An algorithm to assign features to a set of natural classes

Mayer, Connor Joseph
connor.joseph.mayer@gmail.com

Daland, Robert
r.daland@gmail.com

December 10, 2017

Abstract

This squib describes a dynamic programming algorithm which assigns features to a set of natural classes. The input consists of a set of classes, each containing one or more segments; in other words, a subset of the powerset of a segmental alphabet Σ. If a class can be generated as the union of existing features (= intersection of already-processed classes), those features are propagated to every segment in the class. Otherwise, a new feature/value is assigned. The algorithm comes in 4 flavors, which differ with respect to complementation and how negative values are assigned. We show that these variants yield *privative specification*, *contrastive underspecification*, *contrastive specification*, and *full specification*, respectively. The main text sets out necessary background, and illustrates each variant of the algorithm. The Appendix formally proves that each algorithm is sound.

1 Introduction

merge what Connor wrote

2 Definitions and notation

Let Σ denote an alphabet of segments. We will use the term *class* to mean a subset of Σ.

2.1 Natural classes and natural class systems

A *natural class system* $\mathcal{C}$ is a set of classes over Σ, $\mathcal{C} = \{C_i\}_{i=1}^{N}$, which includes Σ itself. To illustrate, a vowel harmony lattice is shown in Fig 1, with downward arrows representing the superset/subset relation.[1]

[1]Technically, this is a not a lattice because it does not contain the empty set. This technical detail is not important for our purposes.

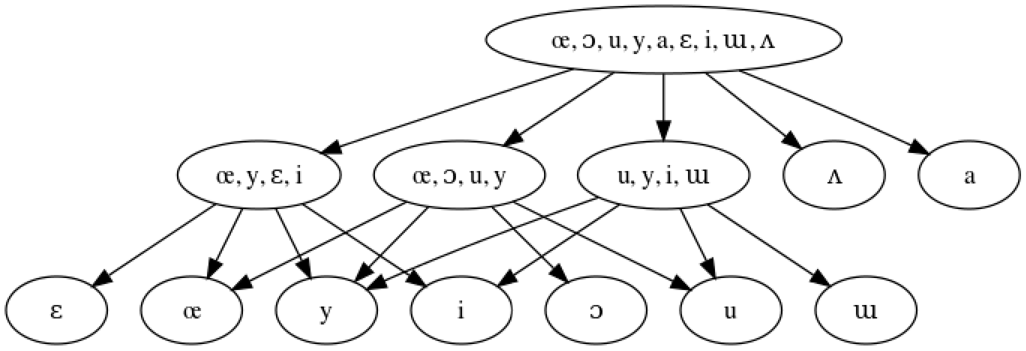

Figure 1: Vowel harmony lattice

2.2 Feature systems and featurizations

A *feature system* is a tuple $(\mathcal{F}, \Sigma, \mathcal{V})$ where

- Σ is a segmental alphabet,
- $\mathcal{V}$ is a set of values, and
- $\mathcal{F}$ is a *featurization*: a set of features $\{f_j\}_{j=1}^{M}$, where each feature is a function $f : \Sigma \to \mathcal{V}$ mapping segments to feature values

A feature system for the vowel harmony lattice shown in Fig. 1 is shown below:

σ	front	back	low	high	round
i	+	−	−	+	−
y	+	−	−	+	+
ɯ	−	+	−	+	−
u	−	+	−	+	+
ɛ	+	−	−	−	−
œ	+	−	−	−	+
ʌ	−	+	−	−	−
ɔ	−	+	−	−	+
a	−	+	+	−	−

Table 1: Example of a fully specified featurization.

In the next subsection we formalize featural descriptors, which relate classes and feature systems.

2.3 Featural descriptors

Let $(\mathcal{F}, \Sigma, \mathcal{V})$ be a feature system. We restrict $\mathcal{V}$ to the following possibilities:

- *privative specification*: $\mathcal{V} = \{+, 0\}$
- *full specification*: $\mathcal{V} = \{+, -\}$
- *contrastive specification*: $\mathcal{V} = \{+, -, 0\}$

A *featural descriptor* $\mathbf{e}$ is a set of feature/value pairs, where the values cannot be 0. For example, $\mathbf{e} = [+\text{front}, -\text{low}]$ is a featural descriptor.

