Introduction to text representations and language modeling

Matthieu Labeau - Associate Professor, Telecom Paris, IPP

IA-312: Natural Language Processing - 04/03/2024

Outline

- Course presentation
- Introduction to basic text processing
- Representing documents
- Representing words
- Computing probabilities: *n*-gram language models
- Issues with symbolic representations

Language Processing: goals

Speech and Language Processing (Jurafsky and Manning) (Chapter I)

Interdisciplinary field, whose goal is to get computers to perform useful tasks [..] like enabling human-machine communication, improving human-human communication, or simply doing useful processing of text or speech.

Applications ?

From the same source : example of **HAL 9000**, in *2001: A Space Odyssey*
$\rightarrow$ **Conversational agent** - what does HAL imply ?

- Language input: *speech recognition, language understanding*
- Language output: *dialogue planning, speech synthesis*
- *Information retrieval, extraction*, and doing inference from it

Statistical Language Processing

Historically, two paradigms: **symbolic** and **stochastic**

- On the symbolic side, formal systems based on logic and *grammars*
- Probabilistic models used early for tasks like optical *character recognition* regained popularity towards the end of the 80s, with models like **Hidden Markov Models** (HMMs) for *speech recognition*
- Probablistic **data-driven** models then rapidly became standard
- Statistical models took over rapidly from 2000, thanks to:
    - A large amount of material and resources (+computational)
    - Efficiency of statistical learning (+unsupervised approaches)
    - Community effort: shared tasks, evaluation campaigns
- .. and now soon represented the state-of-the-art for almost any task; now, **deep learning**

In this course

An introduction to natural language processing focused on **text representations** and **language modeling**:

- How did *pre-deep learning* methods work ?
- What challenges do we encounter when trying to represent natural language (such as textual data) numerically ?
- What did neural networks bring to the table ?
- What allowed for the huge performance increase of NLP models of the last 5 years ?
- How did large language models obtained their new capabilities ?

We will work on **classification** tasks.

- Labs: Python with *Scikit-learn*, then *Pytorch*
- Language processing libraries: *NLTK* and *Gensim*

Schedule and instructions

- 04/03 - A.M: Introduction to text representation and language modeling
- 04/03 - P.M: Lab - Text pre-processing and text representations
- 11/03 - A.M: Neural Language Models, Word Embeddings and Deep learning for NLP
- 11/03 - P.M: Lab - Text classification with Pytorch
- 18/03 - A.M: Structured prediction for NLP
- 18/03 - P.M: Lab - Structured prediction
- 25/03 - A.M: Transformer architecture and Large Language Models
- 25/03 - P.M: Lab - Machine Translation with Seq2seq models

- Slides and references to further content on moodle
- 2 of the 4 labs are graded (25% of the final grade each). They will be **due at the end of the week**
- You will be asked to read, summarize and comment one research paper (50% of the final grade)

Introduction to basic text processing

Basic text processing

What is the bare minimum to process text ?

→ Example of ELIZA

Speech and Language Processing (Jurafsky and Manning) (Chapter II)

```
User:   I am unhappy.
ELIZA:  DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
User:   I need some help, that much seems certain.
ELIZA:  WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
User:   Perhaps I could learn to get along with my mother.
ELIZA:  TELL ME MORE ABOUT YOUR FAMILY
User:   My mother takes care of me.
ELIZA:  WHO ELSE IN YOU FAMILY TAKES CARE OF YOU
User:   My father.
ELIZA:  YOUR FATHER
User:   You are like my father in some ways.
```

Weizenbaum (1966)

- Uses pattern matching to recognize phrases
- Uses a set of rules to translate them into suitable outputs
- Remarkably successful !

- For pattern matching: regular expressions !
- Used then for text normalization and tokenization
- Comparing: using the edit distance

Regular expressions

An algebraic notation for characterizing a set of string - used practically everywhere.

$\rightarrow$ Can be used to find desired occurences of words in a large text

Example: looking for ‘the’:

- Will miss some occurences: `the`
- Will find undesired ones: `[tT]he`
- Will probably have very few false positives or negatives: `[^a-zA-Z][tT]he[^a-zA-Z]`

$\rightarrow$ Can be used to capture and substitute text

- Replacing ‘the’ with ‘The’: `s/the/The`
- Capturing any string ending with ‘er’: `/(.*)er/`
- Getting a superlative: `s/the (.*)er/the (\1)est/`

Regular expressions and ELIZA

Regular expressions also allows for more complex fonctionalities.. but especially, allows for ELIZA:

- Early NLP system that imitated a Rogerian psychotherapist
- Has a series or cascade of regular expression substitutions, matching and changing some part of the input lines:
  - Input lines are uppercased
  - Change all instances of MY to YOUR, I'M to YOU ARE, etc...
  - Then, other patterns are replaced:

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Words

What counts as a word ? Some basic definitions:

- We begin from a **corpus** (plural corpora): computer-readable collection of text
- **Word types** are the number of distinct words in a corpus - usually grouped in a set, the **vocabulary**
- **Word tokens** are instances of types in the running text
- Two types can be different word forms of a same lemma: $\rightarrow$ *cat* and *cats* have the same lemma *cat*

Example: *"I showed my masterpiece to the grown-ups, and asked them whether the drawing frightened them."*

Corpora and resources

Data-driven methods are based on corpora, which have many distinct properties:

- Language (7000+) and varieties, code switching. . .
- Genre (news, scientific, fiction..), specific domain (medical, law. . .)
- Source and authors: how was it written ? Collected ? Why ?

