Formasaurus Documentation

Release 0.5

Mikhail Korobov

March 03, 2016

Contents

Formasaurus is a Python package that tells you the type of an HTML form and its fields using machine learning.

It can detect if a form is a login, search, registration, password recovery, "join mailing list", contact, order form or something else, which field is a password field and which is a search query, etc.

License is MIT.

CHAPTER 1

Install

Formasaurus requires Python 2.7+ or 3.3+ and the following Python packages:

- scipy
- numpy
- scikit-learn 0.17+
- sklearn-crfsuite
- lxml

First, make sure numpy is installed. Then, to install Formasaurus with all its other dependencies run

```
pip install formasaurus[with-deps]
```

These packages may require extra steps to install, so the command above may fail. In this case install dependencies manually, on by one (follow their install instructions), then run:

```
pip install formasaurus
```

CHAPTER 2

Usage

2.1 Basic Usage

Grab some HTML:

```
>>> import requests
>>> html = requests.get('https://www.github.com/').text
```

Then use *formasaurus.extract_forms* to detect form and field types:

```
>>> import formasaurus
>>> formasaurus.extract_forms(html)
[(<Element form at 0x1150ba0e8>,
  {'fields': {'q': 'search query'}, 'form': 'search'}),
 (<Element form at 0x1150ba138>,
  {'fields': {'user[email]': 'email',
    'user[login]': 'username',
    'user[password]': 'password'},
   'form': 'registration'})]
```

Note: To detect form and field types Formasaurus needs to train prediction models on user machine. This is done automatically on first call; models are saved to a file and then reused.

formasaurus.extract_forms returns a list of (form, info) tuples, one tuple for each `<form>` element on a page. `form` is a lxml Element for a form, `info` dict contains form and field types.

Only fields which are

1. visible to user;
2. have non-empty `name` attribute

are returned - other fields usually should be either submitted as-is (hidden fields) or not sent to the server at all (fields without `name` attribute).

There are edge cases like fields filled with JS or fields which are made invisible using CSS, but all bets are off if page uses JS heavily and all we have is HTML source.

By default, info dict contains only most likely form and field types. To get probabilities pass `proba=True`:

```
>>> formasaurus.extract_forms(html, proba=True, threshold=0.05)
[(<Element form at 0x1150db408>,
  {'fields': {'q': {'search query': 0.999129068423436}},
```

```
   'form': {'search': 0.99580680143321776}}),
 (<Element form at 0x1150dbae8>,
  {'fields': {'user[email]': {'email': 0.9980438256540791},
    'user[login]': {'username': 0.9877912041558733},
    'user[password]': {'password': 0.9968113886622333}},
   'form': {'login': 0.12481875549840604,
    'registration': 0.86248202363754578}})]
```

Note that Formasaurus is less certain about the second form type - it thinks most likely the form is a registration form (0.86%), but the form also looks similar to a login form (12%).

`threshold` argument can be used to filter out low-probability options; we used 0.05 in this example. To get probabilities of all classes use `threshold=0`.

To classify individual forms use *`formasaurus.classify`* or *`formasaurus.classify_proba`*. They accept lxml <form> Elements. Let's load an HTML file using lxml:

```
>>> import lxml.html
>>> tree = lxml.html.parse("http://google.com")
```

and then classify the first form on this page:

```
>>> form = tree.xpath('//form')[0]
>>> formasaurus.classify(form)
{'fields': {'btnG': 'submit button',
  'btnI': 'submit button',
  'q': 'search query'},
 'form': 'search'}
```

```
>>> formasaurus.classify_proba(form, threshold=0.1)
{'fields': {'btnG': {'submit button': 0.9254039698573596},
  'btnI': {'submit button': 0.9642014575642849},
  'q': {'search query': 0.9959819637966439}},
 'form': {'search': 0.98794025545508202}}
```

In this example the data is loaded from an URL; of course, data may be loaded from a local file or from an in-memory object, or you may already have the tree loaded (e.g. with Scrapy).