Formally, a featural descriptor is a subset of $(\mathcal{V} \setminus \{0\}) \times \mathcal{F}$. This means that the space of all licit featural descriptors is the powerset of this set. We use the notation $\mathcal{V}^{\mathcal{F}}$ to denote this space: $\mathcal{V}^{\mathcal{F}} = \{\mathbf{e} \,|\, \mathbf{e} \subset (\mathcal{V} \setminus \{0\}) \times \mathcal{F}\}$. Moreover, every featural descriptor $\mathbf{e}$ can be expressed in the form $\mathbf{e} = [\alpha_k F_k]_{k \in K}$, where K is some index set K, each α_k is a value in $\mathcal{V} \setminus \{0\}$, and each F_k is a feature in $\mathcal{F}$. We use this form to relate featural descriptors and the classes they describe.

A featural descriptor describes the class of segments which have at least the feature/value pairs it contains. We use the notation $\langle \mathbf{e} \rangle$ to indicate the natural class that is described by $\mathbf{e}$:

- $\langle [\alpha_k F_k]_{k \in K} \rangle = \{x \in \Sigma \,|\, \forall k \in K \, [F_k(x) = \alpha_k]\}$

Finally, we define $\langle \mathcal{V}^{\mathcal{F}} \rangle = \{\langle \mathbf{e} \rangle \,|\, \mathbf{e} \in \mathcal{V}^{\mathcal{F}}\}$. This is the set of natural classes over Σ that can be described by the feature system $(\mathcal{F}, \Sigma, \mathcal{V})$.

Note that while every featural descriptor in $\mathcal{V}^{\mathcal{F}}$ picks out a class in $\langle \mathcal{V}^{\mathcal{F}} \rangle$, the two are not in 1-1 correspondence. This is because the same class can often be described by multiple featural descriptors. For example, under the the feature system shown in Table 1, the featural descriptor [+front] picks out the same class as the featural descriptor [+front, −low] (the front vowels). Moreover, the featural descriptors [+front, −front] and [+high, +low] both pick out the empty set.

It is straightforward to show that every feature system generates a natural class system. Our goal in the remainder of this paper is to go the opposite direction: starting with a natural class system $\mathcal{C}$ over an alphabet Σ, can we assign a 'good' feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ that describes $\mathcal{C}$? Prior to doing this, it is necessary to define what is meant by a 'good' feature system.

2.4 Criteria for feature systems

Let $(\mathcal{F}, \Sigma, \mathcal{V})$ be a feature system with features $\{f_j\}_{j=1}^{M}$.

- The *feature vector* of a segment x is the tuple $F(x) = (f_j(x))_{j=1}^{M}$.

- Two segments x, y are *featurally distinct* if and only if $F(x) \neq F(y)$; in other words, if they do not match on at least feature.
- The feature system is *well-formed* if every pair of segments in Σ is featurally distinct.
- A feature f_j is *redundant* if $\mathcal{F}' = \mathcal{F} \setminus \{f_j\}$ is well-formed.
- A featurization is *efficient* if it contains no redundant features.

Note that these requirements do not entail that every segment can be uniquely picked out by a featural descriptor. For example, consider the feature system in Table 2. The 'sonorant' [R] can be uniquely picked out by the featural descriptor $[+son, +vcd]$, and the 'voiced' segments [D, R] can be picked out by the featural descriptor $[+vcd]$, but there is no way to uniquely pick out the 'voiceless' obstruent [T], or for that matter the voiced obstruent [D] without [R].

σ	son	vcd
R	+	+
D	0	+
T	0	0

Table 2: Example of a privatively specified featurization.

However, the feature system is still well-formed, because $F(\text{D}) = [0, +] \neq [0, 0] = F(\text{T})$ (and similarly for R). We call a feature system *rich* if every segment can be uniquely picked out by some featural descriptor; we do not impose the requirement that a feature system be rich.

