Developing a Comprehensive Framework for Multimodal Feature Extraction

Quinten McNamara
University of Texas at Austin
Austin, Texas
quinten.mcnamara@utexas.edu

Alejandro De La Vega
University of Texas at Austin
Austin, Texas
delavega@utexas.edu

Tal Yarkoni*
University of Texas at Austin
Austin, Texas
tyarkoni@utexas.edu

1 INTRODUCTION

Feature extraction and input annotation are critical elements of many machine learning and data science applications. It is common for data scientists to first extract low-level or high-level semantic features from video, text, or audio input, and subsequently feed these features into a statistical model [6, 12, 13]. In many settings, the quality of one's features or annotations can be a greater determinant of an application's success than any subsequent analysis or modeling decisions. For example, the capacity of sentiment analysis models to make sense of speech transcripts extracted from human conversations is inherently limited by the quality of the underlying speech-to-text transcription. If a transcript looks like word salad to human eyes, it is unlikely to be rehabilitated by artificial ones. Fortunately, groundbreaking advances in computer vision, speech recognition, and other domains of artificial intelligence have led to a recent proliferation of highly performant, publicly accessible feature extraction services. Cloud-based APIs developed by major companies such as Google, IBM, and Microsoft, as well as hundreds of smaller startups, now allow data scientists to extract rich annotations from videos, images, audio, and text cheaply and at scale. Consequently, the once monumental challenge of extracting near-human-level feature annotations now appears to be, if not completely overcome, then at least substantially mitigated.

In practice, however, a number of important technical barriers remain. While the proliferation of high-quality feature extraction services has made state-of-the-art machine learning technology widely accessible, harnessing this technology in an efficient way often remains a laborious and confusing process. Which of the hundreds of available speech-to-text, face recognition, or sentiment analysis services or tools should one use? How can one efficiently combine the features extracted using different tools into a cohesive pipeline? And what can one do to ensure that one's codebase remains operational in the face of frequent changes to third-party APIs? There are, at present, no easy answers to such questions; consequently, data scientists developing pipelines that involve extensive feature extraction are often forced to write and maintain highly customized and potentially large codebases. The lack of standardization with respect to feature extraction and data annotation is, in our view, one of the most pervasive and detrimental problems currently facing the applied data science community.

Here we introduce a new feature extraction framework designed to address this critical gap. *Pliers* (available at https://github.com/tyarkoni/pliers) is an open-source Python package that provides a standardized, easy-to-use interface to a wide range of feature extraction tools and services. In contrast to previous feature extraction tools, pliers is inherently multimodal: it supports annotation of diverse data types (video, images, audio, and text), and can dynamically convert between data types as needed (e.g., implicitly transcribing speech in video clips in order to enable application of text-based feature extractors). The package is designed with the twin goals of ease-of-use and extensibility in mind: users can apply a wide range of pre-existing feature extraction tools to their data in just a few lines of Python code, and can also easily add their own custom extractors by writing modular classes. A graph-based API allows users to very rapidly develop feature extraction pipelines that output results in a single, standardized format. Thus, pliers can be easily inserted into many existing data science workflows, significantly reducing development time while increasing code clarity and maintainability.

The remaining sections are organized as follows. In section 2, we review relevant previous work, focusing on existing feature extraction toolkits. In section 3, we describe the architecture of the pliers package and highlight several core features with potentially beneficial implications for applied data science workflows. We then demonstrate the flexibility and utility of the package in section 4, where we use pliers to rapidly extract and analyze a broad range of features from the movie stimuli in a large functional neuroimaging dataset. We conclude with a brief discussion of the implications of our work and planned future directions.

2 BACKGROUND

In recent years, major technology companies (e.g., Google and IBM) and smaller startups (e.g., Clarifai and Indico) have released dedicated machine learning services for various computer vision and signal processing tasks. These services can individually be thought of as feature extraction frameworks, each with its own advantages and disadvantages. While these services span an enormous range of features—including everything from face recognition to musical genre detection to part-of-speech tagging—the ability to systematically compare them, or to combine them into cohesive analysis pipelines, is limited by practical considerations. Because each service or tool has its own API, usage requirements, and/or client libraries, there is often considerable overhead involved in accessing and using even a single service, let alone chaining multiple services into integrated pipelines for multimodal feature extraction (e.g.,

*Corresponding author.

KDD '17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08.
https://doi.org/10.1145/3097983.3098075

running sentiment analysis on text transcribed from speech audio, in addition to annotating visual sentiment cues).

In an effort to provide common interfaces to different tools, several authors have previously introduced toolboxes that support multiple feature extraction tools. For example, the *SpeechRecognition* package in Python [21] supports API calls to several audio transcription services via a single interface; the *MIRToolbox* [11] supports extraction of both low-level and high-level audio features; the *MMFEAT* feature extraction toolkit [9] provides an abstraction for a number of visual and audio feature extraction tools used in natural language processing (NLP) frameworks (e.g., bag of features models, etc.); and the *feature-extraction* package [8] simplifies extraction of a number of computer vision features commonly used in image classification tasks. Such packages allow their users to call different feature extraction algorithms or services via a standardized interface, reducing development time and avoiding the need to look through each API's documentation for usage examples.

