UTEP PROSODY-BASED PROPERTY-INFERENCE PACKAGE (PROSPROP) API

Version 0.9, June 2017

Nigel G. Ward, Jason C. Carlson, Ivan Gris, University of Texas at El Paso
with support from Darpa's Lorelei Program, and testing by Anindita Nath

Features not yet complete are in a tiny font, like this.

PURPOSE

Given an audio file as input, this package returns a set of property-value pairs that characterize it, for example "urgent: 1.7", "medical-assistance: 0.2, locally-relevant 1.3".

The ultimate goal is to support downstream, user-facing applications. Short-term use scenarios include:

1. Stand-alone stance inference by government users, either using this code itself, or using it as a template for reimplementation in some other language. The value of stance inference and performance information is given in our papers and reports, such as Inferring Stance from Prosody, available at the stance website: http://www.cs.utep.edu/nigel/stance/ .
2. The LoreHLT evaluation to be run in August 2017, as a stand-alone system. In this case the output properties will be the 11 "situation frame types".
3. Future LoreHLT evaluations as a component of larger systems. The most direct use will be for estimating relevance of a new news segment to a disaster, or to some situation-type relating to that disaster. In future years the output may be combined with other techniques to infer more fine-grained information.
4. Research in how to improve stance prediction, and better model prosody
5. Research in how to improve situation type prediction, and other tasks.

LOCATION

Code: https://github.com/nigelgward/stance/.

Descriptions, know-how, ppm files: http://www.cs.utep.edu/nigel/stance/

INTERNALS

Currently this works by comparing the prosody in the input file, computed over 6-second patches, with the prosody in the annotated reference data, as described in "Inferring Stance in News Broadcasts from Prosodic-Feature Configurations", Nigel G. Ward, Jason C. Carlson, Olac Fuentes. *Computer Speech and Language*, submitted.

USAGE

prosprop() infers from the input audio, via the prosody, the values of the various properties

Inputs:

- audioDir: The directory containing the audio file(s) that contain the segments for which to infer the properties. The audio should be at the top level in this directory. Each file must be in .au format: mono, 16 bit, 8000 or 16000 Hz, as described in the Midlevel documentation. For conversion hints, see the comments in getSegInfo().
- segInfoDir: The directory containing information on segment starts and ends within each audio file. There are a several of possible formats; see the comments in getSegInfo().
- ppmfile The name of a prosody-property mappings (ppm) file (.mat format). Each such file encodes the prosody-to-property mappings derived from a set of annotated reference data. At present these are a shallow encoding, and as such these files are very large. The ppm files currently available are:
    - eng-mono4-testeng-ppm.mat
    - eng-stance-lnews.mat (English, stance, local news)
    - cmn-stance-kazn.mat (Mandarin, stance, KAZN local news)
    - tur-stance-voacri.mat (Turkish, stance, VOA and CRI Bolt data, LDC2014E115)
    - uig-stance-vnews.mat (Uyghur, stance, Lorelei data: various news)
    - spn-stype-vnews.mat (Spanish, situation types, Lorelei data: various news)
    - rus-stype-vnews.mat (Russian, situation types, Lorelei data: various news)
- Stride. How often to compute patches, in milliseconds. 100 or 200 give good performance with reasonable speed.
- Flags. This is a string containing one or more of the following letters:
    - j -- write a json file
    - u -- output user-facing stances instead of raw stances

Output

A matrix of property values, with one row per input audio file and one column for each property.
In addition, a json-format file will be written to the outputs directory if the 'j' flag is specified. This file is in the format used by the USC evaluation scripts.

Since this is a general-purpose package, the set of properties can be anything that has been previously annotated. Thus, depending on the ppm file used, the package can return values for different sets of properties. The range of output values depends on the property set, and will have the same range used by the annotators of the reference data; thus downstream applications may need to rescale or discretize these for their own purposes. Currently raw stance properties are in the range 0-2 and situation-type properties in the range 0-1.