On the other hand, we do wish for the feature system to be expressive enough to assign a licit featural descriptor to every natural class in the input. Thus, for a set of natural classes $\mathcal{C}$ in the 'input', we say that a feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ is *adequate* if $\mathcal{C} \subset \langle \mathcal{V}^{\mathcal{F}} \rangle$.

One might further wish that a solution contain no more features than needed. It turns out that the number of features needed to describe a natural class system depends on the value set. For example, the privative feature system shown in Table 2 cannot uniquely pick out either of the 'obstruents' [D] or [T] (or even both of them together, to the exclusion of the sonorant [R]). However, if the privative specification is made into a full specification by replacing the 0 values with $-$ values, then the 'same' features pick out the following classes:

- $\langle [] \rangle = \{\text{R, D, T}\}$
- $\langle [+son] \rangle = \{\text{R}\}$
- $\langle [-son] \rangle = \{\text{D, T}\}$
- $\langle [+vcd] \rangle = \{\text{R, D}\}$

- $\langle[-vcd]\rangle = \{\text{T}\}$
- $\langle[-son, +vcd]\rangle = \{\text{D}\}$
- $\langle[+son, -son]\rangle = \varnothing$

The privative specification earlier can only pick out {R, D, T} and {R}. Thus, whether a feature system contains the minimum number of features depends on the value set. We conjecture that the algorithms we describe later do in fact assign minimal features given their value sets, but we do not prove it here. For this reason, we only require an efficient (non-redundant) solution, not a minimal one.

3 Intersectional closure

In this section we define the *intersectional closure* of a natural class system $\mathcal{C}$. We prove that if a feature system is expressive enough to generate all the classes in $\mathcal{C}$, it generates the intersectional closure. Then we give a dynamic programming algorithm which efficiently computes the intersectional closure of a natural class system, as well as the intersection relation. It turns out that these structures are exactly what are needed to assign an efficient feature system.

The *intersectional closure* of $\mathcal{C}$, denoted $\mathcal{C}_\cap$, is the natural class system consisting of every class that can be generated by the intersection of finitely many classes in $\mathcal{C}$. In other words, $\mathcal{C}_\cap$ consists of every class in $\mathcal{C}$, as well as any class that can be generated by the intersection of two or more classes in $\mathcal{C}$.

Lemma: The natural class described by the union of two featural descriptors is the intersection of the natural classes described by each featural descriptor.

Proof: Let $\mathcal{C} = \{C_i\}_{i=1}^n$ be a natural class system and $(\mathcal{F}, \Sigma, \mathcal{V})$ a feature set. Further, let $\mathbf{e}_i, \mathbf{e}_j \in \mathcal{V}^\mathcal{F}$, and $C_i = \langle \mathbf{e}_i \rangle$, $C_j = \langle \mathbf{e}_j \rangle$. We need to show that $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle = C_i \cap C_j$. First, suppose $x \in C_i \cap C_j$. Then $x \in C_i$. By definition, x must have the features in $\mathbf{e}_i$. Similarly, $x \in C_j$, and therefore must have the features in $\mathbf{e}_j$. Thus, x has the features in $\mathbf{e}_i \cup \mathbf{e}_j$. This shows that $C_i \cap C_j \subset \langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$. Now, suppose $x \in \langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$. Then x has all the features of $\mathbf{e}_i$, and so $x \in C_i$. Similarly, x has all the features of $\mathbf{e}_j$, so $x \in C_j$. This shows that $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle \subset C_i \cap C_j$. Since both $C_i \cap C_j$ and $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$ are subsets of each other, they are equal.

Theorem: Let $\mathcal{C} = \{C_i\}_{i=1}^n$ be a natural class system and $(\mathcal{F}, \Sigma, \mathcal{V})$ a feature set. If $\mathcal{C} \subset \langle \mathcal{V}^\mathcal{F} \rangle$, then $\mathcal{C}_\cap \subset \langle \mathcal{V}^\mathcal{F} \rangle$. In other words, if $(\mathcal{F}, \Sigma, \mathcal{V})$ is rich enough to generate $\mathcal{C}$, it generates the intersectional closure.

