Computer-Assisted Language Comparison in Practice

Tutorials on Computational Approaches to the History and Diversity of Languages

Generating Phonological Feature Vectors with SoundVectors and CLTS

Arne Rubehn
Chair for Multilingual Computational Linguistics
University of Passau

The recently published Python library `soundvectors` offers a simple and robust method to derive phonological feature vectors for any valid IPA sound via its canonical description. It is designed to interact neatly with the Cross-Linguistic Transcription Systems reference catalog (CLTS), which dynamically parses valid strings in phonetic transcription to describe speech sounds. This study illustrates how both systems can be used together to generate phonological feature vectors for all kinds of sounds without relying on a previously defined lookup table. Additionally, it compares the generated feature vectors with those obtained from two other prominent databases, PanPhon and PHOIBLE, showing how those systems can be accessed from the CLTS data via its Python API `pyclts`.

1 Introduction

Phonological features are a core concept in phonology, and they are probably taught in virtually every introduction to phonology. The idea is simple, yet powerful: Sounds have certain acoustic and articulatory properties and can be represented as the combination of these. Features are basically a binary notion of those properties, allowing for only two values, present (+) or absent (-). This logically dissects sound inventories into natural classes: [m] is related to [n] and [ŋ] as they are all [+nasal], it shares the [+lab] feature with sounds like [p] or [f], while the [+son] feature bundles it together with other sonorants like [l] or [r].

Features thus allow us to assess the dimensions among which certain sounds are similar or dissimilar to each other. This has made them a popular choice for handling phonetic transcriptions in computational models. It gives them access to properties of and relationships between sounds that goes beyond simply checking for equivalence. As an

Computer-Assisted Language Comparison in Practice
Volume 7, Number 2
URL: https://calc.hypotheses.org/7224 DOI: 10.15475/calcip.2024.2.2
Published under a Creative Commons Attributions 4.0 LICENSE
Published on 05/08/2024

example, consider the sounds [p] and [p^h]. A human linguist instantly sees two very similar sounds, they might even be slightly different transcriptions for the exactly same sound. A computer, however, only "sees" two different strings -- [p] would therefore be represented as equivalently (dis-)similar to [p^h] as to [a]. Representing sounds in terms of their features, however, gives the computer access to the information that some sounds are much more similar to each other than others.

Due to their simplicity and efficiency, features have been successfully applied to enhance computational models for linguistic research and natural language processing alike. The former includes questions in historical language comparison (Kondrak 2000), dialectology (Nerbonne and Heeringa 1997), phonological rule induction (Gildea and Jurafsky 1996), and child language acquisition (Somers 1998). The latter is mainly concerned with speech synthesis (Lux and Vu 2022; Staib et al. 2020) and recognition (Metze 2007). Transliteration and named entity recognition tasks have furthermore shown that features are very useful in transfer learning settings, where knowledge from a high-resource language is applied to a low-resource language (Mortensen et al. 2016).

Due to the broad applicability of features and the constantly increasing amount of machine-readable (cross-)linguistic data, several catalogues have been published that offer feature representations for a large collection of sounds. The most popular feature catalogue in computational linguistics is PanPhon (Mortensen et al. 2016, https://pypi.org/project/panphon/), which covers approximately 5,000 sounds and has been used for several NLP applications. Another well-known catalogue - albeit less for the mere feature representations - is PHOIBLE (Moran and McCloy 2019, https://phoible.org), a database for phoneme inventories across the world's languages. PHOIBLE's current version (2.0) covers roughly 3,000 sounds and assigns feature representations to all of them.

In the following, I will briefly outline SoundVectors (Rubehn et al. 2024, https://pypi.org/project/soundvectors), our recently published tool for generating phonological feature vectors. Thereafter I will provide some code examples on how to use the Python library, and ultimately draw a comparison to PanPhon and PHOIBLE.

2 The SoundVectors Library

At first glance, SoundVectors might seem to be yet another library for translating sounds into phonological features. However, there is one crucial aspect that it introduces: It is *generative*. Other tools (like the ones mentioned above) are closed systems, in that they define feature representations for a fixed-size set of sounds. When queried, they simply look up if there is a feature representation for the queried sound and return it.

However, experience in working with cross-linguistic data shows that new, unobserved sounds are frequently encountered with increasing amounts of data (Moran 2012). In many cases, there are just many different ways to write the same sounds: [p^{hwj}], [p^{hjw}], and [p^{jwh}] obviously mean the same thing, but they are different strings that would all need to be accounted for by a closed system. The human linguist, in contrast, does not see such a complex string as one, inseparable unit, but interprets the base sound and the indicated coarticulations independently.

