An algorithm to assign features to a set of phonological classes

Mayer, Connor
connormayer@ucla.edu

Daland, Robert
r.daland@gmail.com

Abstract

This squib describes a dynamic programming algorithm which assigns features to a set of phonological 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

Distinctive features are the building blocks of phonological theory. Features represent phonetic qualities of speech sounds, and can be used in isolation or combination to describe individual sounds or classes of sounds (e.g., ?, ?, ?). This captures the generalization that sounds with similar phonetic properties tend to pattern similarly in the phonologies of languages. For example, the feature $\left[\text{ -voice }\right]$ picks out all sounds without voicing, while the combination of features $\left[\begin{smallmatrix}\text{-voice}\\ \text{-continuant}\end{smallmatrix}\right]$ picks out, for example, the English voiceless stop series.

Distinctive features have traditionally been described as *innate*: that is, all the sounds in the world's languages can be described by the same finite set of features, and the task of the language learner is to decompose the sounds of the ambient language into their consituent features to build a phonological grammar (e.g., ?, ?). This implies that the phonological classes we see in natural languages should be definable by combinations of these innate features.

Although feature theory has been extremely productive in phonology, there is evidence that shows many phonological classes cannot be described as a combination of phonetically-based features. For example, ? (?) conducted a survey of almost 600 languages and showed that, at best, any modern feature system can categorize 71% of attested sound classes. The remaining 29% are *phonetically disparate classes*, which require less theoretically-appealing

descriptions combinations of features to be described, such as XOR feature classes (e.g., [-voice] or [-continuant] but not both).

The preponderance of these phonetically disparate classes has led some researchers to propose that distinctive features are *learned* and *language-specific* (e.g., ?, ?, ?, ?, ?): learners are able to group sounds in their languages into classes regardless of whether they have any phonetic commonality. The striking regularities that exist across languages are explained as by-products of general human cognitive capabilities, such as categorization, sensitivity to frequency, and the ability to generalize, as well as the properties of the human vocal tract and auditory system.

This sets the stage for the goals of the current paper, which are somewhat modest. The basic question we address is the inverse of how features have typically been approached: rather than asking what classes a feature system defines, we instead focus on how a feature system can be learned from a set of predetermined classes. We begin by defining a formal notation for feature systems. We then describe the *intersectional closure* of a set of classes, which must be generated by any featurization of that set. Using the intersectional closure as a tool for efficient calculation, we then describe a suite of algorithms for learning various types of featurizations for a set of input classes and prove their correctness, examining as we do the trade offs between number of classes and number of features that each featurization method makes.

This paper makes several important contributions: first, it demonstrates a method for working backwards to feature systems underpinning learned classes of sounds and provides the code for use in future research. Second, it provides a detailed formalization of what a featurization of classes entails, allowing careful reasoning about the expressiveness of such featurizations. Finally, by comparing multiple types of featurization of a set of classes, it makes explicit predictions about what classes should be describable under each type, which may be useful for future experimental phonological research.

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 Classes and class systems

A *(natural) class system* $(\mathcal{C}, \Sigma)$ consists of an alphabet Σ and a set of classes $\mathcal{C}$ over that alphabet. Fig **??** 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 X to class Y indicates that $Y \subset X$. However, the figure does not include an arrow for every pair of classes that have a subset relationship. For example, the class {ɛ} is a subset of all vowels, but there is not an arrow from the class of all vowels to {ɛ}. This is because the class of front vowels {œ, y, ɛ, i} 'intervenes' between {ɛ} and the class of all vowels.

