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 12, 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 $C_i \subset \Sigma$. To illustrate, a natural class system for a set of vowels is shown in Fig 1; downward arrows indicate superset-/subset-hood.
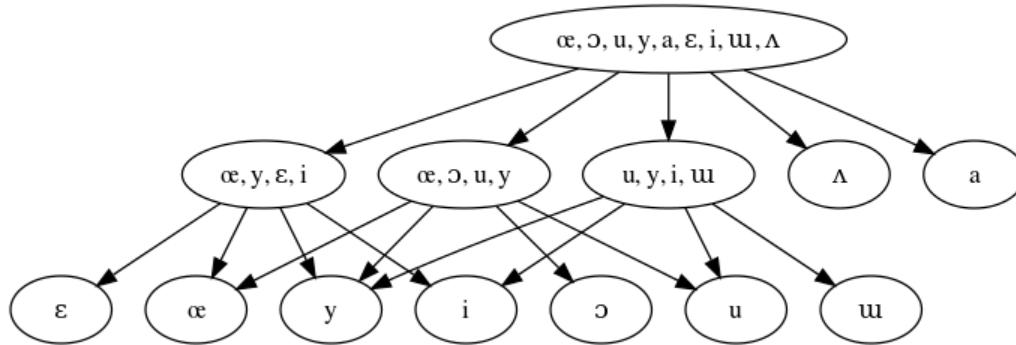
Figure 1: Vowel harmony lattice

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

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

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 $[+\text{front}]$ picks out the same class as the featural descriptor $[+\text{front}, -\text{low}]$ (namely, the front vowels). Moreover, the featural descriptors $[+\text{front}, -\text{front}]$ and $[+\text{high}, +\text{low}]$ 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.

3

2.4 Example

Let $\Sigma = \{$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.

σ	son	vcd
R	+	+
D	0	+
T	0	0

Table 2: Sonorants and obstruents with privative specification.

The set of natural classes it describes, and the simplest featural descriptor for each, are shown below:

- [ ] − {R, D, T}

- [+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) licit featural descriptors is larger:

- [ ] = {R, D, T}

- [+son] = {R}

- [−son] = {D, T}

- $[+\text{vcd}] = \{\text{R, D}\}$

- $[-\text{vcd}] = \{\text{T}\}$

- $[-\text{son}, +\text{vcd}] = \{\text{D}\}$

- $[+\text{son}, -\text{son}] = \varnothing$

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 $C = \langle \bigcup_i \mathbf{e}_i \rangle$:

$$C = \langle \mathbf{e}_1 \rangle \cap \langle \mathbf{e}_2 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle$$
$$= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \rangle \cap \mathbf{e}_3 \cap \ldots \cap \langle \mathbf{e}_n \rangle$$
$$= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \cup \mathbf{e}_3 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle$$
$$\ldots$$
$$= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \cup \ldots \cup \mathbf{e}_n \rangle$$
$$= \langle \bigcup_i \mathbf{e}_i \rangle$$

The preceding chain of logic demonstrates that is a class can be expressed as the intersection of natural classes in $\mathcal{C}$, its features must be 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 $\mathcal{C} = \{C_i\}_{i=1}^N$, whose classes are sorted in decreasing order of cardinality. (This implies that $C_i \not\subseteq C_j$ 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 $\leftarrow \mathcal{C}$
    pairQueue $\leftarrow \{(i,j)|i < j\}$
    intersections $\leftarrow \varnothing$

    while pairQueue $\neq \varnothing$ **do**
      $(i,j) \leftarrow$ POP(pairQueue)
      if $S_{ij} = 0$ **then**
        c $\leftarrow C_i \cap C_j$
        if c $\in$ closure **then** {found an existing intersection}
          $k \leftarrow$ index such that closure$_k$ = c
          PUSH $(i, j, k) \rightarrow$ intersections
        else {found a new class!}
          $N \leftarrow |$closure$|$

```
        APPEND c to closure
        RESIZE S → (N + 1) × (N + 1)
        for k = 1 to N do
            if c ⊂ C_k then {update subset matrix}
                S_{k,N+1} → 1
            end if
            if C_k ⊂ c then
                S_{N+1,k} → 1
            end if
            PUSH (k, N + 1) → pairQueue
            PUSH (N + 1, k) → 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