An *n*-gram model of text

PREREQUISITES

- sets (cardinality)

So far we have mostly studied *n*-gram models for linguistic reasons. These models are very simple, but can capture a fair amount of phonotactic and morphotactic conditions. This in turn shows that these conditions are very simple. But *n*-gram models aren't limited to linguistic theorizing. In fact, they are mostly used in more applied domains.

1 Unigrams for text classification

Suppose your task is to classify texts, for example as part of a search engine. Ideally, this classification would proceed by carefully reading the entire text, interpreting it, and distilling its core themes through some high-level analysis. But that requires a lot of time and skill, and may simply not be feasible in practice. How does one adequately summarize, say, the 1130 pages of Robert Musil's *The Man Without Qualities*, or Grigori Perelman's proof of the Poincaré conjecture? Whatever the right answer, it probably isn't something that can be done quickly and automatically. And while one may be able to pay experts to work on these singular accomplishments, it's much harder to find somebody to summarize papers on cell biology because there are so many published every day. With internet websites, human summarization is completely impossible given how often they are updated and how many new ones are created every minutes. So instead computers have to do the job, and since we haven't figured out a way yet to get computers to understand text, the models are necessarily simple and focussed on surface features. Virtually all of them build on *n*-grams, the core idea being that the meaning of a text can be equated with the words that occur in it.

Let us look at a particularly simple way of formalizing this idea, one where we ignore how often certain words occur. We will also ignore capitalization, as is commonly done in this model. For example, converting the mini-text *Only John could like John* to a set of unigrams (i.e. 1-grams) only preserves the information that the text contains the words *only*, *john*, *could*, and *like*. A few more examples are shown below using the programming language Python.

```
import re
from pprint import pprint

def set_of_words(string):
    """Convert a string to a set of words."""
    tokens = [word for word in re.split("\W", string.lower()) if word]
```

```
    print("Input:", string)
    pprint(set(tokens))
    print("\n")

>>> set_of_words("John is John, that much is obvious!")
Input: John is John, that much is obvious!
{'is', 'obvious', 'john', 'much', 'that'}

>>> set_of_words("The man and the woman are husband and wife.")
Input: The man and the woman are husband and wife.
{'wife', 'the', 'and', 'husband', 'woman', 'are', 'man'}

>>> set_of_words("Police police police police police.")
Input: Police police police police police.
{'police'}
```

A search engine, for instance, could use this model to convert any given website to a set of words. When the user enters a query, e.g. *fed my Gremlin after midnight*, the search engine could convert the query to a set of words, too, and then check which websites have a similar set of words.

Example 1.
Suppose that our search engine has only indexed three websites so far, both of which contain very little text.

- This website is a website about the movie Gremlins.
- I never feed my cat after midnight.
- There's a Gremlin in my closet.

The corresponding sets of unigrams are as follows:

- {this, website, is, a, about, the, movie, gremlins}
- {i, never, feed, my, cat, after, midnight}
- {there's, a, gremlin, in, my, closet}

The query *fed my Gremlin after midnight* is mapped to the set {fed, my, gremlin, after, midnight}. We can now use intersection to see how much each website overlaps with the query.

- ∅
- {my, after, midnight}
- {gremlin, my}

The largest overlap is with the second website— or in more mathematical terms, the intersection of our query with the second website has the largest cardinality.

Exercise 1.
This model is hopelessly olibvious about the connections between words. Give two examples of important connections that this model missed. Would this have changed which website is a better fit?

The general idea of the model is simple enough, and as you can see even the implementation in a programming language is straight-forward. Note that here we are no longer dealing with *n*-gram grammars. The task involves no notion of well-formedness. Instead, unigrams are used as a compressed **representation** of the text, and all reasoning is done over this compressed representation.

2 Unigrams: the pros and cons

The main advantage of the set-of-words model of texts is its simplicity — determining the meaning of a text only requires a very simple function that maps strings of words to sets of words. But while practical applications prize simplicity and efficiency even to the detriment of accuracy, the set-of-words model is just too simple for the vast majority of real-world applications. There are at least three problems:

1. Context is not taken into account at all, even within individual sentences. Among other things, *The dog bit the man* and *The man bit the dog* incorrectly receive the same meaning. And along the same lines, *Not every student thinks they should leave* and *Every student thinks they should not leave* are taken to have identical meanings, too.
2. Since we do not count how often words occur, a text that mentions global warming once in passing is taken to cover this topic to the same extent as one that mentions it over a hundred times.
3. The sets are cluttered with uninformative words like *is*, *the*, *of*, and so on.

The first one can be improved by moving from unigrams to *n*-grams. With bigrams, a headline like *man bites dogs* is represented as the set {man bites, bites dogs}, whereas the much less startling *dog bites man* becomes {dog bites, bites man}. Note that we could also include our edge markers $\rtimes$ and $\ltimes$ to clearly identify the first and last word of a sentence. None of this is an adequate representation of context, but it nonetheless works fairly well in practice - though that might just be because the practical problems language technology is asked to solve nowadays are still fairly simple.

Be that as it may, there are still problems 2 and 3 to take care of. We need a way to keep track of how often specific words occurred, and we want to get rid of uninformative words. Those will require a few bells and whistles we haven't seen before, which will be the subject of the next few units.