2.2 Form Types

Formasaurus detects these form types:

```
                         precision    recall  f1-score   support

                 search       0.91      0.96      0.94       364
                  login       0.96      0.96      0.96       221
           registration       0.97      0.86      0.91       153
password/login recovery       0.88      0.88      0.88        95
        contact/comment       0.87      0.93      0.90       120
      join mailing list       0.90      0.89      0.90       107
      order/add to cart       0.95      0.66      0.78        62
                  other       0.67      0.70      0.69       122

            avg / total       0.90      0.90      0.89      1244

89.5% forms are classified correctly.
```

Quality is estimated based on cross-validation results: all annotated data is split into 20 folds, then model is trained on 19 folds and tries to predict form types in the remaining fold. This is repeated to get predictions for the whole dataset.

See also: https://en.wikipedia.org/wiki/Precision_and_recall

2.3 Field Types

By deafult, Formasaurus detects these field types:

- `username`
- `password`
- `password confirmation` - "enter the same password again"
- `email`
- `email confirmation` - "enter the same email again"
- `username or email` - a field where both username and email are accepted
- `captcha` - image captcha or a puzzle to solve
- `honeypot` - this field usually should be left blank
- `TOS confirmation` - "I agree with Terms of Service", "I agree to follow website rules", "It is OK to process my personal info", etc.
- `receive emails confirmation` - a checkbox which means "yes, it is ok to send me some sort of emails"
- `remember me checkbox` - common on login forms
- `submit button` - a button user should click to submit this form
- `cancel button`
- `reset/clear button`
- `first name`
- `last name`
- `middle name`
- `full name`
- `organization name`
- `gender`
- `day`
- `month`
- `year`
- `full date`
- `time zone`
- `DST` - Daylight saving time preference
- `country`
- `city`
- `state`

- `address` - other address information
- `postal code`
- `phone` - phone number or its part
- `fax`
- `url`
- `OpenID`
- `about me text`
- `comment text`
- `comment title or subject`
- `security question` - "mother's maiden name"
- `answer to security question`
- `search query`
- `search category / refinement` - search parameter, filtering option
- `product quantity`
- `style select` - style/theme select, common on forums
- `sorting option` - asc/desc order, items per page
- `other number`
- `other read-only` - field with information user shouldn't change
- all other fields are classified as `other`.

Quality estimates (based on 20-fold cross-validation):

	precision	recall	f1-score	support
username	0.81	0.91	0.85	187
password	0.99	0.99	0.99	338
password confirmation	0.96	0.99	0.97	97
email	0.94	0.97	0.95	544
email confirmation	0.96	0.85	0.90	26
username or email	0.82	0.41	0.55	34
captcha	0.84	0.82	0.83	83
honeypot	0.17	0.06	0.08	18
TOS confirmation	0.81	0.50	0.62	84
receive emails confirmation	0.36	0.59	0.45	83
remember me checkbox	0.94	1.00	0.97	117
submit button	0.96	0.97	0.96	334
cancel button	0.86	0.60	0.71	10
reset/clear button	1.00	0.83	0.91	12
first name	0.92	0.86	0.89	95
last name	0.88	0.85	0.86	93
middle name	1.00	0.67	0.80	6
full name	0.74	0.82	0.78	120
organization name	0.81	0.43	0.57	30
gender	0.98	0.80	0.88	75
time zone	1.00	0.71	0.83	7
DST	1.00	1.00	1.00	5
country	0.85	0.72	0.78	47
city	0.95	0.68	0.79	53

```
                        state       1.00      0.63      0.77        38
                      address       0.75      0.64      0.69        84
                  postal code       0.95      0.79      0.87        78
                        phone       0.83      0.85      0.84       102
                          fax       1.00      1.00      1.00         8
                          url       0.88      0.66      0.75        32
                       OpenID       1.00      0.75      0.86         4
                 about me text       0.50      0.33      0.40        12
                  comment text       0.86      0.93      0.89       121
     comment title or subject       0.67      0.45      0.53       121
            security question       1.00      0.44      0.62         9
 answer to security question       0.80      0.57      0.67         7
                 search query       0.89      0.95      0.92       350
search category / refinement       0.91      0.87      0.89       376
              product quantity       0.98      0.84      0.90        55
                  style select       0.93      1.00      0.97        14
                sorting option       0.87      0.50      0.63        26
                  other number       0.27      0.15      0.19        27
                     full date       0.47      0.35      0.40        20
                           day       0.96      0.88      0.92        25
                         month       0.96      0.89      0.92        27
                          year       0.97      0.88      0.92        34
              other read-only        1.00      0.42      0.59        24
                         other       0.65      0.78      0.71       710

                   avg / total       0.85      0.84      0.83      4802

83.7% fields are classified correctly.
All fields are classified correctly in 75.3% forms.
```

