Positive n-gram grammars

Prerequisites

- basic math (factorial)

We now have a simple model of phonotactics, i.e. what sequences of sounds may occur in the words of a natural language. According to this model, the phonotactic well-formedness of a word can be determined from the chunks that said word is built from. In a trigram model, for instance, a word is ill-formed iff it contains one or more illicit trigrams. We formalize this in terms of a negative n-gram grammar, which is a finite set of illicit n-grams.

Example 1.
Suppose our alphabet, i.e. the set of available symbols, is $\{a, b, c\}$. Then the negative trigram grammar $\{aac, abc, acc\}$ only permits strings where no symbol is directly sandwiched between a and c.

The model looks rather promising as it can handle a variety of phenomena that have been studied extensively by linguists: word-final devoicing, intervocalic voicing, local assimilation, and various stress rules.

Exercise 1.
For each one of the following phenomena, write a negative n-gram grammar that handles it correctly. For some of them, you have to rephrase the phenomenon as a phonotactic constraint first.

- **intervocalic voicing**: voiceless fricatives (assume s and f) may not occur between vowels (assume a, i, u)
- **local assimilation**: n must be m before b or p
- **local dissimilation**: rVr becomes lVr, where V is a, i, or u
- **penultimate stress**: in words with at least two syllables, stress falls on the last but one syllable (assume that words are strings of stress syllables ($\acute{\sigma}$) and unstressed syllables (σ))

Since the model seems to work well for phonotactics, it is tempting to expand it to other domains of language. But as we will see next, this reveals certain shortcomings of the negative grammar format.

1 Morphotactics

Just like phonotactics regulates the linear order of sounds in a word, **morphotactics** regulates the linear order of **morphemes**. Morphemes consist of multiple sounds and are the building blocks of words (linguists, please keep in mind that once again we won't distinguish between morphemes, morphs, and allomorphs). For example, *denaturalization* is built from the morphemes *de-*, *nature*, *-al*, *-ize*, and *-ation*. Morphemes cannot be combined willy-nilly, they have to follow a specific order. In the case of *denaturalization*, no other order is possible. Even though the word is built up from 5 elements, which could be arranged in $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$ distinct ways, only one of them is actually allowed by English. So morphotactics defines a very tight rule system for how elements may be ordered in a word, much tighter than phonotactics, where individual sounds have more leeway as to where they occur in a word.

Let's see if we can write a negative *n*-gram grammar that allows for *denaturalization* but forbids all illicit orders, e.g. *naturedeationizal*. First, we have to pick the basic building blocks for the *n*-grams. For phonotactics, we used *n*-grams where each symbol is a sound, but this is too fine-grained for morphotactics. Instead, we will use *n*-grams where each symbol is a morpheme. So *-ize -ation* is a bigram, not an 11-gram that consists of 8 letters, 2 hyphens, and 1 space.

```
chunk = "-ize -ation"
print("{} with characters as symbols: {}-gram".format(chunk, len(chunk)))
print("{} with sounds as symbols: {}-gram".format(chunk, len("izaSon")))
print("{} with morphemes as symbols: {}-gram".format(chunk, len(("-ize", "-ation"))))
```

Exercise 2.
For each one of the following *n*-grams, say how large it is depending on what one chooses as the basic symbols that *n*-grams are built from. Possible choices for building blocks are typed characters, morphemes, or words. Not all choice may be appropriate in each case.

- *de-*
- *mpi*
- *John likes Mary*

With each morpheme as a separate symbol, it should be straight-forward to design a negative grammar to generate *de-nature-al-ize-ation* but none of the other orders. Let's first write down the conditions in plain English:

1. start with *de-*,
2. *de-* is followed by *nature*,
3. *nature* is followed by *-al*,
4. *-al* is followed by *-ize*,
5. *-ize* is followed by *-ation*,

6. end with *-ation*.

Easy peasy, so let's write it down as a negative grammar. Here's the list of the forbidden *n*-grams that correspond to each one of the conditions.

1. start with *de-*
    1. ⋊ *$*
    2. ⋊ *nature*
    3. ⋊ *-al*
    4. ⋊ *-ize*
    5. ⋊ *-ation*
2. *de-* is followed by *nature*
    1. *de- $*
    2. *de- de-*
    3. *de- -al*
    4. *de- ize*
    5. *de -ation*
3. *nature* is followed by *-al*
    1. *nature* ⋉
    2. *nature de-*
    3. *nature nature*
    4. *nature -ize*
    5. *nature -ation*
4. *-al* is followed by *-ize*
    1. *-al* ⋉
    2. *-al de-*
    3. *-al nature*
    4. *-al -al*
    5. *-al -ation*
5. *-ize* is followed by *-ation*
    1. *-ize* ⋉
    2. *-ize de-*
    3. *-ize nature*
    4. *-ize -al*
    5. *-ize -ize*
6. end with *-ation*
    1. *-ation de-*
    2. *-ation nature*
    3. *-ation -al*
    4. *-ation ize*

5. *-ation -ation*

Hmm, that didn't turn out as succinctly as one might have hoped.