Resources are unevenly distributed along languages ! They are usually example:

- Labeled data for many tasks - will allow supervised learning
- Lexical Databases like **WordNet** (but also for other languages)
- Careful: *annotation* is a difficult and subjective process

Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science (Bender and Friedman, 2018)

Use data statement to avoid biases !
→ Dataset rationale, data provenance, annotator demographic, ...

Tokenization

The first step to any task: pre-process the text, which begins by segmenting it into words. This is **tokenization** !

- Simplest approach: *space-based* - segments along spaces
  $\rightarrow$ Does not work with some languages !
- What to do with punctuation ? Example:
  'But they answered: "Frighten? Why would anyone be frightened by a hat?" My drawing was not a picture of a hat.'
  and many other depending on the context (hashtags, emails, etc..)
- Tools: we will use packages like NLTK (https://www.nltk.org/)
- Recently popularized, **data-driven tokenization** (*Byte Pair Encoding*, *Wordpiece*) based on subwords
  - Learn a vocabulary of tokens; segment using that vocabulary

Representing documents

Bag-of-words representations

In order to classify a document, we use lexical features. The simplest way to represent a document is as a **bag-of-words**

I walked down the street
I walk down the the avenue
I walked down the avenue
I ran down the street
I walk down the city

Compute the **vocabulary** !

- Which tokenization ?
- How to store it ?

$\rightarrow \mathcal{V} = \{\text{I}, \text{the}, \text{down}, \text{walked}, \text{street}, \text{avenue}, \text{walk}, \text{ran}, \text{city}\}$

Next, count !

Bag-of-words representations

The bag-of-words contains frequency counts, **unordered**: they are simple *lexical features*

I walked down the street
I walk down the the avenue
I walked down the avenue
I ran down the street
I walk down the city

	I	the	down	walked	street	avenue	walk	ran	city
D_1	1	1	1	1	1	0	0	0	0
D_2	1	2	1	0	0	1	1	0	0
D_3	1	1	1	1	0	1	0	0	0
D_4	1	1	1	0	1	0	0	1	0
D_5	1	1	1	0	0	0	1	0	1

Document representation: motivation

The Bag-of-word is a document model counting words

- Assumes that position does not matter . . . hence indifferent to *syntax* and *semantics*

- Main goal: **text classification** (sentiment, spam . . .) !
    - We can use rules based on words . . .
    - Example: with *SentiWordNet*, each 'word' is associated to a positive and negative score . . .
    - . . . or learn a classifier model that will use word frequencies as features
      $\rightarrow$ Naïve Bayes: assuming independance between words

- Also useful for document clustering, information retrieval. . . . why ?

Classification with Naïve Bayes

Multinomial naive Bayes classifier: a *generative* (why ?) linear classifier that naïvely assumes that features are independant

- Goal: for a document d return the class $\hat{c}$ with maximum a posteriori probability among classes: $\hat{c} = \text{argmax}_{c \in \mathcal{C}} P(c|d)$
- Applying **Baye's rule** and the **independance assumption**, and noting $d = (w_1, ..., w_n)$ we get a (*prior* $\times$ *likelihood*) decomposition:

$$\hat{c} = \text{argmax}_{c \in \mathcal{C}} \left[\mathbb{P}(c) \prod_{i=1}^{n} \mathbb{P}(w_i|c) \right]$$

- Idea: we compute $\mathbb{P}(w|c)$ as the frequency of w among all documents $d \in \mathcal{D}$ of class c:

$$\mathbb{P}(w|c) = \frac{count_{\mathcal{D}}(w, c)}{\sum_{w' \in \mathcal{V}} count_{\mathcal{D}}(w', c)}$$

- Given these assumptions, it is legitimate (and practical) to represent a document d with its bag-of-word representation $\mathbf{d}$!

Classification with Naïve Bayes

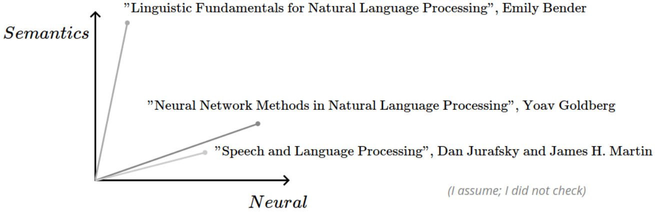
Practical points: Use **log-probabilities** (why ?), add 1 to each count (why ?)... more later.