While existing feature extraction packages take important steps towards a unified feature extraction interface, most also have a number of major limitations. First, virtually all previous tools focus on one particular analysis domain—e.g., speech recognition, NLP, or computer vision. We are not aware of any existing toolbox that attempts to cover everything from video to text (though a few support both video and audio—e.g., the feature-extraction package). Second, with a few notable exceptions (e.g., the Essentia audio analysis package [1]), most frameworks cannot be easily extended by end users—either because their code is not released under a permissive open-source license, or because the codebase is not designed with modularity in mind. Third, many frameworks focus primarily on traditional low-level visual or audio features use for classification or discrimination tasks, and lack support for high-level perceptual labels of the kind facilitated by recent advances in deep learning. Lastly, and perhaps most importantly, existing frameworks tend to offer a limited degree of abstraction. While many provide a unified interface for querying multiple services or tools, users are still typically responsible for writing supplementary code to pass inputs and handle outputs, chain multiple services, and so on. Our goal in developing the pliers framework was to address each of the above limitations by providing a standardized feature extraction interface that was (i) multimodal, (ii) comprehensive, (iii) extensible, and (iv) easy to use.

3 ARCHITECTURE

3.1 Overview

With these goals in mind, we sought to develop an intuitive, standardized, and relatively simple API, taking particular cues from the object-oriented transformer interface popularized by the *scikit-learn* machine learning package [3]. At its core, pliers is structured around a hierarchy of `Transformer` classes that each have a modular, well-defined role in either extracting feature information from an input data object (`Extractor` classes), or converting the input data object into another data object (`Converter` classes). As in scikit-learn, the defining feature of a `Transformer` class in pliers is its implementation of a `.transform()` method that accepts a single input. The input data types are represented using a separate `Stim` (short for stimulus) hierarchy that includes subclasses such as `VideoStim`, `AudioStim`, and so on. One can thus think of pliers workflows as tree-like representations in which `Stim` objects are repeatedly transformed as they flow through a graph, culminating in one or more feature extraction steps at the terminal node(s).

The easiest way to illustrate these principles is with a short Python code sample:

```
video = VideoStim('my_video.mp4')
conv = FrameSamplingConverter(hertz=1)
frames = conv.transform(video)
ext = GoogleVisionAPIFaceExtractor()
results = ext.transform(frames)
```

Listing 1: A simple pliers example

This simple snippet of code reads in a video file, converts it into a series of static image frames (sampling frames at a rate of 1 per second), and then uses the Google Cloud Vision API to perform face recognition on each image, returning the aggregated results from all images in a standardized format that can be easily operated on (see 3.3.4). Thus, with very little configuration and code, users are able to perform a complex feature extraction operation using a state-of-the-art face detection API. The benefits of pliers' standardized `Transformer` API can be further appreciated by considering that we could seamlessly replace the `GoogleVisionAPIFaceExtractor` call in Listing 1 with one of the many other pliers Extractor classes that support image inputs—for example, a `ClarifaiAPIExtractor` that performs object recognition using the Clarifai service (clarifai.com), a `TensorFlowInceptionV3Extractor` that labels images using a pre-trained Tensor Flow inception model [16], and so on. Moreover, as discussed later (3.3.4), pliers also provides a higher-level graph API that allows users to compactly specify and manage large graphs potentially involving dozens of transformation nodes.

3.2 Implementation in Python

Pliers is implemented in the Python programming language, and relies heavily on popular Python scientific computing libraries (most notably, numpy, scipy, and pandas). This choice reflects the widespread usage of Python within the data science community, with many open source libraries available for a variety of tasks. In particular, a large number of feature extraction tools and services are either implemented in Python, or have high-level python bindings or client libraries—greatly simplifying our goal of providing a common interface. The high-level, dynamic nature of the language further facilitates our primary goal of usability and simplicity while remaining relatively performant. In addition to its core Python dependencies, pliers also depends on the *moviepy* Python package for video and audio file manipulation, which in turn depends on the cross-platform ffmpeg library. Many of the individual feature extractors supported in pliers also have their own dependencies (e.g. the Google Cloud Vision extractors rely on the google-api-python-client library). Pliers is maintained under public version control on GitHub (https://github.com/tyarkoni/pliers). Development follows best practices in the open-source community: we

perform continuous integration testing, maintain relatively comprehensive documentation, and include several interactive Jupyter notebooks that exemplify a range of uses of the pliers API. Pliers is platform-independent, though it is tested primarily on Linux and OS X environments.

3.3 Core Features

Pliers has a number of core features that distinguish it from previous feature extraction toolboxes, and can help significantly simplify and streamline data scientists' feature extraction pipelines. These include (i) support for a wide range of input modalities; (ii) breadth of feature extractor coverage; (iii) a highly extensible design; and (iv) a high degree of abstraction. We discuss each of these in turn.

3.3.1 Multimodal support. Pliers is expressly designed to support multimodal feature extraction from a wide range of input modalities. At present, pliers supports four primary input data types: video, image, audio, and text. Each of these modalities is supported by a hierarchy of classes covering specific use cases (e.g., the base `ImageStim` class has a `VideoFrameStim` subclass that represents image frames extracted from video clips). Pliers also implements a generic `CompoundStim` class that provides slots for any other type of `Stim` class, allowing users to create custom inputs to feature extractors that require a combination of different input types. Importantly, and as discussed in more detail in 3.3.4, pliers allows users to seamlessly convert between different `Stim` classes, making it easy to build complex graphs that integrate feature extractors from multiple modalities. Collectively, the supported input types cover many common use cases in applied data science and scientific research settings (we provide a sample application to functional neuroimaging data in section 4)—and as described in 3.3.3, the `Stim` hierarchy can be easily extended to support new use cases.