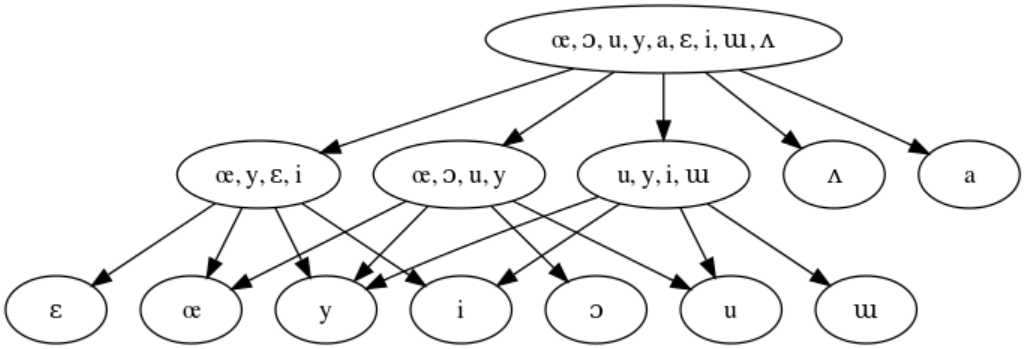

Figure 1: Vowel harmony lattice

We will use the terms *parent/daughter* to refer to cases in which a subset/superset relation holds and there is no intervening class.

Graphically, it is sufficient to indicate parent-daughter relations in figures like **??**, since other superset-subset relations (like $\{\varepsilon\} \subset$ all vowels) are entailed. However, we did not only introduce this concept for aesthetically pleasing graphs. As we show later, the parenthood relation is essential for the featurization algorithm. Thus, we formalize the definition here:

- X is a *parent* of Y (with respect to $\mathcal{C}$) if and only if $Y \subset X$, and $\nexists W \in \mathcal{C}\,[Y \subset W \subset X]$

We further define $\textsc{parents}(Y, \mathcal{C})$ as the set of all parents of Y (with respect to $\mathcal{C}$).

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 of Fig. **??** is shown below:
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*: $\mathcal{V} = \{+, 0\}$

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

Table 1: Example of a feature system.

- *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, $\mathbf{e} = \left[\begin{array}{c} \text{+front} \\ \text{-low} \end{array}\right]$ is a featural descriptor.

To relate featural descriptors and phonological 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 **??**, the featural descriptor $\left[\begin{array}{c} \text{+front} \end{array}\right]$ picks out the same class as the featural descriptor $\left[\begin{array}{c} \text{+front} \\ \text{-low} \end{array}\right]$ (namely, the front vowels). Moreover, the featural descriptors $\left[\begin{array}{c} \text{+front} \\ \text{-front} \end{array}\right]$ and $\left[\begin{array}{c} \text{+high} \\ \text{+low} \end{array}\right]$ 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 **??**.

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 **??** are replaced with −'s:
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}

σ	son	vcd
R	+	+
D	–	+
T	–	–

Table 3: Sonorants and obstruents with full specification.

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

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 and because it will result in fewer natural classes. 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 $\mathcal{C}$ as the set of classes that can be generated by intersecting Σ with any set of classes in $\mathcal{C}$. We relate the intersectional closure to features by showing 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.

3.1 Definition

A collection of sets $\mathcal{C}$ is *intersectionally closed* if and only if $\forall(X, Y \in \mathcal{C})\,[X \cap Y \in \mathcal{C}]$.

We write $\mathcal{C}_\cap$ to indicate the *intersectional closure* of a natural class system $(\mathcal{C}, \Sigma)$. This the smallest intersectionally closed collection which contains Σ and every set in $\mathcal{C}$; in other words, the intersectional closure does not contain any classes except Σ and those which can be generated by finite intersections of classes from $\mathcal{C}$.

3.2 Feature systems generate an intersectional closure

Now we explain why any feature system that covers $\mathcal{C}$ must cover $\mathcal{C}_\cap$. The explanation rides on the dual relationship between featural descriptors and the classes they describe: the class described by the union of two featural descriptors is the intersection of the classes described by each of the descriptors alone. This principle can be illustrated with the following example, using the vowel system in Fig. **??**. Let $\mathbf{e}_1 = \left[\text{ +front } \right]$ and $\mathbf{e}_1 = \left[\text{ +round } \right]$. Then

- $\langle \left[\text{ +front } \right] \rangle$ = {œ, y, ɛ, i}
- $\langle \left[\text{ +round } \right] \rangle$ = {œ, o, y, u}
- $\langle \left[\text{ +front } \right] \rangle \cap \langle \left[\text{ +round } \right] \rangle$ = {œ, y}
- $\langle \left[\text{ +front,+round } \right] \rangle$ = {œ, y

In other words, the set of vowels that are both front and round is the intersection of the set of vowels that are front and the set of vowels that are round. Lemma 1 proves that this kind of relationship holds for any pair of featural descriptors (and the classes they describe).

