ConvoKit: A Toolkit for the Analysis of Conversations

Jonathan P. Chang
Cornell University
jpc362@cornell.edu

Caleb Chiam
Cornell University
cc982@cornell.edu

Liye Fu
Cornell University
liye@cs.cornell.edu

Andrew Z. Wang
Stanford University
anwang@cs.stanford.edu

Justine Zhang
Cornell University
jz727@cornell.edu

Cristian Danescu-Niculescu-Mizil
Cornell University
cristian@cs.cornell.edu

Abstract

This paper describes the design and functionality of ConvoKit, an open-source toolkit for analyzing conversations and the social interactions embedded within. ConvoKit provides an unified framework for representing and manipulating conversational data, as well as a large and diverse collection of conversational datasets. By providing an intuitive interface for exploring and interacting with conversational data, this toolkit lowers the technical barriers for the broad adoption of computational methods for conversational analysis.

1 Introduction

The NLP community has benefited greatly from the public availability of standard toolkits, such as NLTK (Bird et al., 2009), StanfordNLP (Qi et al., 2018), spaCy (Honnibal and Montani, 2020), or scikit-learn (Pedregosa et al., 2011). These toolkits allow researchers to focus on developing new methods rather than on re-implementing existing ones, and encourage reproducibility. Furthermore, by lowering the technical entry level, they facilitate the export of NLP techniques to other fields.

Although much of natural language is produced in the context of conversations, none of the existing public NLP toolkits are specifically targeted at the analysis of conversational data. In this paper, we introduce ConvoKit (https://convokit.cornell.edu), a Python package that provides a unified open-source framework for computationally analyzing conversations and the social interactions taking place within, as well as a large collection of conversational data in a compatible format.

In designing a toolkit for analyzing conversations, we start from some basic guiding principles. Firstly, conversations are more than mere 'bags of utterances', so we must capture what connects utterances into meaningful interactions. This translates into native support of reply and tree structure as well as other dependencies across utterances.

Secondly, conversations are inherently social. People often engage in multiple conversations, and how we understand interactions is contingent on what we know about the respective interlocutors. Similarly, the way we understand each speaker is contingent on their entire conversational history. Thus, a conversational analysis toolkit must allow for the integration of speaker information and behaviors across different conversations.

Thirdly, conversations occur in vastly different contexts, from dyadic face-to-face interactions, to discussions and debates in institutional settings, to online group discussions, and to large-scale threaded discussions on social media. This means that the toolkit must offer a level of abstraction that supports different interaction formats.

Finally, since conversational data is key to many social science fields (e.g. political science, sociology, social psychology), the framework should be accessible to a broad audience: not only experienced NLP researchers, but anyone with questions about conversations who may not necessarily have a high degree of NLP expertise.

In this paper, we describe how these principles guided our design of ConvoKit's framework architecture (Section 2), describe some of the analysis methods (Section 3) and datasets (Section 4) included in ConvoKit, and conclude with some high-level remarks on future developments (Section 5).

2 Framework Architecture

The current state of the software and data ecosystem for conversational research is fragmented: popular conversational datasets are each distributed in different data formats, each using their own task-specific schemas, while similarly, code for reproducing various conversational methods

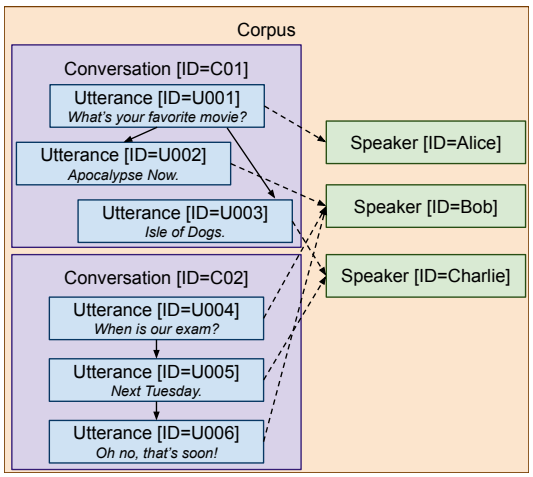

Figure 1: Visualization of the relationship between the four core classes of the `Corpus` hierarchy. Solid arrows denote reply-to relationships between `Utterances`, while dashed arrows denote attribution of each `Utterance` to its authoring `Speaker`.

tends to be ad-hoc with no guarantee of overlapping functionality or cross-compatibility. This combination of factors poses a barrier to both reproducibility and broader adoption.

To address these issues, a *unified* framework for analyzing conversations must provide both a standardized format for *representing* any conversational data, and a general language for describing *manipulations* of said data. Furthermore, as described in Section 1, the representation must go beyond a mere "bag-of-utterances" and natively capture the structure of conversations, while the language of manipulations must be expressive enough to describe actions at different levels of the conversation: individual utterances, entire conversations, speakers in and across conversations, and arbitrary combinations of the above.