3.3.2 Breadth. Pliers seeks to provide unified access to a potentially very large set of feature extraction tools and services. At present, the package supports over 30 different extractors, many of which individually support a large number of more specific models. The current selection of supported extractors has been largely motivated by a proof-of-concept desire to demonstrate the breadth of features that pliers can provide access to; as detailed in the next section, we expect the library of supported tools and services to increase rapidly. Here we describe a partial selection of implemented feature extractors that run the gamut from low-level perceptual analysis to high-level semantic annotations. These include:

- Speech-to-text converters, including wrappers for most of the APIs implemented in the *SpeechRecognition* package [21], in addition to a custom transcription converter that uses IBM's Watson speech-to-text API and has the particular advantage of providing onsets for individual words.
- A `TensorFlowInceptionV3Extractor` that uses a pretrained deep convolutional neural network (CNN) model based on the Inception V3 architecture [17] to label objects in images. Pliers seamlessly handles the download, installation, and execution of the associated TensorFlow model and code in the background, illustrating the package's capacity to support not only API-based services, but also a potentially wide range of pretrained, locally-executed models.
- An `OpticalFlowExtractor` that uses OpenCV's Farneback algorithm to quantify the amount of frame-to-frame optical flow in a video. The availability of Python bindings for OpenCV makes it easy to add any number of other OpenCV-based feature extractors (see 3.3.3).
- Extractors and converters for most of the Google Cloud Vision and Speech API tools—e.g., face detection, object labeling, and speech-to-text conversion. These are some of the most widely used feature extraction services, and pliers makes it considerably easier to apply them to a diverse range of inputs.
- A `PredefinedDictionaryTextExtractor` that maps individual words onto pre-existing dictionaries accessible via the web. Many of these dictionaries were generated via large-scale psycholinguistic studies involving hundreds of human participants rating thousands of common words; thus, pliers provides easy access to lexical variables ranging from word frequency and contextual diversity to age-of-acquisition norms to affective ratings of word valence.
- A `STFTExtractor` that extracts acoustic power in user-configurable frequency bands by applying a short-time Fourier transform to audio input.
- An `IndicoAPIExtractor` that interfaces with the Indico API and supports application of a range of models—including sentiment analysis, emotion detection, and personality inference—to text.

3.3.3 Extensibility. The current set of supported feature extractors represents only a fraction of the coverage we intend to eventually provide. We are continuously adding support for new feature extractors and new data types, and encourage others to as well. To date, we have placed emphasis on adding support for (i) high-level APIs capable of providing human-like annotations (e.g., presence of object in an image, sentiment of text), and (ii) other feature extraction toolkits or services that already themselves support multiple extractors (e.g., the Google Cloud Vision APIs, or the Python SpeechRecognition package). However, in principle, virtually any feature extraction tool that operates over one of the supported input data types can be implemented in pliers. To encourage contributions, we have developed pliers as a highly modular, easily-extensible, object-oriented framework. Listing 2 provides an example of a simple but fully functional `Extractor` that takes text inputs and extracts the number of characters in each string. While this example is trivial, it illustrates the speed with which a new `Extractor` can be developed and immediately embedded into a pliers workflow. The bare minimum required of a new `Extractor` class is that it (i) specifies the input data type(s) it operates on (text, video, etc.) and (ii) defines a new `_extract()` method that takes a `Stim` as input, and returns a new object of class `ExtractorResult` (see 3.3.4).

```
class LengthExtractor(Extractor):

    _input_type = TextStim

    def _extract(self, stim):
        value = np.array([len(stim.text.strip())])
        return ExtractorResult(value, stim, self,
                               features=['text_length'])
```

Listing 2: Sample code for an Extractor class that counts the number of words in each text input

In many cases, users do not even have to write new extractors in order to add valuable new functionality. Some `Extractor` classes provide access to named resources that are loaded from a static configuration file, and functionality can thus be extended simply by adding new entries. For example, the `PredefinedDictionaryExtractor` mentioned above provides access to many web-based dictionary resources that map individual words onto corresponding numerical values (e.g., psycholinguistic databases that provide word frequency, age-of-acquisition, or emotional valence norms for individual words [2, 10, 20]). New lookup dictionaries of this kind can be added simply by adding new entries to a JSON configuration file bundled with the pliers package. Listing 3 provides a sample (partial) entry from this file that encodes instructions to the `PredefinedDictionaryExtractor` on how to download and preprocess a popular set of age-of-acquisition norms [10].

```
{"aoa": {
    "title": "Age-of-acquisition (AoA) norms for
    over 50 thousand English words",
    "description_url": "http://crr.ugent.be/archives/806",
    "source": "Kuperman, V., Stadthagen-Gonzalez, H.,
    & Brysbaert, M. (2012). Age-of-acquisition ratings
    for 30,000 English words. Behavior Research Methods,
    44(4), 978-990.",
    "url": "http://crr.ugent.be/papers/AoA_51715_words.zip",
    "format": "xls",
    "language": "english",
    "index": "Word"
}}
```

Listing 3: Sample resource available through the PredefinedDictionaryExtractor

In still other cases, harnessing new functionality is even easier than adding new entries to a config file. Because many of the feature extraction APIs that pliers supports allow users to easily control the remote resource or model used in feature extraction, specifying a different resource can be as easy as passing in the appropriate parameter. For example, the `IndicoAPIExtractor`—which provides access to the Indico.io service's text analysis tools—accepts an initialization argument that specifies which of its available models (e.g., 'sentiment', 'emotion', or 'personality') to use. As new models are introduced to the Indico.io service, they automatically become available to pliers users.

