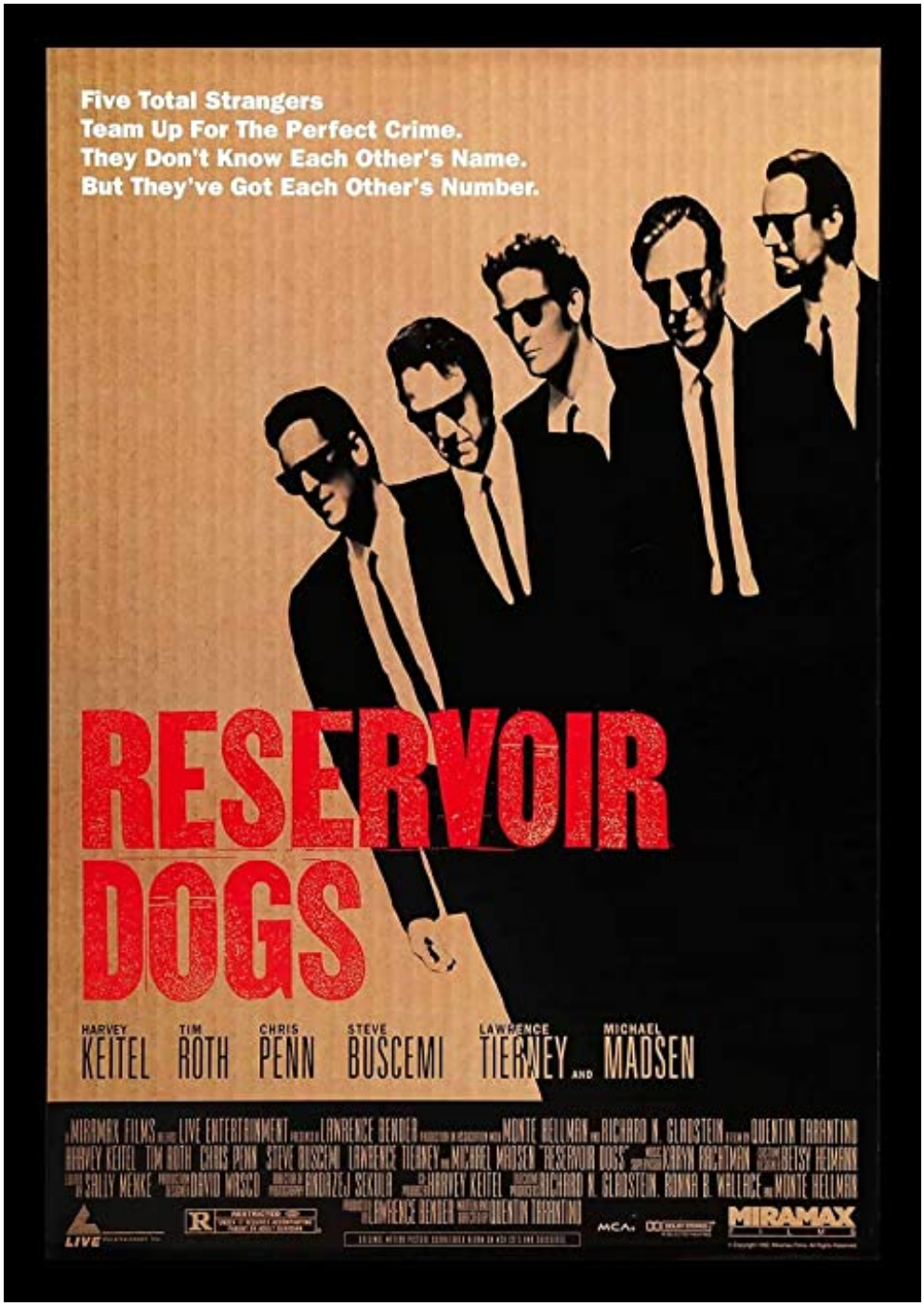
- Example overview
- Frequencies and maximum likelyhood estimation
- Corpora comparison
- N-gram models and Markov assumption
- Evaluating Language Models
- Generalization and zero probabilities