The generative process behind SoundVectors essentially emulates the logic that human linguists apply to come up with feature representations for sounds. We know that a stop is defined as [-cont, -son], or that all vowels are [-cons]. Therefore, SoundVectors generates feature vectors directly from the *descriptors* of a sound (e.g. "stop", "alveolar", "mid-open"). The system therefore requires the sound to already come in its canonical IPA description. The recommended workflow for that is to use the Cross-Linguistic Transcription System (CLTS; https://clts.clld.org, List et al. 2024), which I describe in the following section, but it can be used in combination with other transcription systems as well.

3 Using SoundVectors with CLTS

I proceed by briefly illustrating how to use `soundvectors` in harmony with CLTS. As with all Python projects, it is recommendable to use a fresh virtual environment to avoid nasty dependency clashes. Then, all we need to do is install `soundvectors` and `pyclts` via `pip`:

```
$ pip install soundvectors
$ pip install pycldf
```

Note that `pyclts` also requires you to have a local copy of the CLTS data on your machine. In case you are unfamiliar with the process, please check its README for detailed instructions.

Beyond these two packages, no further dependencies are required for obtaining feature vectors. Therefore, after successfully installing both packages, we are ready to go. Import the core classes as follows:

```
from soundvectors import SoundVectors
from pyclts import CLTS
```

The `CLTS` class contains all kinds of data and information about different transcription systems, as well as a dynamic parsing algorithm (which is the crucial functionality for our purposes). We are only interested in IPA transcriptions, so we "extract" the `bipa` (standing for 'broad ipa') transcription system from the `CLTS` object:

```
# set up CLTS object and extract the bipa transcription system
bipa = CLTS().bipa

# query bipa
bipa["t"]  # returns <pyclts.models.Consonant: voiceless alveolar stop
consonant>
```

As illustrated above, this `bipa` object can now be used to query IPA sounds - it tells us that the string `"t"` denotes the voiceless alveolar stop consonant. This piece of information is crucial for `SoundVectors`, since it internally maps descriptors (like voiceless or stop) onto phonological feature representations.

Luckily, you won't need to worry about that, since SoundVectors takes care of that under the hood. All you have to do is pass `bipa` as your transcription system of choice when constructing the `SoundVectors` object:

```
sv = SoundVectors(ts=bipa)
```

Now you can easily query `sv` and it will generate a feature vector for any sound which is presented in valid IPA notation. Indexing the object will return you a `FeatureBundle` object, which offers several representations of the generated feature vector:

```
feature_bundle = sv["t"]

feature_bundle.cons
> 1

feature_bundle.as_set()
> frozenset({'-son', '-distr', '-cont', '-lab', '-lo', '-long', '+front',
'-laryngeal', '-syl', '-delrel', '-voi', '-round', '+cons', '-velaric',
'-dorsal', '-back', '-nas', '-pharyngeal', '+ant', '+cor', '-cg', '-sg',
'-lat', '-hi'})

str(feature_bundle)
> '+cons,-syl,-son,-cont,-delrel,-lat,-nas,-voi,-sg,-cg,-pharyngeal,-
laryngeal,+cor,-dorsal,-lab,-hi,-lo,-back,+front,0_tense,-round,-
velaric,-long,+ant,-
distr,0_strid,0_hitone,0_hireg,0_loreg,0_rising,0_falling,0_contour,0_bac
kshift,0_frontshift,0_opening,0_closing,0_centering,0_longdistance,0_seco
ndrounded'

feature_bundle.as_vector()
> (1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1,
1, 0, -1, -1, -1, 1, -1, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0)
```

In essence, the `FeatureBundle` object allows you to switch between all kinds of representation, ranging from purely numeric vector representations to verbose, humanreadable string representations. Since the vector representation is arguably the most useful for downstream tasks, there is also a shorthand method to directly obtain the vector from `sv`:

```
sv.get_vec("t")
> (1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1,
1, 0, -1, -1, -1, 1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Finally, tokenized forms in cross-linguistic data are often represented as lists of tokens, e.g. `["n", "aɪ", "s"]`. For this common case, `SoundVectors` offers a convenience method to generate feature vectors for a batch of sounds at once:

```
word = ["n", "aɪ", "s"]

sv(word)
> [
  (1, -1, 1, ..., 0),
  (-1, 1, 1, ..., -1),
  (1, -1, -1, ..., 0)
]
```

4 Comparison With Other Feature Systems

I proceed by briefly showcasing how other catalogues can be accessed from within CLTS in order to compare them to `SoundVectors`. More precisely, the two well-known feature systems of PanPhon (Mortensen et al. 2016) and PHOIBLE (Moran and McCloy 2019) are being compared directly to `SoundVectors` with regard to their distinctiveness, i.e. how many different sounds are actually represented by distinct vectors.

Again, the first step is to import the relevant packages and set up the `CLTS` and `SoundVector` instances:

```
from pyclts import CLTS
from soundvectors import SoundVectors
from collections import defaultdict
from tabulate import tabulate

clts = CLTS()
sv = SoundVectors(ts=clts.bipa)
```

So far, this is exactly what we have already seen in the previous section. Note that there are two further imports which will come in handy later, but you don't need to worry about them now - both packages should already be installed in your virtual environment if you have successfully installed `pyclts` and `soundvectors`.

Next, I define a method that retrieves all relevant transcription data from external catalogues using CLTS. The `id` parameter represents the internal identifier of a catalogue, which will be `"panphon"` and `"phoible"` in our case.

```python
def load_data(id):
    transcriptiondata = clts.transcriptiondata(id)
    unique_sounds = set(transcriptiondata.sounds)
    unique_sounds.remove("<NA>")
    return unique_sounds, transcriptiondata.data
```

Each catalogue (`transcriptiondata`) in CLTS has two main attributes, `sounds` and `data`. The former spans the sound inventory covered by the respective system, normalized according to CLTS conventions; the latter contains all sorts of additional data, including feature representations, which is obviously the part that we are interested in later on. In this method, I cast the sounds into a set to avoid duplicates and remove `<NA>` values, sounds that are not recognized by CLTS.

I define a second convenience method that groups a set of sounds by their feature representation. This allows me to count the equivalence classes, i.e. how many unique feature representations exist among all the sounds, and therefore how many sounds share their feature representation with at least one other sound.

```python
def map_vector_to_sounds(sounds, vector_func):
    features_to_sound = defaultdict(list)
    for s in sounds:
        features_to_sound[vector_func(s)].append(s)

    return features_to_sound, len(features_to_sound), len(sounds)
```

The method obtains the feature representation for every sound in `sounds` by calling the specified function `vector_func`, and then maps the unique feature vectors to the sounds they represent. The length of the resulting dictionary `features_to_sound` is therefore the number of unique feature vectors, while the length of `sounds` obviously is just the size of the present sound inventory.

Finally, we can put the pieces together and compare the distinctiveness of SoundVectors to PanPhon and PHOIBLE respectively:

```python
table = []
for system in ["phoible", "panphon"]:
    unique_sounds, data = load_data(system)
    _, num_unique_vecs_own, num_unique_sounds =
            map_vector_to_sounds(unique_sounds, lambda x:
            data[x][0]["features"])
    _, num_unique_soundvecs, _ = map_vector_to_sounds(unique_sounds,
            sv.get_vec)
    table.append([system.upper(), num_unique_sounds, num_unique_vecs_own,
          num_unique_soundvecs])