Extensibility in pliers is not limited to adding new transformers; new input data types can also be readily added by creating new `Stim` classes—most commonly by simply subclassing an existing `Stim` class. For example, while pliers currently does not provide a built-in interface to social media services such as Facebook or Twitter, our near-term roadmap includes plans to develop tools that make it easy to retrieve tweets and Facebook posts as native pliers objects (e.g., `TweetStim` or `FacebookPostStim` classes that contain metadata in addition to multimedia)—at which point users will be immediately and effortlessly able to apply any of the compatible feature extractors available in pliers.

3.3.4 Abstraction. One of our primary goals in developing pliers is to to create a framework that makes feature extraction as easy as possible, enabling data scientists to develop feature extraction pipelines using clearer, more compact code. To this end, pliers features an extremely high level of abstraction. Many steps that would require explicit routines in other feature extraction packages are performed implicitly in pliers. Here we highlight four particular features in pliers that greatly simplify the feature extraction process in comparison to other feature extraction packages or bespoke pipelines.

Implicit `Stim` conversion. Pliers is capable of implicitly converting between different data types to facilitate feature extraction on an ad hoc basis. For example, suppose one wishes to apply a sentiment extractor to the dialogue in a series of videos. Accomplishing this requires a user to first extract the audio track from the video, and then apply speech-to-text transcription to the audio. Helpfully, pliers is usually able to identify and implicitly execute conversions between different input types, minimizing the user's workload and considerably streamlining the feature extraction pipeline. For example, the two code snippets in Listing 4 produce identical results.

```
# Option A: explicit conversion
conv1 = VideoToAudioConverter()
audio = conv1.transform(video)
conv2 = AudioToTextConverter()
text = conv2.transform(audio)
ext = IndicoAPIExtractor(model='sentiment')
result = ext.transform(text)

# Option B: implicit conversion
ext = IndicoAPIExtractor(model='sentiment')
result = ext.transform(video)
```

Listing 4: Explicit vs. implicit stimulus conversion

Here, pliers implicitly identifies and applies a valid conversion trajectory that transforms a video into text (by stripping the audio track from the video and submitting it to a speech-to-text API). Pliers provides package-level control over which `Converter` to use in cases of ambiguity (e.g., in the above example, the user could specify which of several speech-to-text services they prefer to work with). Moreover, all `Stim` instances retain a comprehensive internal log of applied transformations, allowing the user to easily determine what steps were taken in order to produce the final result.

Native handling of iterable inputs. Every pliers `Transformer` (including all `Extractors` and `Converters`) is inherently iterable-aware, and can be passed an iterable (i.e., a Python list, tuple, or generator) of `Stim` objects rather than

just a single `Stim`. The transformation will then be applied independently to each `Stim`. Furthermore, some `Stim` classes are naturally iterable. For example, a `VideoStim` is made up of a series of `VideoFrameStim`s, and a `ComplexTextStim` is made up of `TextStim`s. For efficiency and memory conservation reasons, the elements will typically be retrieved using Python generator expressions (functions that support lazy evaluation of iterable objects) rather than lists.

Graph API. Pliers also implements a unique graph interface that enables users to effortlessly go from complex stimuli to a potentially large number of features extracted from multiple modalities. The interface allows users to specify `Converter` and `Extractor` nodes through which a stimulus can be run. Suppose we have a video that we want to tag for visual annotation, audio features, lexical variables (e.g., the normative frequency of each spoken word), and spoken word sentiment. This seemingly complex task can be simply configured using the code in Listing 5. Moreover, pliers allows users to easily visualize the executed graph by leveraging the graphviz package; Figure 1 displays the graph generated by the code in Listing 5. Note that, as discussed above, it is usually not necessary to explicitly specify `Stim` conversion steps, as these will be detected and injected implicitly (and hence the diagram in Figure 1 contains many more nodes than were specified in Listing 5).

```
clips = ['video1.mp4', 'video2.mp4']
g = Graph([
    (FrameSamplingConverter(hertz=1),
     ['ClarifaiAPIExtractor',
      'GoogleVisionAPIFaceExtractor']),
    STFTAudioExtractor(hop_size=1, freq_bins=5),
    PredefinedDictionaryExtractor(
      ['SUBTLEXusfrequencyabove1/Lg10WF',
       'concreteness/Conc.M']),
    IndicoAPIExtractor(models=['sentiment'])
])
results = g.run(clips)
```