Training Algorithm: Given data $\mathcal{D}$ and classes $\mathcal{C}$

- Create the vocabulary $\mathcal{V}$ from documents $d \in \mathcal{D}$
- From $\mathcal{D}$: For each class $c \in \mathcal{C}$: compute the log-prior $\log \mathbb{P}(c)$
  - For each word $w \in \mathcal{V}$ compute the log-likelihood $\log \mathbb{P}(w|c)$

- **Inference Algorithm**: Given document d to classify:
  - For each class $c \in \mathcal{C}$: $S(c) = \log \mathbb{P}(c)$
    - For each position $i \in d$:
      If $w_i \in \mathcal{V}$: $S(c) = S(c) + \log \mathbb{P}(w_i|c)$
  - Return $\text{argmax}_{c \in \mathcal{C}} S(c)$

Document as Vectors



- Words are **dimensions** of *documents vectors*
- You can vizualize vectors in a particular set of dimensions of your choosing
- Vectors should be similar for documents that are *related*
  - But what does *similar* mean here ?

Similarity between documents: cosine

The cosine of the angle between the document vectors is the most common similarity metric in NLP

- Based on the dot product, which alone favors long documents: more words, higher values.
- Normalizing the dot product gives us the cosine of the angle between the two vectors:

$$\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{||\mathbf{d}_1||\ ||\mathbf{d}_2||}$$

- Values range in $[-1, 1]$; frequencies $\rightarrow$ similarity is always positive
- $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = 0 \rightarrow$ the documents have no words in common

Still, frequency is not the best measure of association between words:

- It is skewed $\rightarrow$ Zipf's law (more on that later)
- Very frequent words are rarely the most useful for classification (more on that later)

Classification with logistic regression

A *discriminative* linear classifier: learns directly $\mathbb{P}(c|d)$ through computing a **linear score** and applying a **logistic function**.

- Binary case: for a set of documents $d \in \mathcal{D}$ represented by vectors $\mathbf{d}$ learn a vector $\mathbf{w}$ and a bias b maximizing the likelihood of making a good classification into $c = 1$ or $c = 0$.
- The probability $\mathbb{P}(c = 1)$ is obtained by applying the sigmoid to the *scalar product* plus *intercept*:

$$\mathbb{P}(c = 1) = \sigma(\mathbf{w} \cdot \mathbf{d} + b)$$

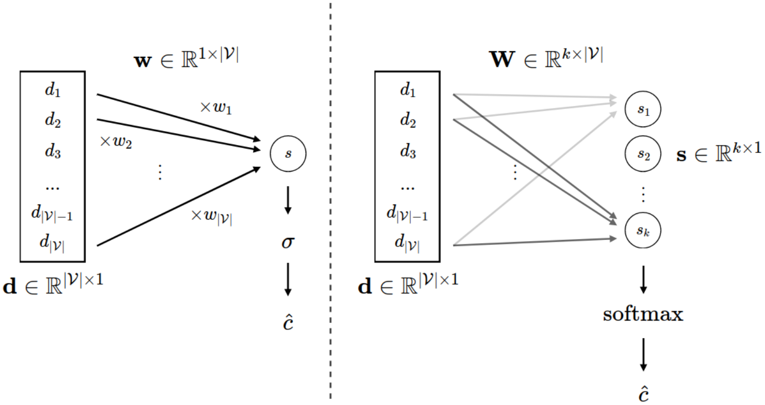
- We want to maximize the likelihood of our data by minimizing the **cross-entropy** between true and predicted classes $\hat{c}$ and c:

$$L(\hat{c}, c) = -\log \mathbb{P}(c|d) = -\left[c \log \hat{c} + (1 - c) \log(1 - \hat{c})\right]$$

- Here, the training is made through **gradient descent**: we minimize that loss function by finding iteratively the direction in which the loss decreases the most and updating the weights accordingly

Classification with logistic regression

The model is easily extended to a multinomial case through using a matrix $\mathbf{W}$, a vector $\mathbf{b}$ and the *softmax* function



Representing words

Words can be vectors too !

We can consider words as vectors of documents

- Or change the **context** for counting words: sentence-words matrix ?
- Why not only use **surrounding words**. . .
- The simplest possible context for words ? Other words ! Gives the *co-occurence* matrix:

	I	the	down	walked	street	avenue	walk	ran	city
I	0	6	5	2	2	2	2	1	1
the	6	2	6	2	2	3	3	1	1
down	5	6	0	2	2	2	2	1	1
walked	2	2	2	0	1	1	0	0	1
street	2	2	2	1	0	0	0	1	0
avenue	2	3	2	1	0	0	1	0	0
walk	2	3	2	0	0	1	0	0	1
ran	1	1	1	0	1	0	0	0	0
city	1	1	1	1	0	0	1	0	0

A little introduction to lexical semantics

In linguistics, study of **word meaning**:

- The meaning of a word (often simplified to its lemma) can consist in **several senses** (when it's the case, the lemma is polysemous - an important task in NLP is word sense disambiguation)
- There exists many relationships between word meanings: for example, two words that share a sense are **synonyms**
- Words can be *similar* but also only *related* (do you think of examples ?)
- How to evaluate word similarity ?
  - Ask humans ? Many datasets of similarities, like *SimLex-999*

Evaluating Semantic Models with (Genuine) Similarity Estimation, (Hill et al, 2015)

- clothes/closet: 1.96
- coast/shore: 9.00

  - Look at the annotator guidelines !