CHAPTER 3

How It Works

Formasaurus uses two separate ML models for form type detection and for field type detection. Field type detector uses form type detection results to improve the quality.

The model is trained on 1000+ annotated web forms - check data folder in Formasaurus repository. Most pages to annotate were selected randomly from Alexa Top 1M websites.

3.1 Form Type Detection

To detect HTML form types Formasaurus takes a <*form*> element and uses a linear classifier (Logistic Regression) to choose its type from a predefined set of types. Features include:

- counts of form elements of different types,
- whether a form is POST or GET,
- text on submit buttons,
- names and char ngrams of CSS classes and IDs,
- input labels,
- presence of certain substrings in URLs,
- etc.

See Form Type Detection.ipynb IPython notebook for more detailed description.

3.2 Field Type Detection

To detect form field types Formasaurus uses Conditional Random Field (CRF) model. All fields in an HTML form is a sequence where order matters; CRF allows to take field order in account.

Features include

- form type predicted by a form type detector,
- field tag name,
- field value,
- text before and after field,
- field CSS class and ID,

- text of field <label> element,
- field title and placeholder attributes,
- etc.

There are about 50 distinct field types.

To train field type detector we need form type labels. There are true form types available directly in training data, but in reality form type detecor will make mistakes at prediction time. So we have two options:

1. Use correct form labels in training, rely on noisy form labels at test/prediction time.
2. Use noisy (predicted) labels both at train and test time.

Strategy (2) leads to more regularized models which account for form type detector mistakes; strategy (1) uses more information.

Based on held-out dataset it looks like (1) produces better results.

We need noisy form type labels anyways, to check prediction quality. To get these 'realistic' noisy form type labels we split data into 10 folds, and then for each fold we predict its labels using form type detector trained on the rest 9 folds.

CHAPTER 4

API Reference

4.1 Classifiers

formasaurus.classifiers.**extract_forms**(*tree_or_html*, *proba=False*, *threshold=0.05*)

Given a lxml tree or HTML source code, return a list of (form_elem, form_info) tuples. form_info dicts contain results of *FormFieldClassifier.classify()* or *FormFieldClassifier.classify_proba()* calls, depending on proba parameter.

formasaurus.classifiers.**classify**(*form*)

Return {'form': 'type', 'fields': {'name': 'type', ...}} dict with form type and types of its visible submittable fields.

formasaurus.classifiers.**classify_proba**(*form*, *threshold=0.0*)

Return dict with probabilities of form and its fields belonging to various form and field classes:

```
{
    'form': {'type1': prob1, 'type2': prob2, ...},
    'fields': {
        'name': {'type1': prob1, 'type2': prob2, ...},
        ...
    }
}
```

form should be an lxml HTML <form> element. Only classes with probability >= threshold are preserved.

class formasaurus.classifiers.**FormFieldClassifier**(*form_classifier=None*, *field_model=None*)

FormFieldClassifier detects HTML form and field types.

classmethod load(*filename=None*, *autocreate=True*, *rebuild=False*)

Load extractor from file filename.

If the file is missing and autocreate option is True (default), the model is created using default parameters and training data. If filename is None then default model file name is used.

