N-gram models of grammaticality

This unit presents a very simple model of language. As you will see, even simple models can give rise to complex questions, questions that cannot easily be answered without math. It might not look like the math you know from high school, but it is still math (some might say it is the high school math that barely qualifies as math because it is about calculating rather than reasoning).

If you already have some mathematical experience, you might want to skip two sections ahead and only come back here to see the linguistic motivation for the mathematics. And the trained linguist may find some of the content here overly simplistic, but we will move on to more intricate problems as our mathematical toolkit grows.

A simple fact about English

English has a large number of different sounds, but not all combinations are possible. For instance, *blink* is a word put *kbinl* is not. And *kbinl* will never be a word of English because it does not obey English **phonotactics**, i.e. the laws that govern the sequence of sounds. This is what separates *kbinl* from *kobinal* or *karbinolium*. None of them are existing words of English, but *kobinal* and *karbinolium* are potential words. We can give them a meaning, start using them, and no native speaker of English will have a problem picking them up. That is because their phonotactics are well-formed.

The knowledge of English phonotactics is what allows native speakers to coin new terms that fit in with the rest of the vocabulary. It is also what makes it hard for English speakers to learn other languages. German, for example, is very happy to start a word with *k* and *n*, as in *Knoblauch*, the German word for garlic. English has words that are spelled with *kn* at the start, like *knight*, but the *k* is never pronounced. As far as phonotactics is concerned, *knight* and *night* are the same word. Phonotactics is one of the most basic aspects of natural language. By natural language I mean languages like English, Chinese, Tongan, Inuktitut, various dialects of Italian, or the specific language that you grew up with. This is in contrast to formal languages, which were designed by humans, e.g. Esperanto, Klingon, or the programming language Python. Linguists we want to precisely formulate the laws of natural language phonotactics. They want to do this at both the language-specific level and across languages:

- **Language specific**: What are the phonotactics of English? German? Language X?
- **General**: What is a possible phonotactic system? What shape can the laws of phonotactics take? What kind of logically conceivable systems of phonotactics can never occur in a natural language?

In order to tackle these questions in a precise manner, we need a formal model of phonotactics. What might that be? It is a safe assumption that a word is phonotactically well-formed iff none of its subparts are ill-formed. So *kobinal* is a possible word because there is nothing wrong with a word that starts with *kobina*, has *obina* in the middle, and ends with *obinal*. But this just raises the question why *kobina* is an acceptable start, and *obina* is a well-formed middle, and *obinal* is an acceptable end. Well, because their subparts are fine. For example, *kobina* is fine because a word can start with *kobin* and have *obin* in the middle and *bina* at the end. Then why

can a word start with *kobin*? Well, because its subparts are fine: a word can start with *kobi*, have *obi* in the middle, and *obin* at the end. As you can already see, we can play that game for a long time, breaking well-formed parts into smaller well-formed parts.

But eventually, things will break down. If we want to know why *ko* is a good start, it's not enough to say that words can start with *k* and have *o* in the middle. Because it is also true that words can start with *k* and have *b* in the middle (as in *kobinal*), yet *kbinal* is not okay even though it starts with *k* and has a *b* in the middle. By looking at individual sounds, we have decomposed things too much to capture the phonotactics of English. So let's take one step up and consider pairs of consecutive sounds. In more technical terms, let's describe English phonotactics in terms of **bigrams**.

Bigrams

Extracting bigrams

A bigram is a string of two elements. For phonotactics, the elements are sounds (to all trained linguists: we do not need to distinguish between phones, allophones, or phonemes yet, so please don't get your panties in a twist). Examples of bigrams include *ko*, *ob*, or *bn*. A given word's **set of bigrams** is the collection of all bigrams that occur in it, without repetitions.