Example of corpus $\mathcal{C}_{m42}$

> Unoccupied France welcomes you to Casablanca.
> Thank you, Captain. It's very good to be here.
> Major Strasser, my aide, Lieutenant Casselle.
> You may find the climate of Casablanca a trifle warm, Major.
> Oh, we Germans must get used to all climates, from Russia to the Sahara. But perhaps you were not referring to the weather.
> What else, my dear Major? [...]

title: casablanca
year: 1942
rating: 8.8
votes: 170874
genres: drama, romance, war
size: 834
words: 9742

Example of corpus $\mathcal{C}_{m170}$



> Toby... who the fuck is Toby? Toby... Toby... think... think... think...
> It's not about a nice girl who meets a sensitive boy. Now granted that's what "True Blue" is about, no argument about that.
> Hey, fuck all that, I'm making a point here. You're gonna make me lose my train of thought.
> Oh fuck, Toby's that little china girl.

title: reservoir dogs
year: 1992
rating: 8.4
votes: 217185
genres: crime, mystery, thriller

size: 465
words: 8359

Frequencies: Given two corpora $\mathcal{C}_i$ and $\mathcal{C}_j$ describing entities i and j, respectively, a very starting step for studying their language is to look at words frequency in the two corpora [1]

Relative frequency: Corpora may be very different in the number N_w of words they contain. Thus, to make them comparable, instead of taking the absolute frequency $count(w)$ of words, we calculate the relative frequency $f(w)$ of words, that is:

$$f(w) = \frac{count(w)}{\sum\limits_{\substack{U_w \\ w_i=i}} count(w_i)} = \frac{count(w)}{N_w},$$

where U_w denotes the number of **unique** words in a corpus (i.e., the size of the corpus **dictionary**) and N_w is the number of words in the corpus (i.e., the number of word occurrences)

Maximum likelihood estimation (MLE)

The use of *relative frequencies* is also a way to estimate **probabilities** as maximum likelihood estimation (MLE).

Given a word w such that $count(w) = 13$ in a corpus with $N_w = 9742$ we have then:

$f(w) = \frac{count(w)}{N_w} = \frac{13}{9742} = 0.0013 \approx p(w)$

0.0013 is so the chance of random extracting the word w from the corpus. However, this is not the best estimate of $p(w)$ in general (depends on the corpus size), but 0.0013 is the probability that makes it **most likely** that w will occur 13 times in a 9742 corpus.

Log probabilities

MLE provides a first, naive, language model, called **Unigram Language Model**, where:

$p(w_n, w_{n-1}, \ldots, w_2, w_1) = p(w_n)p(w_{n-1}) \ldots p(w_2)p(w_1)$

A practical problem with this, is that the more probabilities we multiply, the smaller the product becomes, potentialy causing numerical underflow

Thus, we usually work internally with **log probabilities** exploiting the following observation

$p(w_n)p(w_{n-1}) \ldots p(w_2)p(w_1) = \exp(\log p(w_n) + \log p(w_{n-1}) + \cdots + \log p(w_2) + \log p(w_1))$

Notes about unigram models

```
s = ['welcomes', 'you', 'to', 'casablanca']
p_w = np.prod([m42.p(w) for w in s])
p_w_log = np.sum([m42.p(w, log_probabilities=True) for w in s])
np.allclose([p_w], [np.exp(p_w_log)])
> True
```

In natural language, the order of words plays a crucial role in determining the meaning of sentences. However, **unigram models** deal with words independently.

```
s1 = ['welcomes', 'you', 'to', 'casablanca']
s2 = ['casablanca', 'you', 'to', 'welcomes']
p_w_log_1 = np.sum([m42.p(w, log_probabilities=True) for w in s1])
p_w_log_2 = np.sum([m42.p(w, log_probabilities=True) for w in s2])
np.allclose([p_w_log_1], [p_w_log_2])
> True
```

Compare corpora

Unigram probabilities are however useful to compare corpora. We denote $p(w|\mathcal{C})$ the probability of a word given the corpus $\mathcal{C}$, calculated as the probability $p(w)$ given by the unigram language model of the corpus $\mathcal{C}$. When we compare N corpora $\mathcal{C}_1, \ldots, \mathcal{C}_n$, we can also estimate the global probability $p^*(w)$ of a word by computing

$$p^*(w) = \frac{\sum_{i=1}^{N} count(w)_i}{\sum_{i=1}^{N} N_i}$$

According to this, we aim at estimating how much a word is relevant in a corpus by comparing the observed probability of the word in the corpus against the expected proability of that word in the corpus.