Vector Semantics

How to represent **word meaning** ?

- Using a vector based on the **distribution of context** !
  $\rightarrow$ Based on the distributional hypothesis: contextual information is sufficient to represent linguistic objects

Wittgenstein (Philosophical Investigations, 1953)

For a large class of cases [...] the meaning of a word is its use in the language.

Firth (1957)

You shall know a word by the company it keeps

- Meaning are embedded into a space: these vectors are called **Word Embeddings**: usual representation of meaning in NLP
- In our case, word meaning is represented by a vector describing *the distribution of other words in the neighborhood*
  $\rightarrow$ the **cosine distance** can be used to compute word similarity

Computing word probabilities

Solving ambiguities

Learning to resolve natural language ambiguities (Dan Roth, 1998)

Most tasks in Natural Language Processing can be viewed as resolving ambiguity at one of different levels !

- Ambiguity in written representation (*speech synthesis*)
- Lexical ambiguity, on word grammatical properties (*Part-of-speech tagging*) or sense (*Word sense disambiguation*)
- Syntactical ambiguity (*parsing*)
- Ambiguity in interpreting a statement (*sentiment classification, natural language inference*)

Plus, implicit knowledge makes things difficult. . .

- Background, commonsense knowledge
- Contextual knowledge

NLP Applications

Let's look at a few NLP applications:

- Speech Recognition: we need to know that $\mathbb{P}(\text{'recognize speech'}) > \mathbb{P}(\text{'wreck a nice beach'})$!

NLP Applications

Let's look at a few NLP applications:

- Speech Recognition: we need to know that $\mathbb{P}(\text{'recognize speech'}) > \mathbb{P}(\text{'wreck a nice beach'})$!
- Machine Translation: $\mathbb{P}(\text{'I like avocados'}) > \mathbb{P}(\text{'I like lawyers'})$

NLP Applications

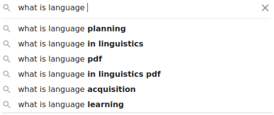
Let's look at a few NLP applications:

- Speech Recognition: we need to know that $\mathbb{P}(\text{'recognize speech'}) > \mathbb{P}(\text{'wreck a nice beach'})$!
- Machine Translation: $\mathbb{P}(\text{'I like avocados'}) > \mathbb{P}(\text{'I like lawyers'})$
- Grammar/Spelling Correction: $\mathbb{P}(\text{'He likes avocados'}) > \mathbb{P}(\text{'He like avocados'})$

NLP Applications

Let's look at a few NLP applications:

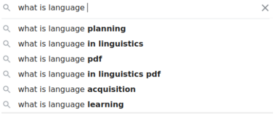
- Speech Recognition: we need to know that $\mathbb{P}(\text{'recognize speech'}) > \mathbb{P}(\text{'wreck a nice beach'})$!
- Machine Translation: $\mathbb{P}(\text{'I like avocados'}) > \mathbb{P}(\text{'I like lawyers'})$
- Grammar/Spelling Correction: $\mathbb{P}(\text{'He likes avocados'}) > \mathbb{P}(\text{'He like avocados'})$
- And directly for ranking:



NLP Applications

Let's look at a few NLP applications:

- Speech Recognition: we need to know that $\mathbb{P}(\text{'recognize speech'}) > \mathbb{P}(\text{'wreck a nice beach'})$!
- Machine Translation: $\mathbb{P}(\text{'I like avocados'}) > \mathbb{P}(\text{'I like lawyers'})$
- Grammar/Spelling Correction: $\mathbb{P}(\text{'He likes avocados'}) > \mathbb{P}(\text{'He like avocados'})$
- And directly for ranking:

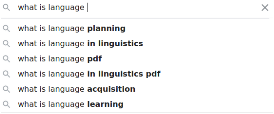


- Goal: **assign the larger probability to the best option !**

NLP Applications

Let's look at a few NLP applications:

- Speech Recognition: we need to know that $\mathbb{P}(\text{'recognize speech'}) > \mathbb{P}(\text{'wreck a nice beach'})$!
- Machine Translation: $\mathbb{P}(\text{'I like avocados'}) > \mathbb{P}(\text{'I like lawyers'})$
- Grammar/Spelling Correction: $\mathbb{P}(\text{'He likes avocados'}) > \mathbb{P}(\text{'He like avocados'})$
- And directly for ranking:



- Goal: **assign the larger probability to the best option !**
- It is also necessary for Optical Character Recognition, Information Retrieval, Summarization, Question Answering... and many other tasks (including anything that necessitate **Text Generation**)

Probabilities on language: Language Modeling

- A **Language Model** (**LM**) estimates the probabilities of text sequences.
  How ?

  $$\mathbb{P}(\text{What is language modeling ?}) = \ ?$$

  Usually, we *count* ! Let's assume that we divide our sentence in **words**.