Example - load the default extractor:

```
ffc = FormFieldClassifier.load()
```

classmethod trained_on(*data_folder*)

Return Formasaurus object trained on data from data_folder

train(*annotations*)

Train FormFieldExtractor on a list of FormAnnotation objects.

classify (*form*)
Return `{'form': 'type', 'fields': {'name': 'type', ...}}` dict with form type and types of its visible submittable fields.

classify_proba (*form*, *threshold=0.0*)
Return dict with probabilities of `form` and its fields belonging to various form and field classes:

```
{
    'form': {'type1': prob1, 'type2': prob2, ...},
    'fields': {
        'name': {'type1': prob1, 'type2': prob2, ...},
        ...
    }
}
```

`form` should be an lxml HTML <form> element. Only classes with probability >= `threshold` are preserved.

extract_forms (*tree_or_html*, *proba=False*, *threshold=0.05*)
Given a lxml tree or HTML source code, return a list of `(form_elem, form_info)` tuples. `form_info` dicts contain results of *classify()* or `classify_proba`() calls, depending on `proba` parameter.

class `formasaurus.classifiers.`**FormClassifier** (*form_model=None*, *full_type_names=True*)
Convenience wrapper for scikit-learn based form type detection model.

classify (*form*)
Return form class. `form` should be an lxml HTML <form> element.

classify_proba (*form*, *threshold=0.0*)
Return form class. `form` should be an lxml HTML <form> element.

train (*annotations*)
Train FormExtractor on a list of FormAnnotation objects.

extract_forms (*tree_or_html*, *proba=False*, *threshold=0.05*)
Given a lxml tree or HTML source code, return a list of `(form_elem, form_info)` tuples. `form_info` dicts contain results of *classify()* or `classify_proba`() calls, depending on `proba` parameter.

`formasaurus.classifiers.`**instance**()
Return a shared FormFieldClassifier instance

4.2 Field Type Detection

Field type detection model is two-stage:

1. First, we train Formasaurus form type detector.
2. Second, we use form type detector results to improve quality of field type detection.

We have correct form types available directly in training data, but in reality form type detecor will make mistakes at prediction time. So we have two options:

1. Use correct form labels in training, rely on noisy form labels at test/prediction time.
2. Use noisy (predicted) labels both at train and test time.

Strategy (2) leads to more regularized models which account for form type detector mistakes; strategy (1) uses more information.

Based on held-out dataset it looks like (1) produces better results.

We need noisy form type labels anyways, to check prediction quality. To get these 'realistic' noisy form type labels we split data into 10 folds, and then for each fold we predict its labels using form type detector trained on the rest 9 folds.

formasaurus.fieldtype_model.**scorer**
Default scorer for grid search. We're optimizing for micro-averaged F1.

formasaurus.fieldtype_model.**get_Xy**(*annotations*, *form_types*, *full_type_names=False*)
Return training data for field type detection.

formasaurus.fieldtype_model.**get_form_features**(*form*, *form_type*, *field_elems=None*)
Return a list of feature dicts, a dict per visible submittable field in a <form> element.

formasaurus.fieldtype_model.**get_model**(*use_precise_form_types=True*)
Return default CRF model

formasaurus.fieldtype_model.**print_classification_report**(*annotations*, *n_folds=10*, *model=None*)
Evaluate model, print classification report

formasaurus.fieldtype_model.**tokenize**()
findall(string[, pos[, endpos]]) -> list. Return a list of all non-overlapping matches of pattern in string.

4.3 Form Type Detection

This module defines which features and which classifier the default form type detection model uses.

formasaurus.formtype_model.**get_model**(*prob=True*)
Return a default model.

formasaurus.formtype_model.**train**(*annotations*, *model=None*, *full_type_names=False*)
Train form type detection model on annotation data

formasaurus.formtype_model.**get_realistic_form_labels**(*annotations*, *n_folds=10*, *model=None*, *full_type_names=True*)
Return form type labels which form type detection model is likely to produce.

formasaurus.formtype_model.**print_classification_report**(*annotations*, *n_folds=10*, *model=None*)
Evaluate model, print classification report

This module provides scikit-learn transformers for extracting features from HTML forms.