Listing 5: Sample graph generation code

Consolidated output. One of the major sources of busywork data scientists often face when building feature extraction pipelines is the need to postprocess and reformat the results returned by different feature extraction tools. Pliers addresses this problem by ensuring that all `Extractor` classes output an `ExtractorResult` object. This is a lightweight container that represents all of the extracted feature information returned by the `Extractor` in a common format, and also stores references to the `Stim` and `Extractor` objects used to generate the result. The raw extracted feature values are stored in the `.data` property, but typically, users will want to work with the data in a more convenient format. Fortunately, every `ExtractorResult` instance exposes a `.to_df()` method that returns a formatted pandas DataFrame (a tabular data type widely used in Python data science applications). Additionally, pliers provides built-in tools to easily merge the data from multiple `ExtractorResult` instances; by default, executing a Graph such as the one in Listing 5 will return a single merged pandas DataFrame that contains all extracted feature information for all processed inputs, along with added timestamp and duration columns that facilitate further manipulation, plotting, and analysis of the results.

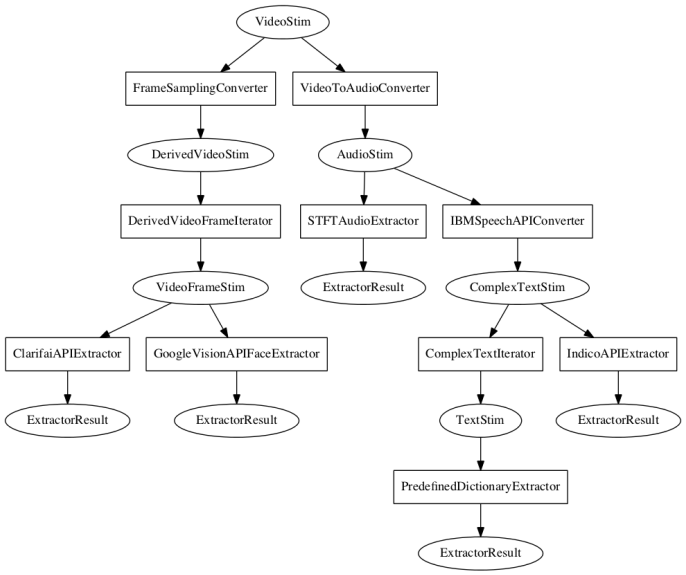

Figure 1: Diagram of the full graph generated by the code in Listing 5. When executed, this graph returns the predictors displayed in Figure 2.

4 APPLICATION TO FUNCTIONAL MRI

To demonstrate the utility of having a standardized, high-level, multimodal feature extraction framework, we applied pliers to the domain of human functional neuroimaging. A primary goal of neuroimaging research is to understand how the brain processes and integrates information from different sensory modalities. In pursuit of this goal, researchers traditionally employ "factorial" designs that experimentally contrast the brain's response to different sets of stimuli which vary systematically along one or more dimensions. For example, to study the neural substrates of face processing, one might compare the brain activity elicited by images of faces to the responses elicited by other image categories such as buildings, animals, or outdoor scenes. Critically, if both sets of images do not differ along irrelevant dimensions—such as luminosity, size or color—differences in brain activity should reflect face-specific processing [7].

Unfortunately, this idealized strategy has several important limitations. Most notably, in practice, it is rarely possible to ensure that the stimuli in experimental conditions vary *only* along the target dimension of interest. Almost certainly, stimuli vary on many additional dimensions that are not of interest, and may confound the neural response, but are omitted from the statistical model. Moreover, by studying the phenomena of interest under highly