- **Chain rule**: decompose in smaller parts:

$$\mathbb{P}(\text{What is}) = \mathbb{P}(\text{What})\mathbb{P}(\text{is} \mid \text{What})$$

$$\mathbb{P}(\text{What is language}) = \mathbb{P}(\text{What is})\mathbb{P}(\text{language} \mid \text{What, is})$$

$$\mathbb{P}(\text{What is language}) = \mathbb{P}(\text{What})\mathbb{P}(\text{is} \mid \text{What})\mathbb{P}(\text{language} \mid \text{What, is})$$

- **Formally**,

$$\mathbb{P}(w_1, ..., w_m) = \prod_{i=1}^{m} \mathbb{P}(w_i | w_{<i})$$

Some terminology

- What counts as a word ? Some basic definitions:
    - We begin from a **corpus** (plural **corpora**): computer-readable collection of text
    - Word **types** are the number of *distinct* words in a corpus - usually grouped in a set, the **vocabulary** (noted $\mathcal{V}$).
    - Word **tokens** are *instances* of types in the running text

- Reminders: **tokenization** allows to segment the corpus into word tokens.
    - It determines the vocabulary $\mathcal{V}$.
    - Spoiler: the size of the vocabulary $|\mathcal{V}|$ has a huge impact on the cost of computation !
    - Word normalization is a way to reduce $|\mathcal{V}|$.

Statistics on words: n-grams

- A **n-gram** is a sequence of n successive items. For *words*, we can see them as an ordered sequence looking $n-1$ words into the past.

- Just using word frequencies: **1-grams** (*unigrams*)
- If you know the previous word: **2-grams** (*bigrams*)
- If you know the two previous words: **3-grams** (*trigrams*)
  $\rightarrow$ of course, this depends on the *tokenization* !

- How would these apply to previous examples ?

- An interesting tool: Google Ngram Viewer

Simplifying assumptions

- We use the simplifying **Markov** assumption: the probability depends upon a fixed number of words (the *order*).

$$\text{(Order 1)} \quad \mathbb{P}(\text{modeling}|\text{What is language}) \approx \mathbb{P}(\text{modeling}|\text{language})$$

$$\text{(Order 2)} \quad \mathbb{P}(\text{modeling}|\text{What is language}) \approx \mathbb{P}(\text{modeling}|\text{is, language})$$

$$\vdots$$

$$\text{(Order k)} \quad \mathbb{P}(w_i|w_1, ..., w_{i-1}) \approx \mathbb{P}(w_i|w_{i-k+1}, ..., w_{i-1})$$

- With the chain rule, we obtain simple language models:

$$\text{(Order 0: Unigram)} \quad \mathbb{P}(w_1, ..., w_m) \approx \prod_{i=1}^{m} \mathbb{P}(w_i)$$

$$\text{(Order 1 : Bigram)} \quad \mathbb{P}(w_1, ..., w_m) \approx \mathbb{P}(w_1) \prod_{i=2}^{m} \mathbb{P}(w_i|w_{i-1})$$

Usual models

- The most used is arguably the trigram:

$$\mathbb{P}(w_1, ..., w_m) \approx \mathbb{P}(w_1)P(w_2|w_1)\prod_{i=3}^{m}\mathbb{P}(w_i|w_{i-2}, w_{i-1})$$

- Then, we obtain the following:
  $\mathbb{P}$(What is language modeling) $\approx \mathbb{P}$(What) $\times\ \mathbb{P}$(is | What) $\times$
  $\mathbb{P}$(language | What, is) $\times\ \mathbb{P}$(modeling | is, language)

- Models are often extended to 4-grams, 5-grams. More is not really sustainable (*Why ?*)
  - Due to long-term dependancies, this is in general insufficient to model language !
  - However, n-gram models are still used in numerous applications

Model definition

- A n-gram model is **parametric**, with the number of parameters $\sim O(|\mathcal{V}|^n)$
- The model is defined by *every possible conditional probability that can be computed given its vocabulary* $\mathcal{V}$. Hence, a n-gram model is defined by the values of:

$$\theta = \{\mathbb{P}(w_n | w_1, ..., w_{n-1}), \forall (w_1, w_2, ..., w_n) \in \mathcal{V}^n\}$$

- How to compute the probability of a word w_n given a context of previous words $h = (w_1, w_2, ..., w_{n-1})$?