print(tabulate(table, headers=["SOURCE", "UNIQUE SOUNDS", "UNIQUE VECTORS
(OWN)", "UNIQUE VECTORS (SOUNDVEC)"]))
```

This code prints the following table as output:

SOURCE	UNIQUE SOUNDS	UNIQUE VECTORS (OWN)	UNIQUE VECTORS (SOUNDVEC)
PHOIBLE	2757	2341	1760
PANPHON	5619	3089	3747

So how can we interpret these results? First, one can see that PHOIBLE covers much less sounds than PanPhon, only around half of its inventory. The reason for the difference lies in different purposes of both datasets. PHOIBLE's size of the inventory depends strictly on the phoneme inventories that the dataset covers. Only those sounds are listed that can be shown to have been proposed for a concrete phoneme inventory. The lower number of sounds thus simply reflects the documentation status of PHOIBLE. PanPhon, on the other hand, was designed for computational tasks, where sound vectors are provided for arbitrary sounds in phonetic transcription data. The idea was to provide a very large number of theoretically possible sounds in order to be able to cope with unseen phonetic transcription data.

The comparison between PanPhon and SoundVectors reveals that the latter system has a higher degree of distinctiveness, generating ~700 more unique feature representations for the given alphabet of 5,619 sounds. At first glance, this results seems obvious when considering the feature inventory sizes: SoundVectors defines 39 features, allowing for much higher combinatoric power than the 25 features defined in PanPhon. These raw numbers, however, are misleading, since in the end, SoundVectors and PanPhon share the same features to describe simple sounds. The difference between the feature inventory sizes has to be attributed mainly to the fact that SoundVectors can describe complex sounds like contour tones, diphthongs, and consonant clusters, for which a number of features was added. Since PanPhon does not cover those sounds, the system is not designed to describe them and thus lacks appropriate features. The 5,619 sounds in question here, however, are directly derived from PanPhon itself -- so there is no complex sound among them, rendering (at least) 10 features from SoundVectors completely unused in this context. That effectively reduces the employed feature inventory from SoundVectors to a size that is fairly comparable to PanPhon, while still maintaining a higher degree of distinctiveness.

Arguably more curious, however, is the comparison between PHOIBLE and SoundVectors. With 37 features, PHOIBLE defines a feature inventory of a very similar size to SoundVectors, albeit a bit smaller. PHOIBLE also contains complex sounds, so here, all of the features defined in SoundVectors actually have to be used. As a result, PHOIBLE appears to be considerably more distinctive than SoundVectors.

However, the increased distinctivity of PHOIBLE vectors is due to the encoding of complex sounds. Consider the consonant cluster [mb]: The value for the feature "nasal" is `"+,-"`, indicating that the first part of the sound is `[+nasal]`, and the second part is `[-nasal]`. PHOIBLE thus allows for features having multiple values at once. This means, that the feature system in PHOIBLE deviates from the binary (ternary, considering non-applicable features) feature notion. This obviously leads to a drastic increase of possible feature combinations, which explains the distinctivity of PHOIBLE's system -- out of the 2,757 defined sounds, 1,021 contain at least one multi-valued feature. While this feature design may be useful for the purpose for which PHOIBLE was created, it is less useful for downstream tasks in computational linguistics, where numerical vectors are required. When allowing features to have multiple values, the effective vector size would be twice as large (since two slots for each complex sound would have to be reserved for each feature value).

5 Conclusion and Outlook

Feature vectors have already proven to be useful for many applications in computational linguistics and will probably continue to do so. With SoundVectors, we have released a lightweight and robust tool that aims to overcome the major weakness of previous systems by integrating a generative aspect, exploiting the power of CLTS in parsing and describing sounds. This tutorial has shown how SoundVectors can be used in practice and how the feature system can be compared to features systems provided by PanPhon and PHOIBLE. Future work will hopefully allow us to illustrate the benefits of a generative system for the handling of phonetic transcriptions in more detail.

References

Gildea, Daniel & Daniel Jurafsky. 1996. Learning bias and phonological-rule induction. Computational Linguistics, 22(4). 497-530. https://aclanthology.org/J96-4003

Kondrak, Grzegorz. 2000. A new algorithm for the alignment of phonological sequences. In 1st Meeting of the North American Chapter of the Association for Computational Linguistics. https://aclanthology.org/A00-2038

List, Johann-Mattis, Cormac Anderson, Tiago Tresoldi & Robert Forkel. 2024. Cross-Linguistic Transcription Systems [Dataset, Version 2.3.0]. Max Planck Institute for Evolutionary Anthropology, Leipzig. https://doi.org/10.5281/zenodo.10997741

Lux, Florian & Thang Vu. 2022. Language-agnostic meta-learning for low-resource text-to-speech with articulatory features. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long

Papers), 6858-6868. Dublin, Ireland: Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.472

Moran, Steven. 2012. Phonetics Information Base and Lexicon. Phd, University of Washington.

Moran, Steven & Daniel McCloy (eds.). 2019. PHOIBLE 2.0. Max Planck Institute for the Science of Human History, Jena. https://doi.org/10.5281/zenodo.2593234

Mortensen, David R., Patrick Littell, Akash Bharadwaj, Kartik Goyal, Chris Dyer & Lori Levin. 2016. Panphon: A resource for mapping IPA segments to articulatory feature vectors. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, 3475-3484.

Nerbonne, John & Wilbert Heeringa. 1997. Measuring dialect distance phonetically. In Computational phonology: Third Meeting of the ACL Special Interest Group in Computational Phonology.

Rubehn, Arne, Jessica Nieder, Robert Forkel & Johann-Mattis List. 2024. Generating Feature Vectors from Phonetic Transcriptions in Cross-Linguistic Data Formats. In Proceedings of the Society for Computation in Linguistics (SCiL) 2024, 205-216. Irvine, CA. https://doi.org/10.7275/scil.2144

Somers, Harold. 1998. Similarity metrics for aligning children's articulation data. In 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2, 1227-1232. https://doi.org/10.3115/980432.980769

Staib, Marlene, Tian Huey Teh, Alexandra Torresquintero, Devang S. Ram Mohan, Lorenzo Foglianti, Raphael Lenain & Jiameng Gao. 2020. Phonological Features for 0-shot Multilingual Speech Synthesis. In Proceedings of INTERSPEECH 2020, Shanghai, China. https://doi.org/10.21437/interspeech.2020-1821

Supplementary Material
Data and code can be found at https://github.com/calc-project/soundvectors-blogpost
Acknowledgements
Many thanks to Johann-Mattis List for his helpful remarks and assistance in typesetting and publishing.
Funding Information
This project has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (Grant agreement No. 101044282). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.