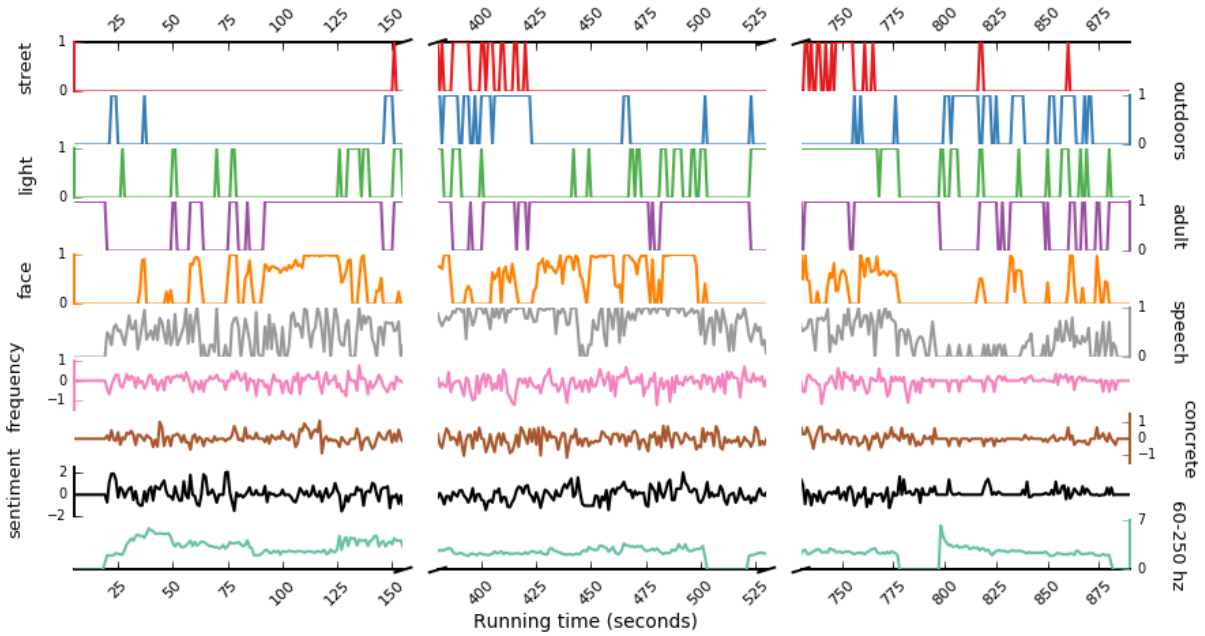

Figure 2: Timeline of automatically extracted features across three periods of the movie stimulus. We used the `ClarifaiAPIExtractor` to label four image classes ('street', 'outdoors', 'light' and 'adult') and the `GoogleFaceAPIExtractor` to determine the probability that a face was present. Next, we applied the `IBMSpeechAPIConverter` to transcribe the movie audio and detect the presence of speech ('speech'). For each transcribed word we used a `PredefinedDictionaryExtractor` to extract lexical norms for word 'frequency' and 'concreteness', and used the `IndicoAPIExtractor` to quantify 'sentiment'. Finally, a `STFTAudioExtractor` was used to quantify acoustic power within the 60-250 hz frequency range (where most human speech is expressed).

controlled, simplifed conditions, experiments often lose their ecological validity—that is, they poorly approximate the complexities of real world scene processing and understanding.

An alternative approach is to use rich, "naturalistic" stimuli, such as audiovisual movies, to examine the dynamic neural response to a large number of potential features. By maximizing the number of predictors in one's statistical model, one can better control for potential confounds, while simultaneously maximizing ecological validity. However, the high cost of manually annotating a large number of features has made such naturalistic approaches relatively rare. Fortunately, pliers allows us to generate large sets of potential predictors rapidly and automatically.

Here, we modeled the brain's responses to a variety of automatically extracted features using publicly accessible data from the seminal Human Connectome Project (HCP [4, 18]). In this study, participants underwent functional magnetic resonance imaging (fMRI) while watching a series of short films, for a total of 60 minutes of audio-visual stimulation. We applied the pliers Graph from Listing 5 to the video stimuli used in the study, resulting in a timeline of feature vectors that span the full duration of the scans (an extract is displayed in Figure 2). We then used the Nipype workflow engine [5]—which provides uniform interfaces to neuroimaging analysis packages such as FSL [15]—to predict brain activity using a subset of 10 automatically extracted features across 35 subjects.

We omit the full details of the fMRI analysis pipeline and statistical model here, as our goal here is to illustrate the practical application of pliers in feature extraction rather than to draw conclusions about the neural correlates of different features. The key point is simply that we model the neural response (technically, the hemodynamic response, which is an indirect measure of neuronal activity) as a linear function of the extracted features. An abridged version of the mixed model we fit is:

$$Y_{it} = \beta_0 + \beta_1 X_{1it} + \ldots + \beta_k X_{kit} + u_{0i} + u_{1i} X_{1it} + \ldots + uki X_{kit} + e_{it},$$

where Y_{it} is the i^{th} participant's neural response at time t, β_0 is a fixed intercept, β_k is the estimated fixed regression coefficient for the k^{th} extracted feature, and X_{kit} is the value of the k^{th} extracted feature at time t in the i^{th} participant, as obtained using pliers. The u terms represents random subject intercepts and slopes intended to account for between-subject variance, and e_{it} is the residual model error. For the sake of brevity, we omit a number of other terms that were included in the actual statistical model in order to account for fMRI-specific modeling concerns (e.g., temporal autocorrelation, low-frequency noise, participant head movement, etc.).

This standard neuroimaging model is fit separately in each of over 200,000 brain voxels, resulting in whole-brain activation maps