$\rightarrow \forall (w_1, w_2, ..., w_n) \in \mathcal{V}^n$

$$\mathbb{P}(w_n | w_{1:n-1}) = \frac{C(w_1, ..., w_{n-1}, w_n)}{C(w_1, ..., w_{n-1})} = \frac{C(w_1, ..., w_{n-1}, w_n)}{\sum_{w' \in \mathcal{V}} C(w_1, ..., w_{n-1}, w')}$$

Indeed, the sum of all n-gram counts that start with a given sequence $w_{1:n-1}$ is equal to the $(n-1)$-gram count for that sequence !

Bigram model: an example

- Let us assume the following corpus: what are the **bigram probabilities** ?

I walked down the street
I walk down the the avenue
I walked down the avenue
I ran down the street
I walk down the city

- Let's count ! First, what is the vocabulary ?

Bigram model: an example

- Let us assume the following corpus: what are the **bigram probabilities** ?

I walked down the street
I walk down the the avenue
I walked down the avenue
I ran down the street
I walk down the city

- Let's count ! First, what is the vocabulary ?

$$\rightarrow \mathcal{V} = \{\text{I}, \text{the}, \text{down}, \text{walked}, \text{street}, \text{avenue}, \text{walk}, \text{ran}, \text{city}\}$$

Counting bigrams

- We can compute the count matrix $\mathbf{C}$, where $\mathbf{C}_{i,j} = C((w_i, w_j))$:

	I	the	down	walked	street	avenue	walk	ran	city
I	0	0	0	2	0	0	2	1	0
the	0	1	0	0	2	2	0	0	1
down	0	5	0	0	0	0	0	0	0
walked	0	0	2	0	0	0	0	0	0
street	0	0	0	0	0	0	0	0	0
avenue	0	0	0	0	0	0	0	0	0
walk	0	0	2	0	0	0	0	0	0
ran	0	0	1	0	0	0	0	0	0
city	0	0	0	0	0	0	0	0	0

Estimating parameters

- Applying the previous formula, we compute the matrix $\mathbf{P}$, where $\mathbf{P}_{i,j} = \mathbb{P}_\theta(w_j|w_i)$:

	I	the	down	walked	street	avenue	walk	ran	city
I	0	0	0	2 / 5	0	0	2/5	1/5	0
the	0	1/3	0	0	1/3	1/3	0	0	1/6
down	0	1	0	0	0	0	0	0	0
walked	0	0	1	0	0	0	0	0	0
street	0	0	0	0	0	0	0	0	0
avenue	0	0	0	0	0	0	0	0	0
walk	0	0	1	0	0	0	0	0	0
ran	0	0	1	0	0	0	0	0	0
city	0	0	0	0	0	0	0	0	0

- What can we notice ? What are some potential issues ?
- What probabilities does this model give to the sentences of the corpus ?

Model evaluation

- **Extrinsic** evaluation:
  - Apply the models that we want to compare on a task, and use the related metric ! (e.g, *speech recognition* and *word error rate*) $\rightarrow$ Time-consuming, task-dependant.
- **Intrinsic** evaluation: **Perplexity**
  - A measure of uncertainty: how surprised is the model by the data ?

$$\text{Perplexity}(w_1, ..., w_m) = \sqrt[m]{\prod_{i=1}^{m} \frac{1}{\mathbb{P}_\theta(w_i | w_{i-n+1}, ..., w_{i-1})}}$$

  - Can be interpreted as a branching factor
  - But is tied to a particular vocabulary $\mathcal{V}$

Estimating parameters: theoretical consideration

- Estimating parameters = getting **counts** from a corpus, normalizing them into **probabilities** = Computing the *relative frequency* !

- It is by definition the **Maximum Likelihood Estimation** (**MLE**).
  - This maximizes the likelihood of the corpus as a whole given the constraints of the model
  - Of course, our bigram model is bad... but it gives the corpus *the best likelihood possible* for any a bigram model

- Perplexity is a simple function of the cross-entropy:

$$\text{Perplexity}(w_1, ..., w_m) = 2^{-\frac{1}{m}\sum_{i=1}^{m}\log(\mathbb{P}_\theta(w_i|w_{i-n+1},...,w_{i-1}))}$$

- The model maximizes the likelihood of the data used to estimate its parameter (training data), hence it minimizes its perplexity
  - We want it to give low perplexity to new (testing) data

Perplexity: an example

- What is the perplexity of the **bigram model** we previously defined on its **training data** ?
- Assume an **unigram model** estimated computing relative frequencies with the same vocabulary, on the same data: what is its perplexity ?
- Now, assume that the model is unigram, but attributes **uniform probabilities** to each word: what is its perplexity ?

- How do you interpret those results ? Can we compare those perplexities ?

- What happens if we try to compute the probabilities / perplexity of the following sentences ?