2 From negative to positive grammars...

The negative bigram grammar above is much larger than one would expect. Perhaps even more problematically, it does not clearly express the relevant generalizations. Intuitively, it would be much more appealing to list what combinations are allowed, rather than forbidden:

1. start with *de-*
    1. $\rtimes$ *de-*
2. *de-* is followed by *nature*
    1. *de- nature*
3. *nature* is followed by *-al*
    1. *nature -al*
4. *-al* is followed by *-ize*
    1. *-al -ize*
5. *-ize* is followed by *-ation*
    1. *-ize -ation*
6. end with *-ation*
    1. *-ation* $\ltimes$

This is a **positive *n*-gram grammar**, where the *n*-grams list what sequences are allowed, rather than forbidden.

Example 2.
The list of bigrams above is $\rtimes$ *de-*, *de- nature*, *nature -al*, *-al -ize*, *-ize -ation*, *-ation* $\ltimes$. If this is interpreted as positive bigram grammar, then only *denaturalization* is well-formed. A string like *nature -al -ize -ation -de* is illicit because it contains the bigram *-ation de-*, which is not part of the positive grammar and thus forbidden. If one adds *nature* $\ltimes$ to the grammar, then *nature* can also be generated.

In positive *n*-gram grammars, all *n*-grams must be of the same length to avoid inconsistencies. That's because with a positive *n*-gram grammar, a word is well-formed iff each one of its *n*-grams is part of the grammar.

Example 3.
Suppose we want to allow both *natural* and *denaturalization*, but not *denatural*. In order to allow the former, the grammar has to contain the bigrams ⋊ *nature*, *nature -al*, and *-al* ⋉. But in combination with the bigrams from the previous example, this would also allow for *denatural*. Instead, then, one might try replacing ⋊ *de-* with the 5-gram ⋊ *de- nature -al -ize*, so that the grammar looks as follows:

- ⋊ *de- nature -al -ize*
- *de- nature*
- *nature -al*
- *-al -ize*
- *-ize -ation*

But then it is unclear how the grammar should be evaluated. If we look at all the 5-grams of ⋊ *de- nature -al -ize -ation*, then only ⋊ *de -nature -al -ize* is part of the grammar and the string is incorrectly ruled out. If we instead look at all the bigrams, then the word is ruled out because ⋊ *de-* is no longer part of the grammar. Either way the mixing of bigrams and 5-grams causes inconsistencies.

Despite the requirement to stick with one fixed length of *n*-grams, positive grammars can be much smaller than negative ones. But the opposite is also true, in particular for mixed negative grammars. It depends on the specific phenomenon.

Exercise 3.
Write both a positive and a negative grammar that each allow only strings of the form *ab*, *abab*, *ababab*, *abababab*, and so on (assume that all symbols are either *a* or *b*). Is one of the two grammars more succinct or general than the other? What if the set of symbols is larger, e.g. *a*, *b*, *c*, and *d*?

Exercise 4.
For each one of the following phenomena, write a positive *n*-gram grammar that handles it correctly. For some of them, you have to rephrase the phenomenon as a phonotactic constraint first.

- **intervocalic voicing**: voiceless fricatives (assume *s* and *f*) may not occur between vowels (assume *a*, *i*, *u*)
- **local assimilation**: *n* must be *m* before *b* or *p*
- **local disimilation**: *rVr* becomes *lVr*, where *V* is *a*, *i*, or *u*
- **penultimate stress**: in words with at least two syllables, stress falls on the last but one syllable (assume that words are strings of stress syllables (*σ́*) and unstressed syllables (*σ*))

Once you're done, contrast the positive grammars against the negative ones

from an earlier exercise. Can you identify some general guidelines for when a positive grammar is preferable to a negative one?

3 . . . and back: Translating between positive and negative grammars

We now have two different kinds of n-gram grammars: positive grammars, and negative grammars. The latter fall into two subtypes, fixed and mixed, but as we have already proved those two are equivalent in the sense that one can freely translate between the two. The same is in fact true for positive and negative grammars.

The idea is very simple. Suppose that your alphabet (i.e. the set of symbols from which strings are built) contains only a and b. Then consider the language $(aba)^+$, which contains *aba*, *ababa*, *abababa*, and so on. The negative grammar generating this language consists of

1. $\rtimes\ltimes$ (no string without any symbols),
2. $\rtimes b$ (don't start with b),
3. aa (don't have a followed by a),
4. bb (don't have b followed by b),
5. $b\ltimes$ (don't end with b).