The code can be involved in various ways:

- In Matlab, by directly calling the prosprop function.
- From Python, by calling prosprop.py, which is a wrapper for the Matlab code
- Through Swagger, once we get it working. This will also enable use as a web service

Sample calls for Matlab and Python are given below.

INSTALLATION

You will need three things: this package, the Midlevel Prosodic Features toolkit, and the pitch computation and related functions from Mike Brookes' Voicebox.

- This code: http://www.cs.utep.edu/nigel/stance/

- Midlevel Toolkit
  - Documentation: https://github.com/nigelgward/midlevel/blob/master/doc/mlv7.pdf
  - code: https://www.github.com/nigelgward/midlevel/
- Voicebox http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html .

Test procedure 1:

- Run Matlab
- addpath *location*/stance/src
- addpath *location*/midlevel/src
- addpath *location*/voicebox
- cd *location*/stance/testeng
- regressionTest();

Test procedure 2:

- Install Matlab/Python Engine: https://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html
- Add the environment variable to the path PythonPath:
  C:\Python34\Lib;C:\Python34\DLLs;C:\Python34\Lib\lib-tk; Path: C:\Python34
  http://stackoverflow.com/questions/3701646/how-to-add-to-the-pythonpath-in-windows-7
- From a console, go to the stance directory and use the following command:

```
python prosprop audio.au ppmfile
```

PROPERTIES

The various property sets and their meanings are defined elsewhere; for convenience they are summarized here in the table.

(The 14 raw stances are described in Inferring Stance from Prosody and in Preliminaries to a Study of Stance in News Broadcasts. The 8 user-facing stance aspects described on page 16 of the Stance Usability Packet. All are available from http://www.cs. utep.edu/nigel/stance/ . The situation types are defined in Lorelei Situation Frame Annotation Guildelines for Speech Data, v 2.6, by Appen.)

Raw Stances	**User-Facing Stances**	**Situation Types**
1. Bad 2. Good 3. Deplorable 4. Praiseworthy 5. Controversial 6. Factual Information 7. Subjective Information 8. Unusual or Surprising 9. Typical or Unsurprising 10. Local 11. Prompting Immediate Action 12. Background 13. New Information	1. Bad … Good 2. Deplorable …. Praiseworthy 3. Subjective … Factual 4. Typical … Unusual 5. Distant … Local 6. Just Talk … Urgent 7. Old Information … New Information 8. Idiosyncratic … Relevant to a Large Group	1. Civil unrest or widespread crime 2. Elections and politics 3. Evacuation 4. Food supply 5. Urgent rescue 6. Utilities, energy, or sanitation 7. Infrastructure 8. Medical assistance 9. Shelter 10. Terrorism or extreme violence 11. Water supply

14. Relevant to a Large Group		

In future, properties might include

- Status Variables, as described in NIST LoReHLT 2017 Evaluation Plan,which will probably map to 5 scalar properties: past-need, current-need, future-need, relief-already-sufficient, and urgency.

- Emotion Properties, as described in the SEC Pilot Plans document by Kathleen McKeown. These will probably map to 4 scalar properties: positive, negative, anger, fear.

SYSTEM REQUIREMENTS

This system has been tested with Matlab R2016a. It should also work with Matlab R2014a or greater.

The python wrapper was tested on Python 3.4; it should also work with Python 3.2 or 3.3, but no higher than 3.4.

DIRECTORY STRUCTURE

```
doc
        Documentation, including this file
src
        Source code, in matlab, including:
        -prosprop.m top-level function to infer properties from prosody
        - regressionTest.m a small-scale test
        -prosprop.py a wrapper to call the above from python
        -makePPM.m, a function to create a prosody-property mapping file
        -mono4.fss, a feature-set-specification file
        -README.txt, more information on the functions present here
ppmfiles
        Models of the prosody-properties mapping, for various languages, data sets, and
        annotations
aufiles
        input audio files to be classified
swagger
        # Swagger server and .yaml file for the API provided for reference.
        # You may want to regenerate these files using the .yaml to ensure
        # proper paths and server generation.
outputs
        where the mat file (and sometimes also a json file) for outputs are written
extensions
        extra code for experiments etc. outside the main workflow.
```

CREATING PPM FILES

In general, UTEP intends to release new prosody-property mapping (ppm) files as needed, for example for a new Lorelei incident language, at the stance website. If you wish to do this yourself, use makePPM.m. This handles a couple of annotation formats and a couple of directory structures, but if your data is organized differently, you'll need to make adjustments.