I walked down the city
I ran down the avenue
I run down the street

Issues with symbolic representations

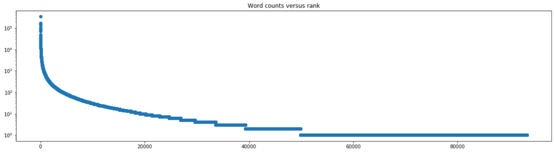
Generalization

- Huge issue: **Zero probability n-grams**
  - In a corpus, occuring n-grams represent a very small part of the possible n-grams.

In Shakespeare's work (Example from Dan Jurafsky)

- 884,647 tokens and a vocabulary of 29,066 words
- 300K out of the 844M bigrams appear $(0,04\%)$!

- Tied to **Zipf's Law**: $frequency \propto 1/rank$



Reducing the vocabulary

We can choose a way group words, to put together **many tokens in one type**:

- Removing upper-cases ? $\rightarrow$ depends on the task ! Information retrieval does not need them - but information extraction does !

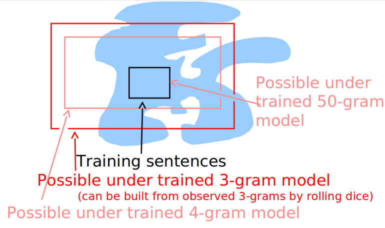
- **Lemmatization**: represent words as their lemma
  - Done by morphological parsing
  - Necessary for some languages: e.g, Turkish.
  - Tools - we can also use here **NLTK**
- Other process: **stemming** - crudely cutting words

- Of course, we can also remove frequent words *which may be useless*: **stopwords**

$\rightarrow$ But all of this necessitates to establish and maintain **list of rules**

Parenthesis : Acceptability

From Jason Eisner's NLP class



- Combining acceptable n-grams can easily give unacceptable English.
- Is there a way to rule them out ?
  - Is increasing the amoung of training text enough ?
- Are n-grams enough ?

- Direction not taken in this class: *grammar modeling*
- We'll stick to statistics, within the current paradigm (solution: scaling *way way* up)

Better language models: Smoothing

- To avoid zero probability n-grams, we use smoothing: the idea is to **'discount' a little of probability mass** from observed events and **re-allocate it** among the unseen ones.
  - Intuition: after the word 'the', our bigram model gave...
  - We can use very simple and straightforward smoothing, called **'add-one'**, but there exist smart and efficient ways of doing it.

Add-1 smoothing

Back to the probability estimated by relative frequency:

$$\mathbb{P}(w_n|w_{1:n-1}) = \frac{C(w_1, ..., w_{n-1}, w_n)}{C(w_1, ..., w_{n-1})}$$

To avoid the *numerator count being zero*, we simply **add one to every possible count**. This is also called **Laplace Smoothing**:

$$\mathbb{P}(w_n|w_{1:n-1}) = \frac{C(w_1, ..., w_{n-1}, w_n) + 1}{C(w_1, ..., w_{n-1}) + |\mathcal{V}|}$$

- Avoids making never-seen sequences impossible.
- What happens if the corpus is small, but the vocabulary large ?
    - Think about the balance between *known* and *novel events*.
    - You can replace 1 by a small δ. (But how to pick it ?)
- Overall, this is helpful - but why treating **all novel events as the same** ?

Advanced smoothing

- **Backoff:**
  - When you don't have enough information at order n, back-off to smaller orders:

$$\mathbb{P}(\text{is language}) = 0 \Rightarrow \mathbb{P}(\text{is language}) \approx \mathbb{P}(\text{language})$$

  - Use the *backed-off* probability instead of δ !

- **Interpolation:** Instead of replacing by a smaller order, interpolate between all smaller orders:

$$\mathbb{P}(\text{modeling}|\text{is, language}) = \lambda_1 \mathbb{P}(\text{modeling}|\text{language}) + \lambda_2 \mathbb{P}(\text{modeling})$$

- 'Modern' methods:
  - **Kneser-Ney**: very popular - backoff + *discounting*
  - Other smoothing methods: *Witten-Bell*, *Good Turing*

Better document representations: TF-IDF

What would be a better way than directly removing frequent-but-not-significant words (stopwords) ?

- Idea 1: instead of using count $c(w, d)$ of word w in document d, use a **smoothed** term frequency $\text{TF}(w, d) = \log_{10}(c(w, d) + 1)$
- Idea 2: give higher weight to words that occur in only a few documents, using their inverse document frequency. Noting $cd(w)$ the count of documents w appears in and N the total number of documents,

$$\text{IDF}(w) = \log_{10}\left(\frac{N}{cd(w)}\right)$$

The weight given to word w in document d is $\text{TF}(w, d) \times \text{IDF}(w)$

- What happens if a word is present in every document ?

Better word representations: using word relationships

Let's assume that $\mathbf{w}_1$ and $\mathbf{w}_2$ have high cosine similarity:

- That implies they have the same distributional context !
- What can we know about their relationship: *synonym, antonym* ?

Now, let's assume that $c(w_1, w_2)$ is high. What could it imply ?

