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 18, 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 pair 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. We will also use the notation $\mathcal{P}(X)$ to indicate the *powerset* of X – the set of all subsets of X (including the empty set $\varnothing$ and X itself).

2.1 Natural classes and natural class systems

A *natural class system* $\mathcal{C}$ over an alphabet Σ is any subset of $\mathcal{P}(\Sigma)$ which includes Σ itself. Formally, $\mathcal{C} = \{C_i\}_{i=1}^{N}$, where each $C_i \subset \Sigma$. Fig 1 illustrates this definition with a natural class system over a set of vowels.

In this graph, each node corresponds to a class, and a (downward) arrow from class C_i to class C_j indicates that $C_j \subset C_i$. To be more precise, downward arrows indicate that

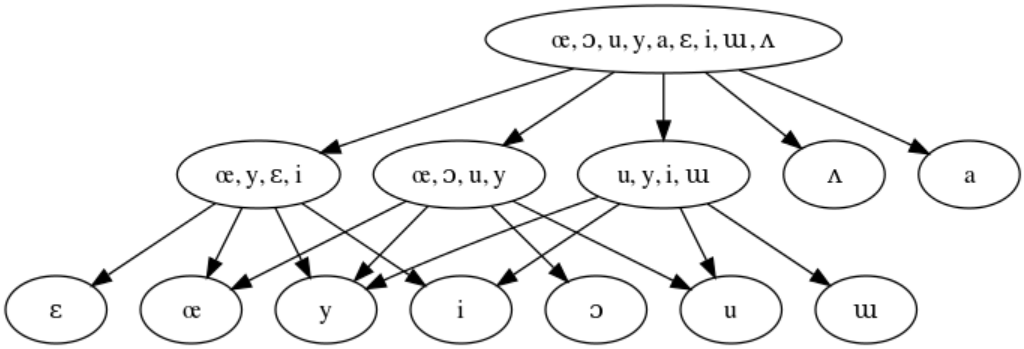

Figure 1: Vowel harmony lattice

there is no 'intervening' class C_k. For example, the front vowels {ɔɛ, y, ɛ, i} intervene between all vowels and {ɛ}, so we do not include an arrow from the top node to the {ɛ}, node. This relation will prove important for the algorithms we discuss later, and so we formalize it here: C_i is a *parent* of C_j (and C_j is a *daughter* of C_i) if and only if

- $C_j \subset C_i$, and
- $\nexists C_k \in \mathcal{C}\,[C_j \subset C_k \subset C_i]$

Any directed graph like Fig 1 has an equivalent representation with a square matrix. Row i is identified with class C_i, column j with C_j; and the value in the (i, j) cell indicates the relationship between C_i and C_j. For binary relations like the subsethood and daughterhood, it is convenient to use the Boolean field: the set $\{T, F\}$ with operations $\cdot$ (logical AND), $+$ (logical OR), and additive inverse $-$ (logical NOT). The *subset matrix* is defined thus:

$$S_{ij} = \begin{cases} T & \text{if } C_j \subset C_i \\ F & \text{otherwise} \end{cases}$$

The advantage of writing a relation in this way is that fundamental matrix operations correspond to various measures of interest. For example, when S is defined as above, and S^2 is calculated from ordinary matrix multiplication, then S^2 indicates the subset-of-a-subset relation, which corresponds graphically to 'paths' of length 2 on a subset graph. In other words, $(S^2)_{ij} = T$ means there is a C_k such that $S_{ik} = T$ AND $S_{kj} = T$. If we further let AND denote the element-wise multiplication operator, then the *daughter matrix* can be computed from the subset matrix as follows:

$$D = S \text{ AND } -S^2$$

This matrix expressed the daughterhood relation: $D_{ij} = T$ means C_j is a daughter of C_i. (Similarly, the subset matrix can be expressed as the sum of all powers of the daughter matrix, $S = \sum_{n=1}^{N} D^n$. For this problem, we generally expect to compute D from S, rather than *vice versa*.)

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

To illustrate, 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 feature system.

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, $\mathbf{e} \subset (\mathcal{V} \setminus \{0\}) \times \mathcal{F}$. For example, $\mathbf{e} = [+\text{front}, -\text{low}]$ is a featural descriptor.