The positive grammar, on the other hand, contains

1. $\rtimes a$ (you may start with a),
2. ab (a may be followed by b),
3. ba (b may be followed by a),
4. $a\ltimes$ (you may end with a).

Now compare this to the list of all possible bigrams over a, b, and $\$$:

1. $\rtimes\ltimes$,
2. $\rtimes a$,
3. $\rtimes b$,
4. $a\ltimes$,
5. aa,
6. ab,
7. $b\ltimes$,
8. ba,
9. bb.

Notice anything? Each one of those bigrams is either in the negative grammar or in the positive one (but never in both). So in order to convert a positive grammar to a negative one, or the other way round, it suffices to first compute all possible

n-grams and then remove all those that are in the grammar that is to be converted to the opposite polarity.

Example 4.
Suppose our alphabet contains only a and that the only well-formed string is aa. This would be the case if we have a positive trigram grammar containing:

- $\rtimes\rtimes a$
- $\rtimes aa$
- $aa\ltimes$
- $a\ltimes\ltimes$

The set of all possible (and useful) trigrams over the alphabet is as follows:

- $\rtimes\rtimes a$
- $\rtimes aa$
- $aa\ltimes$
- $a\ltimes\ltimes$
- $\rtimes\rtimes\rtimes$
- $\rtimes\rtimes\ltimes$
- $\rtimes\ltimes\ltimes$
- $\ltimes\ltimes\ltimes$
- $\rtimes a\ltimes$
- aaa

Removing all trigrams of the positive trigram grammar leaves us with the following list:

- $\rtimes\rtimes\rtimes$
- $\rtimes\rtimes\ltimes$
- $\rtimes\ltimes\ltimes$
- $\ltimes\ltimes\ltimes$
- $\rtimes a\ltimes$
- aaa

You can verify for yourself that a negative trigram grammar that contains those three trigrams (and no other *n*-grams) can only generate aa over the alphabet $\{a\}$.

```
from itertools import product

def all_ngrams(alphabet, n):
    """Build all n-grams over alphabet."""
    # for n = 0, we want the empty set rather than {''}
    if n == 0:
        return set()
```

```
    else:
        return set(''.join(ngram)
                   for ngram in product(alphabet, repeat=n))

def posneg_conversion(grammar, alphabet, n):
    """Convert between positive and negative n-gram grammars.

    Arguments
    ---------
    grammar: set
        grammar that is to be converted
    alphabet: set
        alphabet for the grammar
    n: int
        length of n-grams
    """
    return all_ngrams(alphabet, n) - grammar

neg_gram = set(['aa', 'ba'])
alphabet = set(['a', 'b', '$'])
pos_gram = posneg_conversion(neg_gram, alphabet, 2)

print("The original grammar is:")
print(neg_gram)
print("The opposite polarity version is:")
print(pos_gram)
```

Exercise 5.
English allows for *nature*, *natural*, *naturalize*, *denaturalize*, *naturalization*, and *denaturalization*, but not *denature* or any of misordered forms like *naturizalation*. Write a grammar that generates all the well-formed forms but none of the ill-formed ones. It is up to you whether you want to use a positive or a negative grammar. If you use a negative grammar, it can be in the mixed format, with *n*-grams of varying lengths.

4 An important take-home message

The next section will give a formal proof that this simple conversion strategy will always result in an equivalent grammar. By "equivalent" we mean that the two grammars generate exactly the same strings. But beyond pure math, there is an important insight here that will be with us for pretty much the rest of the course: one and the same thing can be specified in many different ways. Depending on one's

criteria, one way may be better than another. In some cases, a positive grammar may be smaller than a negative one. But for some phenomena it is the other way round, and negative grammar also has the advantage that they can be made more compact by using a mixed format instead of a fixed length for all *n*-grams.

There's many examples of this kind of interdefinability in mathematics. Logical formulas, for example, can be put into a normal form that is harder to read for humans but easier to implement for computers. So-called **finite-state automata** can be viewed as a special case of Boolean matrix multiplication (we'll talk about this one in quite some detail). This may seem bewildering to the linguists among you. Linguists like to talk about *the* grammar, *the* feature system, *the* constraints of the grammar, as if those were concrete objects of a singular nature — like a chair is a chair is a chair. Linguistics is driven by the search for *the* correct description of linguistic knowledge. Linguists want the "source code" of the language program that runs in the human brain, not just any implementation that behaves the same. But this quest for *the* correct specification cannot work for abstract concepts, and all linguistic concepts are abstract. When dealing with abstract ideas, you want to be able to conceptualize them in as many distinct ways as possible. True understanding comes from the ability to describe one and the same thing in many different ways, each one with its unique advantages and its unique opportunities for new insights.

5 Recap

- A positive *n*-gram grammar is a finite list of allowed *n*-grams.
- Positive grammars can be converted to negative grammars, and the other way round.
- Having multiple descriptions of the same thing is a boon, not a bane.