- Examples:

	I	$\neg$ I
the	123	1245
$\neg$ the	987	9437

	Computer	$\neg$ Computer
Semantics	7	19
$\neg$ Semantics	31	11735

- What values should we look at to understand how *significant* co-occurence counts are ?

Pointwise Mutual Information

Pointwise Mutual Information (PMI): Evaluate how **unexpected** is two words co-occuring !

- PMI: log-ratio of the *joint probability* of two words and the *product of their marginals*:

$$\text{PMI}(w_i, w_j) = \log\left(\frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right)$$

- This probability corresponds to the observed frequency:

$$\text{PMI}(\mathbf{M}, w_1, w_2) = \log\left(\frac{M_{w_1,w_2} \times \left(\sum_{k=1}^{n}\sum_{l=1}^{n} M_{k,l}\right)}{\left(\sum_{l=1}^{n} M_{w_1,l}\right) \times \left(\sum_{k=1}^{n} M_{k,w_2}\right)}\right)$$

- Values range in $[-\infty, \infty]$ but negative values are unreliable (why ?)
- Extreme values are still an issue (division by small values) !
- As TF-IDF, can be used to compute similarities.

Other difficulties with symbolic representations

Two main issues:

- Models are huge ($|\mathcal{V}|^n$) causing memory and computation issues
- Despite smoothing, models are sparse and can't generalize well.

- We can avoid both of these issues by using **dense**, **distributed** representations: we would like to replace:

this . . . . . . *by this*.

$$\mathbf{word} = \boxed{\begin{matrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}} \in \mathbb{N}^{|\mathcal{V}|} \qquad \mathbf{word} = \boxed{\begin{matrix} 0.212 \\ 0.792 \\ -0.177 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{matrix}} \in \mathbb{R}^d$$

- How to build such representations to **embed linguistic information** ?

Better document representations: dimension reduction

- Cosine similarity on symbolic representations does not work well: should all dimensions (represented by words) matter the same ?
- How can we represent documents by doing **more than counting words** ?

Idea: take advantage of the latent structure in the association between the set of words and the set of documents

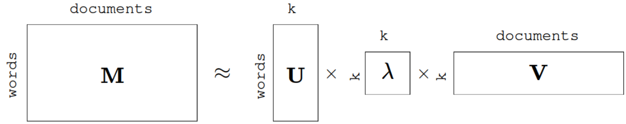
- First method: linearly reduce the dimension to put forward higher-order relationships
  $\rightarrow$ Use Singular Value Decomposition (SVD)

$$\mathbf{M} = \mathbf{U}\lambda\mathbf{V}^{\intercal}$$

  - λ diagonal matrix, with eigenvalues - ordered
  - $\mathbf{U}, \mathbf{V}$ orthogonal; eigenvectors
  - Keep the k first columns of $\mathbf{U}$, for the k largest eigenvalues, to obtain embeddings $\in \mathbb{R}^k$
  - But very costly (quadratic in memory, cube in flops)

Latent Semantic Analysis

This method is called **Latent Semantic Analysis**:



- The new space is interpreted as **topics**: this is the first method for *topic modeling*
- Reminder: SVD rotates the axis along directions of largest variations among the documents (generalized least-squares method)
- Useful for information retrieval, but also if we need **higher-order features than word counts**
- Can help to represent documents in topic space for classification !
  - Besides the cost, it must be re-run if you add new documents.

More on Topic Modeling

Other topic modeling methods: mostly **generative models** - we model the generation of words as *random, following a distribution*

- **Probabilistic LSA**: the generation of words follows a mixture of *conditionally independant multinomial distributions*, given topics:

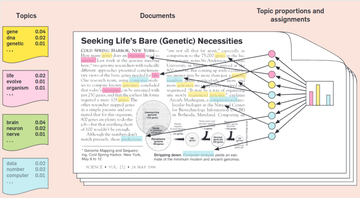
$$P(w,d) = \sum_t P(t)P(d|t)P(w|t) = P(d)\sum_t P(t|d)P(w|t)$$

  where $P(d|t)$ relates to V and $P(w|t)$ to U: non-negative values

- **Latent Dirichlet Allocation (LDA)**:

  We assume that the topic distribution has a *Dirichlet prior* (a family of continuous multivariate distributions)

Probablistic Topic Models (Blei, 2012)



Summary of difficulties, solutions - and what's next

- Being of the **size of the vocabulary** $\mathcal{V}$, vectors can get huge: memory and computation issues
- Vectors are **sparse** - many empty dimensions
- When computing similarity, should all dimensions (represented by other words) matter the same ?
- We only have access to direct relationships between words
- Skewed frequency is always a challenge

Three types of solutions - which can be combined:

- Reduce the vocabulary (*lemmatization*, *removing stop words*. . .) and smoothe the probability
- Next idea: **re-weight** the vectors (*TF-IDF*, *PMI*)
- And then: obtain **dense representations**
  - With linear methods (*topic modeling*)
  - With neural models: for next time !