To relate featural descriptors and natural classes, note that every featural descriptor $\mathbf{e}$ can be expressed in the form $\mathbf{e} = \{\alpha_k F_k\}_{k=1}^{K}$ (where each α_k is a value in $\mathcal{V} \setminus \{0\}$, and each F_k is some feature function $f_j \in \mathcal{F}$). Informally, we say that a featural descriptor describes the class of segments which have (at least) the feature/value pairs it contains. Formally, we write $\langle \mathbf{e} \rangle$ to indicate the natural class that corresponds to the featural descriptor $\mathbf{e}$:

$$\langle \{\alpha_k F_k\}_{k=1}^{K} \rangle = \{x \in \Sigma \mid F_k(x) = \alpha_k \text{ for every } k\}$$

We use the notation $\mathcal{V}^{\mathcal{F}}$ to denote the powerset of $(\mathcal{V} \setminus \{0\}) \times \mathcal{F}$, i.e. the set of all licit featural descriptors. Lastly, we define $\langle \mathcal{V}^{\mathcal{F}} \rangle = \{\langle \mathbf{e} \rangle \mid \mathbf{e} \in \mathcal{V}^{\mathcal{F}}\}$, the set of all natural classes described by some featural descriptor in $\mathcal{V}^{\mathcal{F}}$. We say that the feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ generates the natural class system $\langle \mathcal{V}^{\mathcal{F}} \rangle$.

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 of Table 1, the featural descriptor [+front] picks out the same class as the featural descriptor [+front, −low] (namely, the front vowels). Moreover, the featural descriptors [+front, −front] and [+high, +low] both pick out the empty set.

We say that a feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ *covers* a natural class system $\mathcal{C}$ if $\mathcal{C} \subset \langle \mathcal{V}^{\mathcal{F}} \rangle$; in other words if the feature system provides a distinct featural representation for every class in $\mathcal{C}$. In the remainder of this squib, we show how to construct a feature system that covers an arbitrary natural class system $\mathcal{C}$.

We begin with a worked-out example illustrating the difference between privative and full specification with the same segmental alphabet. Then we introduce the notion of intersectional closure, which leads naturally to a featurization algorithm for privative specification. Simple modifications yield algorithms for contrastive and full specification.

2.4 Example

Let $\Sigma = \{\text{R, D, T}\}$. Informally, the reader may think of [R] as a sonorant, [D] as a voiced obstruent, and [T] as a voiceless obstruent; accordingly we use the feature names *son* and *vcd*. In this section, we illustrate the consequences of privative versus full specification, using featurizations that are isomorphic (that is, they match on the + values, and differ only as to whether the non-+ values are 0 or −). We begin with Table 2.
The set of natural classes it describes, and the simplest featural descriptor for each, are shown below:

- [ ] – {R, D, T}

σ	son	vcd
R	+	+
D	0	+
T	0	0

Table 2: Sonorants and obstruents with privative specification.

- [+son] – {R}
- [+vcd] – {R, D}

Note that this featurization provides no featural descriptor that uniquely picks out the voiceless obstruent [T], no way to pick out the obstruents [T] and [D] to the exclusion of [R], and no way to pick out the voiced obstruent [D] without [R].

Next, consider the isomorphic featurization in which the 0's from Table 2 are replaced with −'s:

σ	son	vcd
R	+	+
D	−	+
T	−	−

Table 3: Sonorants and obstruents with full specification.

The set of natural classes this featurization describes is much larger, because the number of (extensionally distinct) featural descriptors is larger:

- [ ] = {R, D, T}
- [+son] = {R}
- [−son] = {D, T}
- [+vcd] = {R, D}
- [−vcd] = {T}
- [−son, +vcd] = {D}
- [+son, −son] = ∅

An important generalization emerges from comparing these featurizations: the more 0's in the featurization, the greater the number of distinct feature functions that will be required to cover the same natural class system. In one sense, privative specification is more complex, because it will normally involve more features. However, in another sense, it is simpler,

because there are only + values to handle. Therefore, we will treat privative specification first. Prior to this, we introduce the notion of intersectional closure – the data structure that proves useful for efficiently assigning a privative feature system.

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 is needed to efficiently assign a 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 $\mathcal{C}$. Formally, $\mathcal{C}_\cap = \{\bigcap C' \in P \mid \exists P \in \mathcal{P}(\mathcal{C})\}$ (where $\mathcal{P}(\mathcal{C})$ is the powerset of $\mathcal{C}$). We show that if a feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ is rich enough to cover $\mathcal{C}$, it generates $\mathcal{C}_\cap$.