Lemma 1: Let $(\mathcal{F}, \Sigma, \mathcal{V})$ be a feature system. If $\mathbf{e}_i, \mathbf{e}_j \in \mathcal{V}^\mathcal{F}$, then $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle = \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$.

A formal proof is included in the Appendix. The key utility of this Lemma is that it can be applied inductively, to relate the union of multiple featural descriptors with the intersection of multiple classes.

Theorem 1: Let $(\mathcal{C}, \Sigma)$ 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 Y be an arbitrary class in $\mathcal{C}_\cap$. By definition of $\mathcal{C}_\cap$, there exist $\{X_i \in \mathcal{C}\}_{i \in I}$ (for some index set I, which we will omit hereafter) such that and $Y = \bigcap_i X_i$. The hypothesis that $\mathcal{C} \subset \langle \mathcal{V}^\mathcal{F} \rangle$ implies that for every such X_i, there exists a featural descriptor $\mathbf{e}_i$ such that $\langle \mathbf{e}_i \rangle = X_i$. Thus, $Y = \bigcap_i X_i = X_1 \cap X_2 \cap \ldots \cap X_n$ 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 $Y = \langle \bigcup_i \mathbf{e}_i \rangle$:

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

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.

The intersectional closure is defined as all possible intersections of classes in $\mathcal{C}$. Thus, if $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$, it covers the intersectional closure. This completes the proof.

3.3 Algorithm

The following algorithm yields the intersectional closure of a natural class system $(\mathcal{C}, \Sigma)$. A proof by induction is given in the Appendix.

Ensure: $\mathcal{C}_\cap$ is the intersectional closure of the input $\mathcal{C}$

$\mathcal{C}_\cap \leftarrow \{\Sigma\}$
$\mathcal{Q} \leftarrow \mathcal{C}$

while $\mathcal{Q} \neq \varnothing$ **do**
  $X \leftarrow$ DEQUEUE($\mathcal{Q}$)
  if not $X \in \mathcal{C}_\cap$ **then**
    APPEND($\mathcal{C}_\cap$, X)
    for $Y \in \mathcal{C}_\cap \setminus \{X\}$ **do**
      ENQUEUE($\mathcal{Q}, X \cap Y$)
    end for
  end if
end while

3.4 Parenthood in the intersectional closure

As we will see shortly, the advantage of explicitly computing the intersectional closure is that *a new feature is required for all and only the classes which have a single parent* (in the intersectional closure). The core reason for this is that if a class has two parents, it must be their intersection. We prove this here.

Theorem 2: Let $(\mathcal{C}, \Sigma)$ be a natural class system and $Y \in \mathcal{C}_\cap$. Then PARENTS(Y) $= \{X_1, X_2\}$ implies $Y = X_1 \cap X_2$.

Proof: First, we show $Y \subset X_1 \cap X_2$. This follows trivially from the definition of parenthood: X_1 is a parent of Y implies $Y \subset X_1$, X_2 is a parent of Y implies $Y \subset X_2$, and so every element in Y is in both X_1 and X_2.