These criteria directly lead to the two core abstractions underlying ConvoKit: the `Corpus`, representing a collection of one or more conversations, and the `Transformer`, representing some action or computation that can be done to a `Corpus`. To draw an analogy to language, `Corpus` objects are the nouns of ConvoKit, while `Transformers` are the verbs.

Representing conversational data. The main data structure for organizing conversational data in ConvoKit is the `Corpus`, which forms the top of a hierarchy of classes representing different levels of a conversation (Figure 1): A `Corpus` is a collection of `Conversations`, each `Conversation` is made up of one or more `Utterances`, and each `Utterance` is attributed to exactly one `Speaker` (but each `Speaker` can own multiple `Utterances`). `Conversations`, `Utterances` and `Speakers` are identified by unique IDs. Conversation structure is represented by the `reply_to` field of the `Utterance` class, which specifies the ID of the other `Utterance` it replies to (i.e., its parent node in the conversation tree). ConvoKit leverages the relationships between `Utterances`, `Speakers`, and `Conversations` to provide rich navigation of a `Corpus`, such as tree traversal of `Utterances` within a `Conversation` or chronological iteration over all of a `Speaker`'s `Utterance` history.

Custom metadata. Objects in the `Corpus` hierarchy contain some basic information that is generally useful for most operations on conversational data, such as the text content and timestamp of each `Utterance`. However, any use of ConvoKit beyond basic analyses will likely require additional task-specific information. This is supported by ConvoKit in the form of *metadata*. Each of the four classes in the hierarchy contains a field called `meta`, which is a lookup table that may be used to store additional information about the `Corpus`, `Conversation`, `Utterance`, or `Speaker` under some descriptive name. In practice, metadata ranges in complexity from speaker ages to sub-utterance level DAMSL speech act tags.

Manipulating conversational data. ConvoKit supports conversational analyses centered on any level of the hierarchy; for instance, one may wish to examine linguistic characteristics of `Utterances`, characterize a `Conversation` in terms of the structure of its `Utterances`, or track a `Speaker`'s behavior across the `Conversations` they have taken part in throughout their lifetime.

Such flexibility in analysis is achieved by abstracting manipulations of conversational data through the `Transformer` class. At a high level, a `Transformer` is an object that takes in a `Corpus` and returns the same `Corpus` with some modifications applied. In almost all cases, these modifications will take the form of changed or added metadata. For example, the `PolitenessStrategies Transformer` annotates every `Utterance` with a feature vector that counts the presence of politeness features from Danescu-Niculescu-Mizil et al. (2013), while `UserConvoDiversity` annotates every `Speaker` with a measure of their linguistic diversity across the whole `Corpus`.

The key to ConvoKit's flexibility is that, while a Transformer can represent any arbitrary manipulation of a Corpus and operate at any level of abstraction, all Transformer objects share the same syntax—that is, the Transformer class API represents a general language for specifying actions to be taken on a Corpus. This interface is directly modeled after the scikit-learn class of the same name: a Transformer provides a fit() function and a transform() function. fit() is used to prepare/train the Transformer with any information it needs beforehand; for example, a Transformer that computes bag-of-words representations of Utterances would first need to build a vocabulary. transform() then performs the actual modification of the Corpus.

In addition to these standard functions, Transformers also provide a summarize() helper function that offers a high-level tabular or graphical representation of what the Transformer has computed. For example, PolitenessStrategies offers a summarize() implementation that plots the average occurrence of each politeness feature. This can be helpful for getting a quick sense of what the Transformer does, for simple exploratory analyses of a Corpus, or for debugging.

A single Transformer on its own might not make significant changes, but because Transformers return the modified Corpus, multiple Transformers can be chained together, each one taking advantage of the previous one's output to produce increasingly complex results (see Figure 2 for an example).

```
1  corp = Corpus(filename=download(
2    'movie-corpus'))
3
4  # Preprocessing step
5  tc = TextCleaner()
6  tc.transform(corp)
7
8  # Constructing new metadata
9  for c in corp.iter_conversations():
10   genders = [s.meta['gender'] for s
     ↪ in c.iter_speakers()]
11   convo.meta['mixed'] = 'M' in
     ↪ genders and 'F' in genders
12
13 # Analysis step
14 fw = FightingWords()
15 fw.fit(corp,
16   class1_func=lambda utt: utt.
     ↪ get_conversation().meta['mixed'],
17   class2_func=lambda utt: not utt.
     ↪ get_conversation().meta['mixed'])
18 fw.summarize(corp)
```

Figure 2: Basic example code demonstrating how combining different Transformers, and leveraging the Corpus hierarchy's rich navigation features and metadata functionality, can be used to study conversational data—in this example, comparing the language used in mixed-gender and single-gender movie dialogs.