Pointwise Mutual Information (PMI)

$$pmi(a,b) = \log \frac{p(a,b)}{p(a)p(b)} = \log \frac{p(a|b)}{p(a)} = \log \frac{p(b|a)}{p(b)},$$

where $p(a)p(b)$ is the expected probabilities of observing the pair (a,b) by chance (assuming independence), while $p(a,b)$ is the observed probability of the pair (a, b).

That in our case is

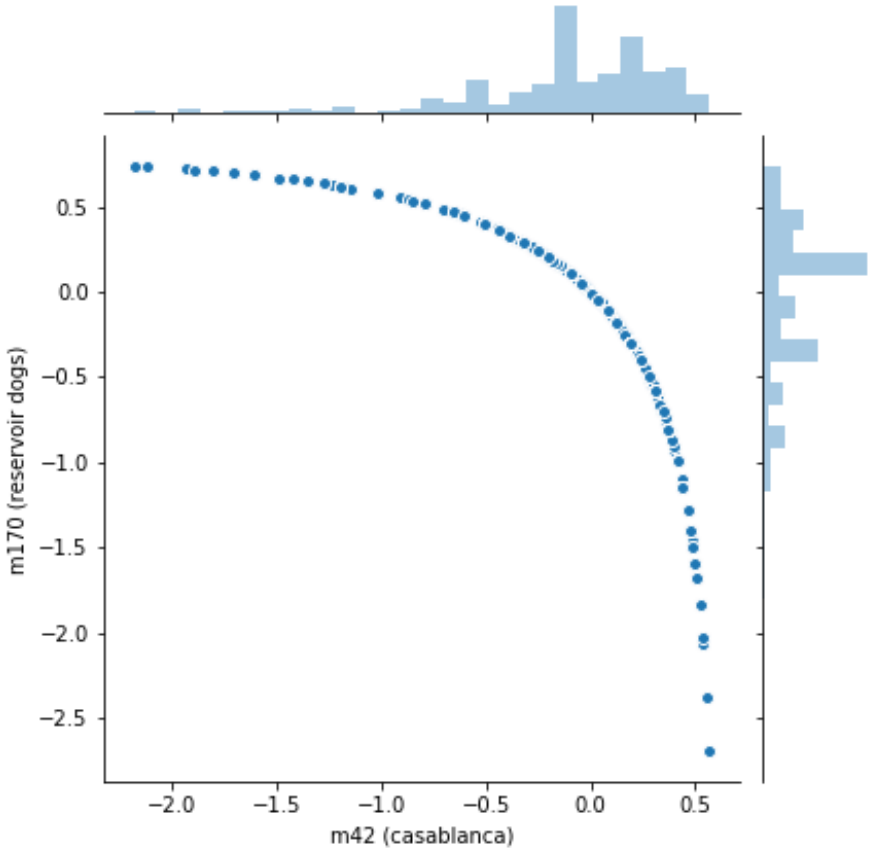
$$pmi(w,\mathcal{C}) = \log \frac{p(w,\mathcal{C})}{p(w)p(\mathcal{C})} = \log \frac{p(w|\mathcal{C})}{p(w)} = \log \frac{p(w)}{p^*(w)},$$

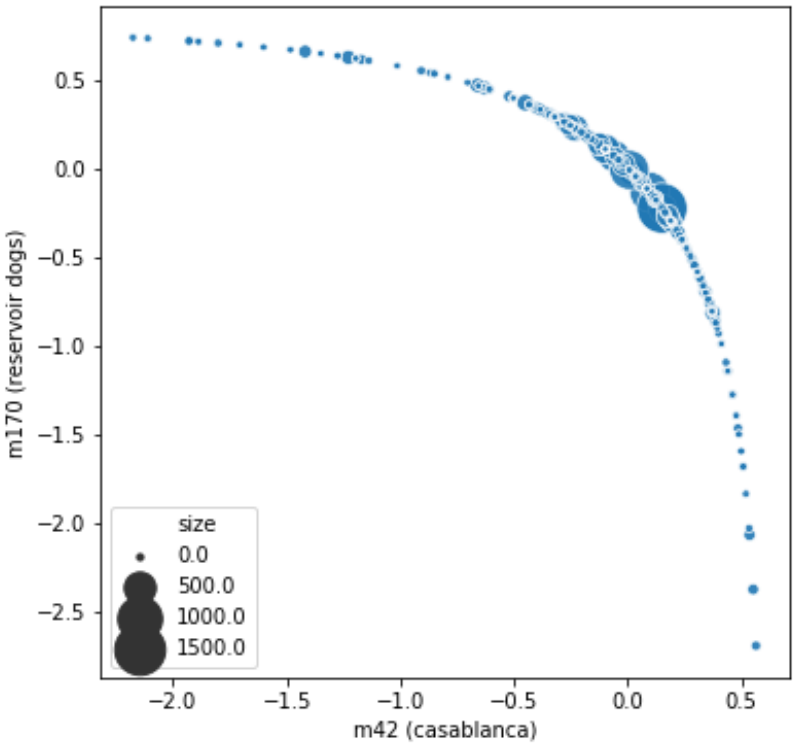
where $p(w)$ is the probability of the word w estimated by the unigram language model of $\mathcal{C}$