Next, we show that $X_1 \cap X_2 \subset Y$. Suppose not, i.e. $\exists x \in (X_1 \cap X_2) \setminus Y$. As we already showed, $Y \subset (X_1 \cap X_2)$, and $(X_1 \cap X_2) \in \mathcal{C}_\cap$. Of course $X_1 \cap X_2 \subset X_1$ follows from fundamental properties of sets, and $X_1 \in \mathcal{C}_\cap$ by hypothesis. These may be combined to yield $Y \subset (X_1 \cap X_2) \subset X_1$, so that $(X_1 \cap X_2)$ 'intervenes' between Y and X_1. But this implies that Y cannot be a daughter of X_1, a contradiction. Thus, $X_1 \cap X_2 \subset Y$ and $Y \subset X_1 \cap X_2$, proving that $Y = X_1 \cap X_2$.

By very similar logic as in Theorem 2, it is straightforward to show that if a class in $\mathcal{C}_\cap$ has *more* than 2 parents, then it must be the null set. It is similarly easy to show that Σ is the only class in $\mathcal{C}_\cap$ with 0 parents. Thus, every nonempty class in the intersectional closure of $\mathcal{C}$ besides Σ has either 1 parent or 2, and in the latter case, it is the intersection of those two parents. We illustrate this point in the next subsection by by graphing the intersectional closure of the vowel system in Fig. **??**.

3.5 Illustration of the intersectional closure

TODO: add figure

4 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 $Y \setminus X$

Require: $\mathcal{V} = \{+, -, 0\}$
Require: Precompute $C' < -\textsc{IntersectionalClosure}(C)$
Require: $\mathcal{C}'$ is sorted in decreasing order of size $(\forall i, j\,[i < j \rightarrow |C'_i| \geq |C'_j|)$
Require: parent matrix P ($|C'| \times |C'|$): $P_{ij} = 1$ if $C'_j \subset C'_i \wedge \neg\exists k[C'_j \subset C'_k \subset C'_i]$, 0 otherwise
Ensure: output featurization $\mathcal{F} = \{f_j\}_{j=1}^{M}$ such that $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$
classQueue $\leftarrow \{c_k \in C' | \sum_{j=1}^{|C'|} P_{jk} = 1\}$
$i \leftarrow 1$

while classQueue $\neq \varnothing$ **do**
  $c_k \leftarrow \textsc{Pop}$(classQueue)
  parent $= \{c_i \in C | P_{ik} = 1\}$
  $c'_k \leftarrow$ parent $\setminus c_k$
  if $c'_k \in C$ **then**

$$f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ - & \text{if } x \in c'_k \\ 0 & \text{otherwise} \end{cases}$$

    classQueue $\leftarrow \{x \in \text{classQueue} | x \neq c'_k\}$
  else

$$f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ 0 & \text{otherwise} \end{cases}$$

end if
$\mathcal{F} \leftarrow \mathcal{F} \cup f_i$
$i \leftarrow i + 1$
end while

5 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)

Require: $\mathcal{V} = \{+, -, 0\}$
Require: Precompute $C' < -\textsc{IntersectionalClosure}(C)$
Require: $\mathcal{C}'$ is sorted in decreasing order of size $(\forall i, j\,[i < j \rightarrow |C'_i| \geq |C'_j|)$
Require: parent matrix P $(|C'| \times |C'|)$: $P_{ij} = 1$ if $C'_j \subset C'_i \wedge \neg\exists k[C'_j \subset C'_k \subset C'_i]$, 0 otherwise
Ensure: output featurization $\mathcal{F} = \{f_j\}_{j=1}^{M}$ such that $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$

classQueue $\leftarrow \{c_k \in C' | \sum_{j=1}^{|C'|} P_{jk} = 1\}$
$i \leftarrow 1$

while classQueue $\neq \varnothing$ **do**
$c_k \leftarrow \textsc{Pop}$(classQueue)
parent $= \{c_i \in C | P_{ik} = 1\}$
$c'_k \leftarrow$ parent $\setminus c_k$

$$f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ - & \text{if } x \in c'_k \\ 0 & \text{otherwise} \end{cases}$$

$C' \leftarrow (C' \cup \{c'_k\})$
$\textsc{Recalculate}$ P from C'
classQueue $\leftarrow \{c_l \in$ classQueue$| c_l \neq c'_k \ \wedge \sum_{j=1}^{|C'|} P_{jl} = 1\}$
$\textsc{Sort}$(classQueue)
$\mathcal{F} \leftarrow \mathcal{F} \cup f_i$
$i \leftarrow i + 1$
end while