Lemma: If $\mathbf{e}_i, \mathbf{e}_j \in \mathcal{V}^\mathcal{F}$, then $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle = \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$.

Proof: The proof proceeds by showing that $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle \subset \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$ and $\langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle \subset \langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$. Let $C_i = \langle \mathbf{e}_i \rangle$ and $C_i = \langle \mathbf{e}_i \rangle$. 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 $x \in C_j$. Therefore $x \in C_i \cap C_j$. 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 $\mathcal{C}_\cap \subset \langle \mathcal{V}^\mathcal{F} \rangle$.

Proof: Let C be an arbitrary class in $\mathcal{C}_\cap$. By definition of $\mathcal{C}_\cap$, there exist $\{C\}_{i=1}^n$ such that $C_i \in \mathcal{C}\, \forall i$ and $C = \bigcap_i C_i$. The hypothesis that $\mathcal{C} \subset \langle \mathcal{V}^\mathcal{F} \rangle$ implies that for every C_i in $\mathcal{C}$, there exists a featural descriptor $\mathbf{e}_i$ such that $\langle \mathbf{e}_i \rangle = C_i$. Thus, $C = \bigcap_i C_i = C_1 \cap C_2 \cap \ldots \cap C_n$ can also be written $C = \bigcap_i \langle \mathbf{e}_i \rangle = \langle \mathbf{e}_1 \rangle \cap \langle \mathbf{e}_2 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle$. It follows by induction that $C = \langle \bigcup_i \mathbf{e}_i \rangle$:

$$
\begin{aligned}
C &= \langle \mathbf{e}_1 \rangle \cap \langle \mathbf{e}_2 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle \\
&= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \rangle \cap \mathbf{e}_3 \cap \ldots \cap \langle \mathbf{e}_n \rangle \\
&= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \cup \mathbf{e}_3 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle \\
&\ldots \\
&= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \cup \ldots \cup \mathbf{e}_n \rangle \\
&= \langle \textstyle\bigcup_i \mathbf{e}_i \rangle
\end{aligned}
$$

The preceding chain of logic demonstrates that if a class can be expressed as the intersection of natural classes in $\mathcal{C}$, then its features are the union of the features in each of those classes. Thus, if $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$, it covers the intersectional closure. This completes the proof.

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 $\mathcal{C} = \{C_i\}_{i=1}^{N}$.

Ensure: $\mathcal{C}_\cap$ is the intersectional closure of $\mathcal{C}$
Ensure: $\mathcal{I}(i,j) = k$ whenever $C_i \cap C_j = C_k$ (for all i, j, k)

$\mathcal{C}_\cap \leftarrow \mathcal{C}$ {ordered list of classes}
$N \leftarrow |\mathcal{C}_\cap|$ {size of $\mathcal{C}_\cap$}
$\mathcal{Q} \leftarrow \{(i,j) \mid 1 \le i, j \le N, i \ne j\}$ {queue of natural class index pairs}
$\mathcal{I} \leftarrow \varnothing$ {hash table: $(i,j) \to k$ means $C_k = C_i \cap C_j$}

while $\mathcal{Q} \ne \varnothing$ **do**
  $(i,j) \leftarrow \textsc{pop}(\mathcal{Q})$
  if not $(C_j \subset C_i$ **or** $C_i \subset C_j)$ **then**
    $C \leftarrow C_i \cap C_j$
    if $C \in \mathcal{C}_\cap$ **then**
      $\mathcal{I}(i,j) \leftarrow \textsc{index}(\mathcal{C}_\cap, C)$
    else
      $\textsc{append}(\mathcal{C}_\cap, C)$
      $N \leftarrow N + 1$
      for $k = 1$ to N **do**
        PUSH $(k, N) \to \mathcal{Q}$
        PUSH $(N, k) \to \mathcal{Q}$
      end for
    end if
  end if
end while

4 Privative specification

Require: $\mathcal{C}$ is sorted in decreasing order of size $(\forall i, j\,[i < j \rightarrow |C_i| \geq |C_j|)$
Require: $\mathcal{V} = \{+, 0\}$
Ensure: output featurization $\mathcal{F} = \{f_j\}_{j=1}^{M}$ such that $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$

$\mathcal{Q}, \mathcal{I} \leftarrow$ IntersectionalClosure($\mathcal{C}$)
while $\mathcal{Q} \neq \varnothing$ **do**

end while

The algorithm is considered sound if it returns a covering feature set for every valid input.

Proof:

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