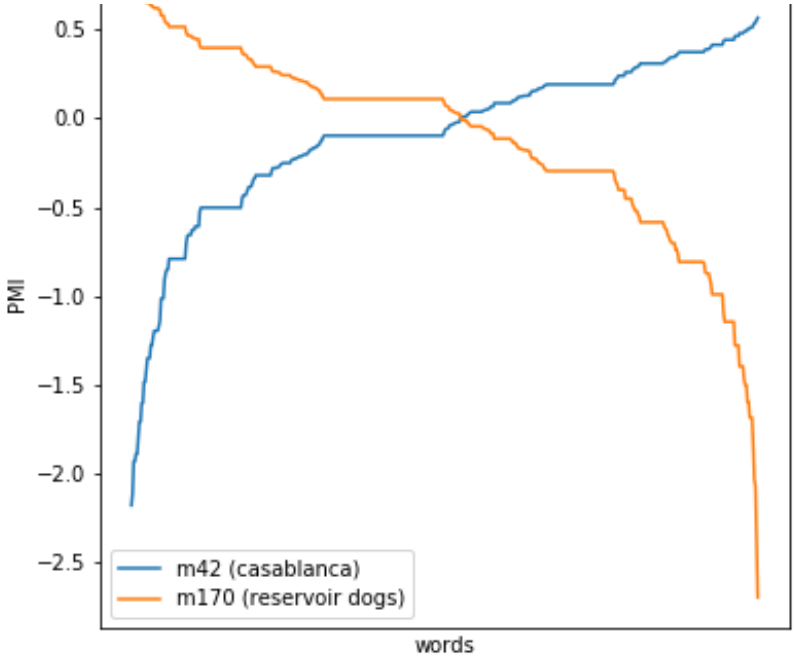
Example (I)

	woman	girl	man	boy	hello	yesterday
m42 (casablanca)	0.263	-0.074	0.178	-0.228	0.524	-0.074
m170 (reservoir dogs)	-0.431	0.079	-0.257	0.213	-1.625	0.079s

Example (II)


m170 (reservoir dogs)
m42 (casablanca)
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5


m170 (reservoir dogs)
m42 (casablanca)
size
0.0
500.0
1000.0
1500.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5



Dealing with text sequences

To overcome the limitation of assuming words to be independent, we should estimate the probability of a word sequence $p(w_i, \dots, w_n)$ by taking into account the ordered sequence of words, assuming that the probability of a word w_i depends on the previous w_{i-1} words (which is a more realistic assumption when working with natural language). In order to perform such an evaluation, we exploit the **chain rule** as follows:

$$p(w_1, \dots, w_n) = p(w_i) \cdot p(w_2 \mid w_1) \cdot p(w_3 \mid w_1, w_2) \cdot \dots \cdot p(w_n \mid w_1, w_2, \dots, w_{n-1}) = \prod_{i=0}^{n} p(w_i \mid w_1, \dots, w_{i-1})$$

Conditional probability estimation

Each term of the chain rule can be estimated as follows, using the Bayes rule:

$$p(w_i \mid w_1, \dots, w_{i-1}) = \frac{p(w_1,\dots,w_{i-1}|w_i)p(w_i)}{p(w_1,\dots,w_{i-1})} = \frac{p(w_1,\dots,w_i)}{p(w_1,\dots,w_{i-1})} = \frac{count(w_i,\dots,w_i)}{count(w_i,\dots,w_{i-1})}$$

Note that the probability $p(w_1, \dots, w_i)$ should be estimated as $\frac{count(w_1,\dots,w_{i-1},w_i)}{\sum_w count(w_1,\dots,w_{i-1},w)}$,

where $count(w_1, \dots, w_{i-1}, w)$ denotes the frequency of any sequence starting with $w_i, \dots w_{i-1}$.