3 Transformers

In this section, we introduce some of the built-in Transformers that are available for general use. Broadly speaking, we can group the functionality of Transformers into three categories: preprocessing, feature extraction, and analysis.

Preprocessing refers to the preliminary processing of the Corpus objects prior to some substantive analysis. For example, at the Utterance-level, preprocessing steps can include converting dirty web text into a cleaned ASCII representation (implemented in TextCleaner) or running a dependency parse (implemented in TextParser). At the Conversation-level, preprocessing steps might include merging consecutive utterances by the same speaker, while at the Speaker-level, they might include merging contributions from speakers with multiple user accounts.

Feature extraction refers to transformation of conversational data, such as utterance text or conversational structure, into (numerical) features for further analysis and applications. An example of an Utterance-level feature extractor is the previously described PolitenessStrategies, while an example of a Conversation-level feature extractor is HyperConvo, which constructs a hypergraph representation of the Conversation and extracts features such as (generalized) reciprocity, indegree and outdegree distributions, etc.

Analysis is the process of combining Utterance, Conversation and Speaker features and metadata into a statistical or machine learning model to achieve a higher-level understanding of the Corpus. For example, FightingWords implements Monroe et al. (2008)'s method for prinicpled comparison of language used by two subsets of a Corpus; Classifier acts as a wrapper around any scikit-learn machine learning model and can be used to classify Utterances, Conversations, or Speakers based on the output of feature extraction Transformers; and Forecaster

implements Chang and Danescu-Niculescu-Mizil (2019)'s method for modeling the future trajectory of a `Conversation`.

Figure 2 illustrates how `Transformers` belonging to each category can be combined in sequence to perform a practical conversational task: comparing the language used in movie dialogs containing characters of different genders to that used in dialogs containing only one gender.[1]

4 Datasets

ConvoKit ships with a diverse collection of datasets already formatted as `Corpus` objects and ready for use 'out-of-the-box'. These datasets cover the wide range of settings conversational data can come from, including face-to-face institutional interactions (e.g., supreme court transcripts), collaborative online conversations (e.g., Wikipedia talk pages), threaded social media discussions (e.g., a full dump of Reddit), and even fictional exchanges (e.g., movie dialogs).[2]

The diversity of these datasets further demonstrates the expressiveness of our choice of conversation representation. We also provide guidelines and code for transforming other datasets into ConvoKit format, allowing ConvoKit's reach to extend beyond what data is already offered.

5 Conclusions and Future Work

In this paper, we presented ConvoKit, a toolkit that aims to make analysis of conversations accessible to a broad audience. It achieves this by providing intuitive and user friendly abstractions for both representation and manipulation of conversational data, thus promoting reproducibility and adoption.

ConvoKit is actively being developed. While it is currently heavily centered around text analysis (with other modalities being only indirectly supported as metadata), providing first-class support for spoken dialogs is considered as an important line for future extension. In addition, we aim to continue to incorporate new datasets, analysis methods, and integrate with other parts of the NLP software ecosystem that could benefit from ConvoKit's abstractions, including dialog generation toolkits such as ParlAI (Miller et al., 2018).

ConvoKit is an open-source project and we welcome contributions of any kind, ranging from bugfixes and documentation, to augmenting existing corpora with additional useful metadata, to entirely new datasets and analysis methods.[3]

Acknowledgments

We thank the anonymous reviewers for their thoughtful comments and are grateful to all ConvoKit contributors. This work was supported by an NSF CAREER award IIS-1750615. Zhang was supported in part by a Microsoft PhD Fellowship.

References

Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Jonathan P. Chang and Cristian Danescu-Niculescu-Mizil. 2019. Trouble on the Horizon: Forecasting the Derailment of Online Conversations as they Develop. In *Proceedings of EMNLP*.

Cristian Danescu-Niculescu-Mizil, Moritz Sudhof, Dan Jurafsky, Jure Leskovec, and Christopher Potts. 2013. A Computational Approach to Politeness with Application to Social Factors. In *Proceedings of ACL*.

Matthew Honnibal and Ines Montani. 2020. spaCy: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing.

Alexander H. Miller, Will Feng, Adam Fisch, Jiasen Lu, Dhruv Batra, Antoine Bordes, Devi Parikh, and Jason Weston. 2018. ParlAI: A Dialog Research Software Platform. *arXiv:1705.06476 [cs]*.

Burt L. Monroe, Michael P. Colaresi, and Kevin M. Quinn. 2008. Fightin' Words: Lexical Feature Selection and Evaluation for Identifying the Content of Political Conflict. *Political Analysis*, 16(04).

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal Dependency Parsing from Scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

[1] This example, together with its output and other examples, can be found at https://convokit.cornell.edu/documentation/examples.html.

[2] A complete list of datasets can be found at https://convokit.cornell.edu/documentation/datasets.html.

[3] See contribution guidelines on the ConvoKit webpage.