Example 1 The bigrams of *kobinal* are *ko*, *ob*, *bi*, *in*, *na*, and *al*. The bigrams of *banana* are *ba*, *an*, and *na*.

```
def bigrams(word):
    return sorted(list(set(''.join(bigram)
                           for bigram in zip(word,word[1:]))))

def bigram_print(word):
    print("The bigrams of", word, "are:")
    print(bigrams(word))

bigram_print("kobinal")
bigram_print("banana")
# try some words of your own
bigram_print("wordone")
bigram_print("wordtwo")
```

Exercise 1 Consider the word *supercalifragilisticexpialidocious*. For each one of the following, say whether it is a bigram of the word.

1. fr
2. z
3. doci
4. pail

5. sit
6. super

Adding edge markers

One shortcoming of this simple notion of bigrams is that one cannot tell which bigrams occurred at the beginning and the end of the word. For example, *ababa* and *babab* have the same bigrams, *ab* and *ba*.

```
def bigrams(word):
    return sorted(list(set(''.join(bigram)
                           for bigram in zip(word,word[1:]))))

def bigram_print(word):
    print("The bigrams of", word, "are:")
    print(bigrams(word))

bigram_print("ababa")
bigram_print("babab")
```

But for phonotactics it is actually important to know how a word starts and how it ends. We have to make some changes to capture this information with bigrams. Concretely, we will add an edge marker $. Then *ababa* will be *$ababa$*, and *babab* will be *$babab$*. Now one can tell clearly which bigrams occurred at the start and the end.

Example 2 To calculate the bigrams of *kobinal*, we first expand it to *$kobinal$*. Then we extract bigrams as usual, giving us the following list: *$k*, *ko*, *ob*, *bi*, *in*, *na*, *al*, *l$*.

```
def bigrams(word):
    word = "$" + word + "$"
    return sorted(list(set(''.join(bigram)
                           for bigram in zip(word,word[1:]))))

def bigram_print(word):
    print("The bigrams of", word, "are:")
    print(bigrams(word))

bigram_print("ababa")
bigram_print("babab")
```

Exercise 2 Consider once more the word *supercalifragilisticexpialidocious*. Which one of the following is among its bigrams (with edge markers):

1. fr
2. z

3. $$
4. $s
5. s$$

Bigram grammars

So now that we have a firm grasp of bigrams, it is actually fairly easy to formulate our first hypothesis about natural language phonotactics: every phonotactic system is described by a set of forbidden bigrams. If a word contains at least one forbidden bigram, it is illicit. We call such a collection of forbidden bigrams a **negative bigram grammar**.

If our hypothesis is correct, then it should be possible to write down a collection of bigrams for English so that all phonotactically well-formed words are allowed, and only those. Every ill-formed word must contain at least one forbidden bigram.

Example 3 Contrast the well-formed *kobinal* against the ill-formed *kbin*. If *kb* is a forbidden bigram of English, then *kbin* is ill-formed. In order for *kobinal* to be well-formed, none of the following bigrams may be forbbiden: *$k*, *ko*, *ob*, *bi*, *in*, *na*, *al*, *l$*.

As the example above shows, forbidding the bigram *kb* rules out *kbin* as an illicit word for English. But there is a problem: *kb* does occur in some well-formed words, such as *cookbook* or *workbench*. Linguists might object that each one of them is a compound and thus, as far as phonotactics is concerned, might be two words rather than one. However, that does not solve the problem of Star Wars's Admiral Ackbar, pronounced *akbar*. The problem with *kbin* is not *kb*, it's *kb* at the start of the word. The forbidden sequence is not *kb*, but rather *$kb*. This is a **trigram**, not a bigram.

n-gram grammars

We can generalize the notion of bigram to sequences of arbitrary length. A trigram is a sequence of three elements, a 4-gram contains four elements, and quite generally, an **n-gram** consists of **n** elements.