However, the sum all counts starting with $w_i, \dots w_{i-1}$ is equal to $count(w_1, \dots, w_{i-1})$, which makes it possible to simplify the formula.

Conditional probability estimation in practice

In order to perform calculations, we need to get $count(w_1, \ldots, w_i)$ from the data. However, it is highly unlikely to observe a sufficient number of time a long sequence $w_1, \ldots, w_i$ in a corpus in order to have a robust estimation.

We have a trade-off: long sequences provide a more realistic estimation, but the estimation itself it is less robust. To address the problem, we exploit the **Markov assumption**, also known as **short-memory assumption**, that states that only the last $n-1$ words are used to estimate the conditional probability of a word.

This is called **n-gram language model**, that is:

$p(w_i \mid w_1, \ldots, w_{i-1}) \approx p(w_i \mid w_{i-n+1}, \ldots, w_{i-1})$

Bigram and trigram models

In practical applications, with medium sized corpora, n is usually set to 2 or 3, resulting in a **bigram (2-gram)** or a **trigram (3-gram)** model [2]

2-gram $\rightarrow p(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})} \rightarrow p(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i \mid w_{i-1})$

3-gram $\rightarrow p(w_i \mid w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})} \rightarrow p(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i \mid w_{i-1}, w_{i-2})$

Note that the 1-gram model is the unigram language model and corresponds to assuming word independence.

Implementation notes

In order to implement 2-gram and 3-gram models (let's take 2-gram as the example), we need to:

1. Add special words to text to mark sentence starting and ending
2. Indexing sequences of 2 words for counting (3 words and 2 words for 3-gram)

```
example = [m42.tokenize(x, remove_stopwords=False) for x in
           m42.get_lines_text() if 'york' in x.lower()]
tokens = example[1]
print(tokens)
> ['how', 'about', 'new', 'york']
mark = ['#S'] + tokens + ['#E']
print(mark)
> ['#S', 'how', 'about', 'new', 'york', '#E']
bigrams = list(nltk.ngrams(mark, 2))
print(bigrams)
> [('#S', 'how'), ('how', 'about'), ('about', 'new'), ('new', 'york'),
('york', '#E')]
```

Example (I)

2-grams and 3-grams take into account word order in sequences

```
print(m42.unigram['new'], m42.unigram['york'],
      m42.bigram[('new', 'york')], m42.bigram[('york', 'new')])
> 6 5 5 0
print(m42.p('new', 'york'))
> 0.83
print(m42.p('york', 'new'))
> 0
```

Example (II)

[('#S', 'i'), ('i', 'was'), ('was', 'born'), ('born', 'in'), ('in', 'new'), ('new', 'york'), ('york', 'city'), ('city', '#E')]

	i	was	born	in	new	york	city	#E
#S	0.14	0	0	0	0	0	0	0
i	0	0.04	0	0	0	0	0	0
was	0	0	0.02	0	0	0	0	0
born	0	0	0	1	0	0	0	0
in	0	0	0	0	0.02	0	0	0
new	0	0	0	0	0	0.83	0	0
york	0	0	0	0	0	0	0.2	0
city	0	0	0	0	0	0	0	0

Evaluating language models

Two options:

1. **Extrinsic evaluation**: embedding the language model into a real application and measure the application improvement
2. **Intrinsic evaluation**: define an evaluation metric and measure the performance of the model independent from any application

For the intrinsic evaluation, we need a **training set** on which the model is built and a **testing set**, made of texts unseen by the model on which run the evaluation. The testing set needs to be different but consistent with the language learned by the model.

Perplexity

The idea of testing is that whichever model assigns a higher probability to the test set, meaning it more accurately predicts the test set, is a better model. Given two probabilistic models, the better model is the one that has a tighter fit to the test data or that better predicts the details of the test data, and hence will assign a higher probability to the test data.