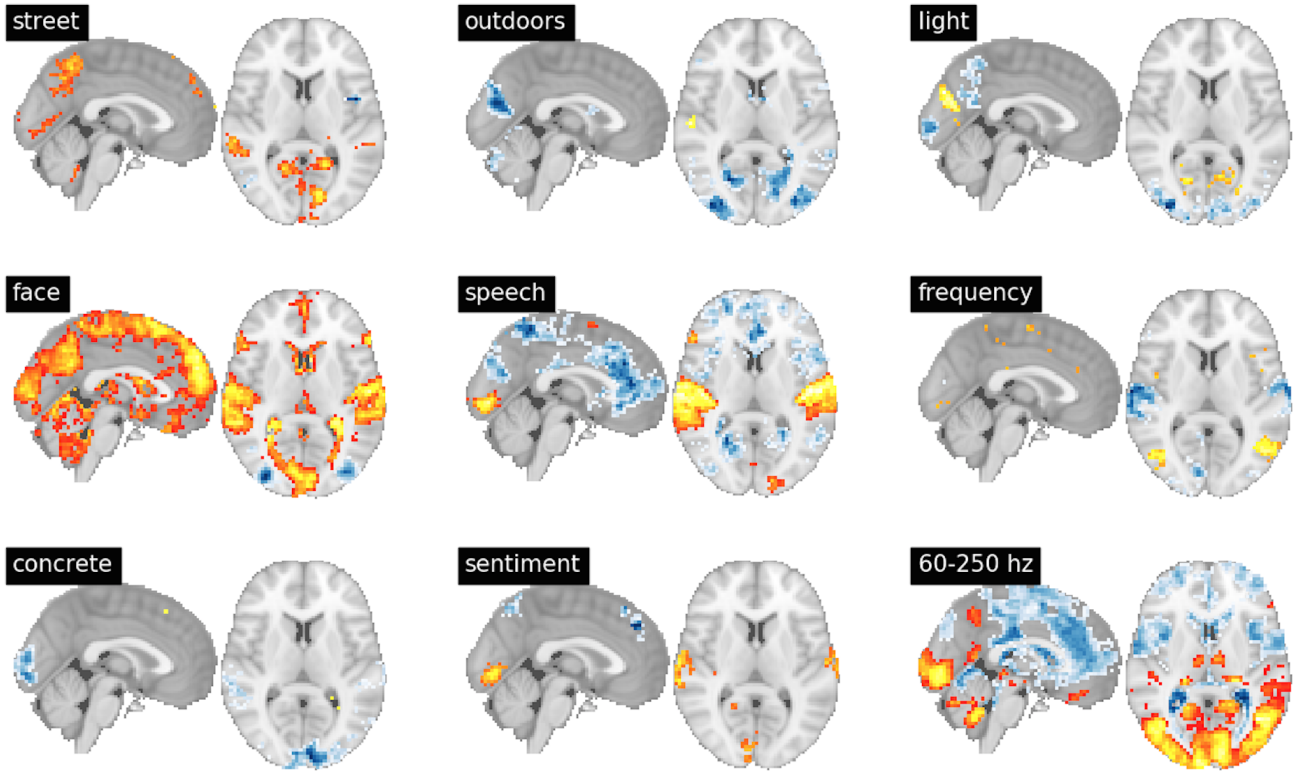

Figure 3: Pattern of brain activity associated with automatically extracted features from an audio visual movie. Labels (e.g., "street", "frequency", etc.) correspond to the rows and descriptions in Figure 2. Yellow areas indicate voxels that were positively associated with the predictor, whereas blue areas indicated voxels with a negative association. More intense colors reflect stronger effects. Maps were thresholded at a statistical threshold of $p < .001$ for display purposes.

displaying those voxels across the brain that were consistently associated (as assessed using a standard significance test, and correcting for multiple comparisons) with each of the extracted features. We were able to identify distributed patterns of neural activity that were statistically significantly associated with all but one of the extracted features (Figure 3). While interpretation of these results is of little importance here, we note in passing that a number of the patterns exhibited in these statistical maps replicated the neural correlates previously identified in more conventional factorial experimental designs. For example, in the auditory domain, speech was associated with activity in areas established language processing regions, such as the superior frontal gyrus (SFG) [14], and in the visual domain, distinct image tags such as "outdoors", "street" and "light", were associated with differential activation in brain regions important for natural scene recognition, such as visual and retrosplenial cortices [19].

In the present case study, we demonstrated how automated feature extraction can be used to rapidly test novel hypothesis in naturalistic fMRI experiments. Pliers allows researchers to quickly and flexibly test a potentially large number of neuroscientifically interesting hypotheses using pre-existing fMRI datasets. Furthermore, the rapid and automated nature allows researchers to quickly generate results and more easily replicate their analyses in new datasets, increasing the generalizability of their findings. Although more work is necessary to optimize statistical modeling in naturalistic datasets—in part due to concerns of collinearity between extracted features—the present approach enables a new class of neuroimaging study that promises to maximize ecological validity and encourage re-use of openly accessible datasets. Of course, this example highlights just one among many potential uses for our feature extraction framework.

5 LIMITATIONS

In its present state, pliers is already fully functional and suitable for use in exploratory or quick-start feature extraction pipelines across a range of applied data science and research settings. Nevertheless, pliers remains under active development, and currently has a number of limitations worth noting. First, while pliers already supports a number of state-of-the-art, widely used feature extraction tools and services, there are hundreds, if not thousands, of others that are currently unsupported but could be readily added to the package. The utility of pliers should continue to increase as we and others continue to add new transformers to the package.

Second, our emphasis on ease-of-use and a high degree of abstraction reduces development time, but comes at the cost of decreased performance. At present, Pliers supports basic caching of transformer outputs in order to prevent repeated computationally (or, in the case of some APIs, financially) expensive transformations, has basic parallelization across iterable inputs, and adopts lazy evaluation patterns whenever possible. However, it otherwise makes no explicit effort to optimize memory consumption or CPU usage. Moreover, scalability is inherently limited by the performance of the individual tools and services being invoked. In view of these considerations, Pliers is at present unlikely to be suitable for production use in Big Data environments; rather, the target use case is small-to-medium data science pipelines where data exploration and/or rapid prototyping are the primary goals and scalability is not of great concern.

Third, Pliers does not provide fine-grained control over extraction parameters for many of the underlying tools and algorithms. For example, the `OpticalFlowExtractor` (which quantifies the amount of motion between consecutive movie frames) offers no control over the parameters of the Farneback algorithm it relies on. Instead, pliers places emphasis on easy specification of those parameters most likely to vary across common use cases. For example, the `ClarifaiAPIExtractor` allows the user to specify (upon initialization) the Clarifai model to query, as well as any specific classes or labels to retrieve (e.g., to quantify the likelihood that an "animal" is present in each image).