Example 4 Let us first look at the bigrams, trigrams, and 4-grams of *kobinal*. The bigrams of *kobinal* are *ko*, *ob*, *bi*, *in*, *na*, and *al*. The trigrams are *kob*, *obi*, *bin*, and *nal*. The 4-grams are *kobi*, *obin*, *bina*, and *inal*.
For *banana*, the only bigrams are *ba*, *an*, and *na*. The trigrams are *ban*, *ana*, and *nan*. The 4-grams are *bana*, *anan*, and *nana*.

```
def ngrams(word, n):
    return sorted(list(set(''.join(ngram)
                           for ngram in zip(*[word[i:]
                                              for i in range(n)]))))

def ngram_print(word, n):
    print("The {}-grams of {} are:".format(n, word))
```

```
    print(ngrams(word, n))

for n in [2, 3, 4]:
    ngram_print("ababa", n)
    ngram_print("babab", n)
    print()
```

One problem with large *n*-grams is that some words may be shorter than *n* even after edge markers have been added. Just what are the 4-grams of *a*? To avoid this, we pad out the word with n-1 edge markers.

Example 5 Now consider the bigrams, trigrams, and 4-grams of *kobinal* with edge markers. For bigrams, we have to look at *$kobinal$*, which has the bigrams *$k*, *ko*, *ob*, *bi*, *in*, *na*, *al*, and *l$*. The trigrams are computed over *$$kobinal$$*, so they're *$$k*, *$ko*, *kob*, *obi*, *bin*, *nal*, *al$*, and *l$$*. The 4-grams are *$$$k*, *$$ko*, *$kob*, *kobi*, *obin*, *bina*, *inal*, *nal$*, *al$$*, and *l$$*.
For *banana*, the bigrams are now *$b*, *ba*, *an*, *na*, and *a$*. The trigrams are *$$b*, *$ba*, *ban*, *ana*, *nan*, *na$*, and *a$$*. The 4-grams are *$$$b*, *$$ba*, *$ban*, *bana*, *anan*, *nana*, *ana$*, *na$$*, and *a$$$*.

```
def ngrams(word, n):
    word = "$" * (n-1) + word + "$" * (n-1)
    return sorted(list(set(''.join(ngram)
                           for ngram in zip(*[word[i:]
                                              for i in range(n)]))))

def ngram_print(word, n):
    print("The {}-grams of {} are:".format(n, word))
    print(ngrams(word, n))

for n in [2, 3, 4]:
    ngram_print("ababa", n)
    ngram_print("babab", n)
    print()
```

Now we can finally state clearly why *kobinal*, *workbench* and *akbar* are all phonotactically well-formed, whereas *kbin* is not: the latter contains the illicit trigram *$kb*.

We also refine our original hypothesis about the phonotactics of natural languages: every phonotactic system is described by a set of forbidden *n*-grams.

Exercise 3 Consider once more the word *supercalifragilisticexpialidocious*. For each one of the following, say whether it is a bigram of the word (with edge markers), a trigram, a 4-gram, or none of those choices.

1. $fr
2. z

3. do$c
4. s$$$
5. sit
6. $sup

Exercise 4 Suppose a language has the vowels *a* and *u*, the voiced consonants *z* and *v*, and the voiceless consonants *s* and *f*. The language also has intervocalic voicing, which means that no voiceless consonants may occur between vowels. Write an *n*-gram grammar that expresses this fact.

A look at German

Unlike English, German is perfectly fine with words that starts with *kn*. But just like English, it does not like words that start with *rb*. We can capture this by writing a trigram grammar for German that contains the forbidden trigram *$rb*. But German also has a process known as word-final devoicing: voiced sounds become voiceless at the end of a word. So whereas an English speaker will happily pronounce *woods* with a *z* at the end, a German speaker turns the *z* into an *s* like in *trinkets*. Alright, no big deal, we just forbid *z$* too. Or do we?

We now have a forbidden trigram *$rb* and a forbidden bigram *z$*. Are we allowed to mix bigrams and trigrams this way? More generally, can every negative *n*-gram grammar also contain *k*-grams, where $k < n$? Could this create inconsistencies, or make negative *n*-gram grammars more powerful? We'll see in the next section.

Exercise 5 Consider the formal language where all strings are sequences of *a*, *b*, and *c* such that

1. every string starts with *a*
2. no string ends with *c*
3. *a* and *c* are always separated by at least two symbols.

Write a negative *n*-gram grammar for this language such that all *n*-grams have the same length.