6 Full specification

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

Require: $\mathcal{V} = \{+, -\}$
Require: Precompute $C' < -\textsc{IntersectionalClosure}(C)$
Require: $\mathcal{C}'$ is sorted in decreasing order of size $(\forall i, j\, [i < j \rightarrow |C'_i| \geq |C'_j|)$
Require: parent matrix P $(|C'| \times |C'|)$: $P_{ij} = 1$ if $C'_j \subset C'_i \wedge \neg\exists k[C'_j \subset C'_k \subset C'_i]$, 0 otherwise
Ensure: output featurization $\mathcal{F} = \{f_j\}_{j=1}^{M}$ such that $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$

classQueue $\leftarrow \{c_k \in C' | \sum_{j=1}^{|C'|} P_{jk} = 1\}$
$i \leftarrow 1$

while classQueue $\neq \varnothing$ **do**
  $c_k \leftarrow \textsc{Pop}$(classQueue)
  $c'_k \leftarrow \Sigma \setminus c_k$
  $f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ - & \text{if } x \in c'_k \end{cases}$
  $C' \leftarrow (C' \cup \{c'_k\})$
  $\textsc{Recalculate}$ P from C'
  classQueue $\leftarrow \{c_l \in \text{classQueue} | c_l \neq c'_k \ \wedge \sum_{j=1}^{|C'|} P_{jl} = 1\}$
  $\mathcal{F} \leftarrow \mathcal{F} \cup f_i$
  $i \leftarrow i + 1$
end while

.1 Proof of Lemma 1

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 $C_j = \langle \mathbf{e}_j \rangle$. 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.

.2 Proof of intersectional closure algorithm

We will show that the algorithm generates every class in the intersectional closure by induction. $\mathcal{C}_\cap$ is initialized to contain Σ. Moreover, $\mathcal{Q}$ is initialized to contain every class in $\mathcal{C}$. Each of these must be 'transferred' to the intersectional closure because they do not belong to it already (dequeued from $\mathcal{Q}$, and appended to $\mathcal{C}_\cap$). This demonstrates that every intersection of 0 classes (Σ) and 1 class from $\mathcal{C}$ (namely, $\mathcal{C}$ itself) belongs to $\mathcal{C}_\cap$. Now, suppose that the algorithm has guaranteed that every intersection of n classes from $\mathcal{C}$ is in $\mathcal{C}_\cap$. If there exists a $Y \in \mathcal{C}_\cap$ which can be written as the intersection of $n+1$ classes, i.e. $Y = X_1 \cap X_2 \cap \ldots \cap X_{n+1} = Y' \cap X_{n+1}$ where $Y' = X_1 \cap X_2 \cap \ldots \cap X_n$. Since every intersection of n classes is in $\mathcal{C}_\cap$, Y' must be in $\mathcal{C}_\cap$. Now, regardless of whether X_{n+1} was transferred from $\mathcal{Q}$ to $\mathcal{C}_\cap$ before or after Y' was, there was some point at which one was in $\mathcal{Q}$ and the other in $\mathcal{C}_\cap$. When the **for** loop dequeued the one in $\mathcal{Q}$, it added the intersection of this one with all others in $\mathcal{C}_\cap$ – i.e. $Y' \cap X_{n+1}$. Either this class was already in $\mathcal{C}_\cap$, or else it was not; and in the latter case, it was transferred. Thus, all sets generated by the intersection of $n+1$ classes from $\mathcal{C}$ are in $\mathcal{C}_\cap$. This completes the proof.

.2.1 Representing subset relations as matrices

Any directed graph like Fig **??** 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 expresses the daughterhood relation: $D_{ij} = T$ means C_j is a daughter of C_i.[1] The daughterhood relation is of central importance to the featurization algorithms

[1] 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*.

described later.