parserator Documentation

Release 0.4.1

Cathy Deng, Forest Gregg

Mar 29, 2018

Contents

Contents:

Parserator is a toolkit for making domain-specific probabilistic parsers, built on python-crfsuite. To create a parser, all you need is some training data to teach your parser about its domain.

Parserator will help you:

- initialize a parser module
- create labeled training data from unlabeled strings
- train your parser model on labeled training data

CHAPTER 1

What a probabilistic parser does

A probabilistic parser makes informed guesses about the structure of messy, unstructured text. Given a string, a parser will break it out into labeled components.

Parserator creates parsers that use conditional random fields to label components based on (1) features of the component string and (2) the order of labels. A probabilistic parser learns about features and labels from labeled training data.

CHAPTER 2

Use cases for a probabilistic parser

A probabilistic parser is particularly useful for sets of strings that have common structure/patterns, but can deviate from those patterns in ways that are difficult to anticipate with hard coded rules.

For example, in most cases, US addresses start with street number. But there are exceptions: sometimes valid addresses deviate from this pattern (e.g. addresses starting with building name, PO box) and furthermore, addresses in real datasets often include typos & errors. Because there are infinitely many patterns and possible typos to account for, a probabilistic parser is well-suited to parse US addresses.

A neat thing about a probabilistic approach (as opposed to a rule-based approach) is that the parser can continually learn from new training data, and continually improve its performance.

Some examples of existing parsers that use parserator:

- usaddress - Our first probabilistic parser and the basis for the parserator toolkit, it parses any address in the United States. Read our blog post on how it works.
- probablepeople - A parser for romanized person names.

Try out these parsers on our web interface!

Examples of other domains where a probabilistic parser can be useful:

- addresses in another country
- product names/descriptions (e.g. parsing 'Twizzlers Twists, Strawberry, 16-Ounce Bags (Pack of 6)' into brand, item, flavor, weight, etc)
- citations

CHAPTER 3

Installation

```
pip install parserator
```

CHAPTER 4

How to make a parser using parserator

1. **Initialize a new parser**

   ```
   parserator init [YOUR PARSER NAME]
   python setup.py develop
   ```

   The `init` command will initialize a new parser in your current directory. For example, running `parserator init foo` will generate the following directories and files:

   ```
   foo/
   foo/__init__.py
   foo_data/
   foo_data/labeled_xml/
   foo_data/unlabeled/
   setup.py
   tests/
   tests/test_tokenizing.py
   ```

2. **Configure the parser to your domain**

   - **configure labels**
     - The labels (i.e. the set of possible tags for the tokens) are defined by `LABELS` in `__init__.py`
   - **configure tokenizer**
     - The tokenize function in `__init__.py` determines how any given string will be split into a sequence of tokens to be tagged, via a regex pattern object. In defining the regex pattern, it's helpful to consider what characters should split tokens (e.g. should the string 'foo;bar' be one token or two) and if so, how characters should be captured in tokens (e.g. should 'foo;bar' be split into ['foo;', 'bar'] or ['foo', ';bar']
     - In the tests repo, there is a test file for testing the performance of the tokenize function. You can adapt it to your needs & run the test w/ `nosetests .` to ensure that you are splitting strings properly.

- **optional: additional config**
  - PARENT_LABEL and GROUP_LABEL are training data XML tags (see #4 for more info on the training data format). For example, the name parser has PARENT_LABEL = 'Name' & GROUP_LABEL = 'NameCollection'

3. **Define features relevant to your domain**
   - In __init__.py, features are defined in the tokens2features and tokenFeatures functions. Given an individual token, tokenFeatures should return features of that token - for example, a length feature and a word shape (casing) feature.
   - Given a sequence of tokens, tokens2features should return all features for the tokens in the sequence, including positional features - for example, the features of previous/next tokens, and features for tokens that start/end a string.
   - For examples of features in other domains, see features for names and features for U.S. addresses.
4. **Prepare training data**
   - Parserator reads training data in the following XML form, where token text is wrapped in tags representing the correct label, and sequences of tokens are wrapped in a parent label (specified by PARENT_LABEL in __init__.py):

```
<Collection>
    <TokenSequence><label>token</label> <label>token</label> <label>token</
↪label></TokenSequence>
    <TokenSequence><label>token</label> <label>token</label></TokenSequence>
    <TokenSequence><label>token</label> <label>token</label> <label>token</
↪label></TokenSequence>
</Collection>
```

   - If you have labeled strings in other formats, they will need to be converted to this XML format for parserator to read the data. In data_prep_utils.py, there are some tools that can help you do this. For example, the sequence2XML function reads labeled sequences represented as a list of tuples and returns the analogous XML represention: [(token, label), (token, label), ...] -> <TokenSequence><label>token</label> <label>token</label> ... </TokenSequence>
   - If you only have raw, unlabeled strings, parserator can help you manually label tokens through a command line interface. To start a manual labeling task, run parserator label [infile] [outfile] [modulename]
     - The infile option should be the filepath for a csv, where each line is a string
     - The outfile option should be the filepath for an xml file, where manually labeled training data will be written. If you specify an existing xml file as the outfile, the newly labeled strings will be appended at the end of the xml file.
     - When you exit a manual labeling task, any strings from the infile that were not labeled will be written to a separate csv, with unlabeled_ prepended to the filename, so that in the future, you can pick up where you left off.
     - Within the manual labeling task, you will be prompted with tokens to label. To label tokens, enter the number corresponding to the correct tag. To see a mapping of numbers to labels, type 'help'
     - If the parser model (learned_settings.crfsuite by default) already exists, the console labeler will use it to inform the manual labeling task.
5. **Train your parser**

- To train your parser on your labeled training data, run `parserator train [traindata] [modulename]`
- **The traindata argument can be the path to a single file, a list of filepaths separated by a comma (no space), or a g**

  ```
  parserator train mytraindata/labeled.xml usaddress
  parserator train file1.xml,file2.xml,file3.xml usaddress
  parserator train "my/training/data/*.xml" usaddress
  ```

- After training, your parser will have an updated model, in the form of a .crfsuite settings file (learned_settings.crfsuite by default)
- If there was an existing .crfsuite settings file, it will be renamed with the model creation timestamp appended
- Once the settings file exists, the parse and tag methods use it to label tokens in new strings

6. **Repeat steps 3-5 as needed!**

CHAPTER 5

Important links

- Documentation: http://parserator.rtfd.org/
- Web interface for trying out parsers: http://parserator.datamade.us/
- Blog post: http://datamade.us/blog/parse-name-or-parse-anything-really/
- Repository: https://github.com/datamade/parserator
- Issues: https://github.com/datamade/parserator/issues
- Distribution: https://pypi.python.org/pypi/parserator

CHAPTER 6

Indices and tables

- genindex
- modindex
- search