Lastly, an important potential challenge for pliers is the continued maintenance of the package in light of not-infrequent version upgrades to the various tools and services that pliers supports. We expect that the third-party APIs pliers leverages for most of its extractors will improve over time, which means that some extractor code will occasionally need to be reconfigured to account for external API changes. To combat this, we intend to diligently keep up-to-date with the supported APIs and services. Maintenance of the pliers codebase is facilitated by the use of a continuous integration testing service (Travis-CI) that notify the developers whenever a previously functional transformer begins to fail.

6 CONCLUSION

We have outlined the key properties and major benefits of a new framework for high-level, standardized, multimodal feature extraction. Pliers enables users to easily and quickly extract a variety of image, audio, and text features using a standardized, highly extensible, and relatively simple interface, providing a powerful tool for rapid prototyping and data exploration when developing data science pipelines.

7 ACKNOWLEDGMENTS

This work was supported by a National Institutes of Health award to TY (NIH award R01MH109682).

REFERENCES

[1] D. Bogdanov, N. Wack, E. Gomez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, X. Serra, and et al. Essentia: an audio analysis library for music information retrieval. *International Society for Music Information Retrieval Conference*, page 493-498, 2013.

[2] M. Brysbaert and B. New. Moving beyond kuĂŀera and francis: A critical evaluation of current word frequency norms and the introduction of a new and improved word frequency measure for american english. *Behavior Research Methods*, 41(4):977-990, 2009.

[3] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, and A. Mueller. Api design for machine learning software: experiences from the scikit-learn project. *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases*, Sep 2013.

[4] M. F. Glasser, S. N. Sotiropoulos, J. A. Wilson, T. S. Coalson, B. Fischl, J. L. Andersson, J. Xu, S. Jbabdi, M. Webster, J. R. Polimeni, and et al. The minimal preprocessing pipelines for the human connectome project. *NeuroImage*, 80:105-124, 2013.

[5] K. Gorgolewski, C. D. Burns, C. Madison, D. Clark, Y. O. Halchenko, M. L. Waskom, and S. S. Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Front Neuroinform*, 5, 08 2011.

[6] R. Hyam. Automated image sampling and classification can be used to explore perceived naturalness of urban spaces. *Plos One*, 12(1), Apr 2017.

[7] N. Kanwisher and G. Yovel. The fusiform face area: a cortical region specialized for the perception of faces. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 361(1476):2109-2128, 2006.

[8] A. Khosla, W. A. Bainbridge, A. Torralba, and A. Oliva. Modifying the memorability of face photographs. *2013 IEEE International Conference on Computer Vision*, 2013.

[9] D. Kiela. Mmfeat: A toolkit for extracting multi-modal features. *Proceedings of ACL-2016 System Demonstrations*, 2016.

[10] V. Kuperman, H. Stadthagen-Gonzalez, and M. Brysbaert. Age-of-acquisition ratings for 30,000 english words. *Behavior Research Methods*, 44(4):978-990, Dec 2012.

[11] O. Lartillot, P. Toiviainen, and T. Eerola. A matlab toolbox for music information retrieval. *Data Analysis, Machine Learning and Applications Studies in Classification, Data Analysis, and Knowledge Organization*, page 261-268, 2008.

[12] M. Mazloom, R. Rietveld, S. Rudinac, M. Worring, and W. V. Dolen. Multimodal popularity prediction of brand-related social media posts. *Proceedings of the 2016 ACM on Multimedia Conference - MM '16*, 2016.

[13] J. C. Rangel, M. Cazorla, I. GarcÃŋa-Varea, J. MartÃŋnez-GÃşmez, Ã. Fromont, and M. Sebban. Computing image descriptors from annotations acquired from external tools. *Advances in Intelligent Systems and Computing Robot 2015: Second Iberian Robotics Conference*, page 673-683, 2015.

[14] S. K. Scott and I. S. Johnsrude. The neuroanatomical and functional organization of speech perception. *Trends in Neurosciences*, 26(2):100-107, 2003.

[15] S. M. Smith, M. Jenkinson, M. W. Woolrich, C. F. Beckmann, T. E. Behrens, H. Johansen-Berg, P. R. Bannister, M. D. Luca, I. Drobnjak, D. E. Flitney, and et al. Advances in functional and structural mr image analysis and implementation as fsl. *NeuroImage*, 23, 2004.

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[18] D. C. Van Essen, S. M. Smith, D. M. Barch, T. E. Behrens, E. Yacoub, and K. Ugurbil. The wu-minn human connectome project: An overview. *NeuroImage*, 80:62-79, 2013.

[19] D. B. Walther, E. Caddigan, L. Fei-Fei, and D. M. Beck. Natural scene categories revealed in distributed patterns of activity in the human brain. *Journal of Neuroscience*, 29(34):10573-10581, 2009.

[20] A. B. Warriner, V. Kuperman, and M. Brysbaert. Norms of valence, arousal, and dominance for 13,915 english lemmas. *Behavior Research Methods*, 45(4):1191-1207, 2013.

[21] A. Zhang. Speech recognition, 2017.