Given a test set $T = (w_1, \ldots, w_n)$, the performance of a model is measured by **perplexity**:

$$pp(T) = p(w_1, \ldots, w_n)^{-\frac{1}{n}}$$

Which is for a 2-gram model:

$$pp(T) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{p(w_i|w_{i-1})}}$$

A dataset for testing

Due to the limited number of words per movie and to the diversity of movies, training a model on a movie and then testing it on another movie is meaningless. To solve this, we create a special kind of language model in which, instead of dealing with sequences of **words**, we work with sequences of **part-of-speech** tags. This way all the movies contain a consistent information.

Example

```
import spacy
nlp = spacy.load("en_core_web_sm")

example = list(m42.get_lines_text())[0].lower()
print(example)
> unoccupied france welcomes you to casablanca.
print([x.pos_ for x in nlp(example)])
> ['ADJ', 'NOUN', 'VERB', 'PRON', 'ADP', 'NOUN', 'PUNCT']
```

Compare models

```
training = md.Movie(db_name=db_name, movie_id='m42')
test = md.Movie(db_name=db_name, movie_id='m170')
```

Average perplexity on POS models:

unigram 11.552
bigram 5.368
trigram 3.489

Average perplexity on TEXT models:

unigram 21.563
bigram 1.609
trigram 0.093

Generalization and zero probabilities

Language models, as many other statistical models, suffer from sparsity and unknown words. A word w_u that is in the test set but not in the training set, is said to be **unknown** in the model, which means that $p(w_u) = 0$.

Zeros are a problem for two reasons.

1. We are underestimating the probability of all sorts of words that might occur.
2. If the probability of any word in the test set is 0, the entire probability of the test set is 0. By definition, perplexity is based on the inverse probability of the test set. Thus if some words have zero probability, we cannot compute perplexity at all, since we cannot divide by 0.

Smoothing

Zeros are a problem also for words that are not unknown words but appear in a test set in an unseen context (for example they appear after a word they never appeared after in training).

To address this problem, an idea is to take some probability mass from some more frequent events and give it to the events that the model have never seen.

This process is called **smoothing**.

Laplace smoothing

The idea of Laplace Smoothing is just to add 1 to any word count as if zero words will be observed at least one time. Of course, this changes also the denominator of MLE, because we need to take into account the quantity W that is the total number of words (bacause each word has now a +1 count)

$p(w_i) = \frac{count(w_i)+1}{N_w+W}$

As an alternative way, instead of adding 1, we can add a fractional count k to the smoothing. This approach is called **k-smoothing**.

$p(w_i) = \frac{count(w_i)+k}{N_w+kW}$

Backoff and Interpolation (I)

Two other startegies to address zero counts is to use backoff or interpolation

Backoff: If we do not have enough evidence to rely on a n-gram, we exploit the $(n-1)$-gram. If still we do not enough evidence we downgrade the estimation untill we just use the unigram model.

$$p(w_1, w_2, w_3) = \begin{cases} p(w_3 \mid w_1, w_2) & \text{if } count(w_1, w_2, w_3) \geq k; \\ p(w_3 \mid w_2)p(w_2 \mid w_1) & \text{if } count(w_2, w_3) \geq k \wedge count(w_1, w_2) \geq k; \\ p(w_1)p(w_2)p(w_3) & \text{otherwise.} \end{cases}$$

Backoff and Interpolation (II)

Interpolation: with interpolation we always take into account all the n-gram levels by linearly combining them using an hyper parameter λ which can also be dynamically defined according to the evidence (counting) of the components.

$$p(w_3 \mid w_1, w_2) = \lambda_1 p(w_3) + \lambda_2 p(w_3 \mid w_2) + \lambda_3 p(w_3 \mid w_1, w_2)$$

with

$$\sum_i \lambda_i = 1$$

Evaluation with smoothing

Models performances with **k-smoothing** ($k = 0.5$) and log probabilities

Average perplexity on POS models:

unigram 0.93
bigram 0.83
trigram 0.8

Average perplexity on TEXT models:

unigram 0.986
bigram 0.835
trigram 0.628

---

1. We assume that documents have been tokenized in words by enforcing some tokenization and normalization function ↩

2. with huge corpora it is possible to scale at 4-gram or 5-gram. Rarely more. ↩