Proof: Let C_i, C_j be classes in $\mathcal{C}$, so that $C_i, C_j \in \langle \mathcal{V}^\mathcal{F} \rangle$. This means that there exist featural descriptors $\mathbf{e}_i, \mathbf{e}_j \in \mathcal{V}^\mathcal{F}$ such that $\langle \mathbf{e}_i \rangle = C_i$ and $\langle \mathbf{e}_j \rangle = C_j$. By the Lemma above, $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle = C_i \cap C_j$. This illustrates that the union operation on a pair of featural descriptors corresponds to the intersection operation on the corresponding pair natural classes. Since union and intersection operations are associative, the extension to any number of finite unions/intersections proceeds by induction (e.g. if $C_i, C_j, C_k \in \mathcal{C}$, $C_i \cap C_j \cap C_k = (C_i \cap C_j) \cap C_k$; $(C_i \cap C_j) \in C_\cap$ and $C_k \in C_\cap$, so $C_i \cap C_j \cap C_k \in \mathcal{C}_\cap$).

Next, we give an algorithm which computes the intersectional closure, a modified variant of Dijkstra's shortest-paths algorithm. As we will show later, the computational benefit of precomputing the intersectional closure is that it efficiently computes the intersection relation, which reduces the computational complexity of the featurization algorithm. We assume that the input is a natural class system $\mathcal{C} = \{C_i\}_{i=1}^N$, whose classes are sorted in decreasing order of cardinality. (This implies that $C_i \nsubseteq C_j$ whenever $j > i$, so there is no need to check the subset relation.)

```
**Require:** $\mathcal{C}$ sorted by decreasing size ($|C_i| \geq |C_{i+1}|$ for every $i = 1 \ldots N-1$)
**Require:** subset matrix $S$ ($N \times N$): $S_{ij} = 1$ if $C_j \subset C_i$, 0 otherwise

  closure ← $\mathcal{C}$
  pairQueue ← $\{(i,j) | i < j\}$
  intersections ← $\varnothing$

  **while** pairQueue $\neq \varnothing$ **do**
    $(i,j)$ ← POP(pairQueue)
    **if** $S_{ij} = 0$ **then**
      c ← $C_i \cap C_j$
      **if** c $\in$ closure **then** {found an existing intersection}
        $k$ ← index such that closure$_k = c$
        PUSH $(i,j,k)$ → intersections
      **else** {found a new class!}
        $N$ ← |closure|
        APPEND $c$ to closure
        RESIZE $S \rightarrow (N+1) \times (N+1)$
        **for** $k = 1$ to $N$ **do**
          **if** c $\subset C_k$ **then** {update subset matrix}
            $S_{k,N+1} \rightarrow 1$
          **end if**
          **if** $C_k \subset$ c **then**
            $S_{N+1,k} \rightarrow 1$
          **end if**
          PUSH $(k, N+1)$ → pairQueue
```

```
            PUSH $(N+1, k) \to$ pairQueue
          end for
        end if
      end if
    end while
```

4 Privative specification

achieved by assigning a new feature [+f] only, to every segment in X

5 Contrastive underspecification

achieved by assigning a new feature [+f] to every segment in X, and if $Y \setminus X$ (the complement of X with respect to Y) is in the input, then [-f] is assigned to every segment in $Y \setminus X$

6 Contrastive specification

achieved by assigning a new feature [+f] to every segment in X, and [-f] to every segment in $Y \setminus X$ (even if $Y \setminus X$ was not in the input)

7 Full specification

achieved by assigning a new feature [+f] to every segment in X, and [-f] to every segment in $\Sigma \setminus X$

A Formal proof of the algorithm

A.1 Privative underspecification

A.2 Contrastive underspecification

A.3 Contrastive specification

A.4 Full specification