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

November 28, 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 Features, lattices, definition, and notation

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

Definition: A *natural class system* $\mathcal{C}$ is a set of classes over Σ, $\mathcal{C} = \{C_i\}_{i=1}^{n}$, which includes Σ itself, and the empty set (i.e. $\varnothing, \Sigma \in \mathcal{C}$).

Every natural class system is a subset of the powerset of Σ – the set of all subsets of Σ. This kind of structure is the canonical example of a *lattice* under the subset relation. That is, the subset relation $\subseteq$ forms a *partial order* (meaning that not all pairs of sets in $\mathcal{C}$ are comparable), and for every pair $C_i, C_j \in \mathcal{C}$, there is a greatest common subset and a least common superset. An example is shown below – a vowel harmony lattice (the empty set is not shown):

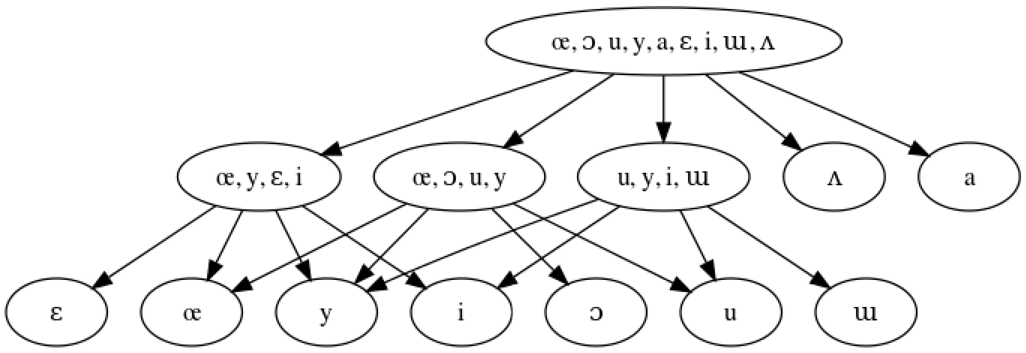

Figure 1: Vowel harmony lattice

Definition: A *featurization* of an alphabet Σ, $\mathcal{F}(\Sigma)$ assigns a set of feature/value pairs to every segment Σ. Formally, a featurization consists of a set of features $\mathcal{F}(\Sigma) = \{f_j\}_{j=1}^{k}$, where each feature is a function $f_j : \Sigma \to \{+, -, 0\}$.

As an example, a featurization of the vowel harmony lattice above 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.

We will call a featurization *well-formed* if every pair of segments in Σ is featurally distinct. It is known that every well-formed featurization gives rise to a natural class system. [1]

Definition: Let $\mathcal{C} = \{C_i\}_{i=1}^{n}$ be a natural class system. The *intersective closure* of $\mathcal{C}$,

[1] It is always possible to make ill-formed featurizations become well-formed. For example, suppose that [ptk] are not given distinct place features. One way to make the featurization well-formed is to add their place features. Another way to make it well-formed is to replace instances of [ptk] with a meta-symbol [T] in Σ, yielding a new segmental alphabet $\Sigma' = \Sigma \setminus \{p, t, k\} \cup \{T\}$.

denoted $\mathcal{C}^{\cap}$, is the natural class system consisting of every class in $\mathcal{C}$, as well as any class that can be generated by the intersection of two or more classes in $\mathcal{C}$. Formally,

$$\mathcal{C}^{\cap} = \{C \mid \exists(i_1, i_2, \ldots, i_n)\,[(\forall i_k\, C_{i_k} \in \mathcal{C}) \wedge (C = \cap_{k=1}^{n} C_{i_k})]\}$$

We will provide a dynamic programming algorithm which efficiently calculates the intersective closure of a natural class system, while simultaneously filling out a data structure which tracks the intersections. This algorithm lays the ground for the *privative specification* variant of the featurization algorithm. The other variants of the featurization algorithm differ only in whether complements of classes are considered (and if so, complements with respect to the smallest containing class, or the entire alphabet).

3 A dynamic programming algorithm for computing intersectional closure

describe the algorithm

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