A ppm file is just a mat file with the following variables:

- **provenance**: a string describing the source of the mappings. This will specify at least the language, the annotation type (stance or situation types), the feature set, and the date.
- **propertyNames**: a cell array of strings
- **featureSpec**. The name of a feature-set specification (.fss) file, such as mono4.fss. This specifies what features to compute for the input file; these will be the same as those used to create the ppm file. In addition to what is specified, two temporal features are always appended.
- **means**. A vector of means for each of the features, computed over the reference set. This is uses to normalize the input-file features in the same way.
- **stddevs**. A vector of standard deviations, similarly.
- **algorithm**. A string that specifies which algorithm to use; currently only kNN is available.
- **model**. A model based on the training data, suitable for use by 'algorithm'. For kNN, the model is of the form 'segmentData', which is an array of structs, one per segment, where each struct includes
    - **features** the prosodic-features for each patch of this segment. The rows are patches (one every 10oms throughout the data) and the columns features, as described by featurespec.
    - **properties** a vector of property values, for example 14 stance values
    - **broadcastName** for debug: the audio file or directory this segment belongs to
    - **startTime** for debug: start time in seconds of this segment within broadcast
    - **endTime** for debug: end time in seconds of this segment within broadcast

INPUT-FORMAT NOTES

Universally our task involves news segments within broadcasts, but there are two formats for this

- UTEP has each broadcast in a file, with segment start/end times marked by the annotators, and thus in csv files. Regarding their format, see the comments in readStanceSpreadsheet.m
- LDC/Appen delivers each segment as a file, with broadcast provenance indicated by the directory structure and filenames. Fortunately the segment start/end times, relative to the broadcast, were also marked by the stance annotators. This is not true for the situation-frame annotations, so we may need to do some extra work there.

In the past for LDC/Appen data, we've processed each segment file individually, but this loses information. Specifically the normalization parameters get computed over too-short segments, rather than entire broadcasts. Also, less critically, many prosodic features are mostly invalid near segment starts and ends. So we need to concatenate the files to recreate the broadcasts. Then, provided the segments start ends are available, and they are (see above), everything goes smoothly: it processes/normalizes per broadcast, then pulls out the features for each news segment, using featuresForSegment().

To prepare for this takes a preliminary step. Since a typical LDC/Appen path is

IL3_EVAL_AUDIO/001/AUDIO/IL3_EVAL_001_001.flac

This requires us to concatenate using, e.g.:

sox 001/AUDIO/*flac -r 8000 concat001.au

and so on for 002 etc. A convenient way to do this is with ls > shellscript then editing to add the sox commands, then running it. That is, this conversion is best done outside matlab.

MATLAB-PYTHON INTEGRATION NOTES

How to call Matlab functions from Python: https://www.mathworks.com/help/matlab/matlab_external/call-matlab-functions-from-python.ht

How to call user scripts on Matlab from Python: https://www.mathworks.com/help/matlab/matlab_external/call-user-script-and-function-from-python.html

Some additional configuration may be required to access Matlab from Python in your server. You can find the steps at https://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html

HISTORY

This is a rewrite of the February 2017 release. The major changes are:

- a simple interface, with a single top-level function to call
- better documentation and comments
- sample data and outputs (see comments in the code)
- support for handling LDC-Appen style corpora, with individual audio files for each segment
- support for both stance and situation-frame types
- addition of temporal features, which improve performance