For all features X is a list of lxml <form> elements.

class formasaurus.formtype_features.**FormElements**
Features based on form HTML elements: counts of elements of different types, GET/POST form method.

class formasaurus.formtype_features.**Bias**
The same as `clf.intercept_`, but with regularization applied. Used mostly for debugging.

class formasaurus.formtype_features.**FormText**
Text contents inside the form.

class formasaurus.formtype_features.**FormInputNames**
Names of all non-hidden <input> elements, joined to a single string.

class formasaurus.formtype_features.**FormInputHiddenNames**
Names of all <input type=hidden> elements, joined to a single string.

class formasaurus.formtype_features.**FormLinksText**
Text of all links inside the form. It is helpful because e.g. registration links inside login forms are common.

class formasaurus.formtype_features.**SubmitText**
Text of all <submit> buttons, joined to a single string.

class formasaurus.formtype_features.**FormUrl**
<form action> value

class formasaurus.formtype_features.**FormCss**
Form CSS classes and ID

class formasaurus.formtype_features.**FormInputTitle**
<input title=...> values

class formasaurus.formtype_features.**FormLabelText**
<label> values

class formasaurus.formtype_features.**FormInputCss**
CSS classes and IDs of <input> elemnts

class formasaurus.formtype_features.**OldLoginformFeatures**
Features that loginform library used.

formasaurus.formtype_features.**loginform_features**(*form*)
A dict with features from loginform library

4.4 Working with Training Data

A module for working with annotation data storage.

class formasaurus.storage.**Storage**(*folder*)
A wrapper class for HTML forms annotation data storage. The goal is to store the type of each <form> element from a web page. The data is stored in a folder with the following structure:

```
config.json
index.json
html/
    example.org-0.html
    example.org-1.html
    foo.example.org-0.html
    ...
```

`html` folders contains raw contents of the webpages. `index.json` file contains a JSON dict with the following records:

```
"RELATIVE-PATH-TO-HTML-FILE": {
    "url": "URL",
    "forms": ["type1", "type2", ...],
    "visible_html_fields": [
        {"name1": "type1", "name2": "type2", ...},
        ...
    ],
}
```

Key is the relative path to a file with page contents (e.g. "html/example.org-1.html"). Values:

- "url" is an URL the webpage is downloaded from.

- "forms" contains an array of form type identifiers. There must be an identifier per each `<form>` element on a web page.
- "visible_html_fields" contains an array of objects, one object per `<form>` element; each object is a mapping from field name to field type identifier.

Possible form and field types are stored in `config.json` file; you can read them using `get_form_types()` and `get_field_types()`.

initialize(*config*, *index=None*)
Create folders and files for a new storage

get_index()
Read an index

write_index(*index*)
Save an index

get_config()
Read meta information, including form and field types

get_field_schema()
Return `AnnotationSchema` instance. *r.types* is an OrderedDict with field type names {full_name: short_name}; *r.types_inv* is a {short_name: full_name} dict; *r.na_value* is a short name of type name used for unannotated fields.

get_form_schema()
Return `AnnotationSchema` instance. *r.types* is an OrderedDict with form type names {full_name: short_name}; *r.types_inv* is a {short_name: full_name} dict; *r.na_value* is a short name of type name used for unannotated forms; *r.skip_value* is a short name of a type name which should be skipped.

add_result(*html*, *url*, *form_answers=None*, *visible_html_fields=None*, *index=None*, *add_empty=True*)
Save HTML source and its <form> and form field types.

iter_annotations(*index=None*, *drop_duplicates=True*, *drop_na=True*, *drop_skipped=True*, *simplify_form_types=False*, *simplify_field_types=False*, *verbose=False*, *leave=False*)
Return an iterator over `FormAnnotation` objects.

iter_trees(*index=None*)
Return an iterator over `(filename, tree, info)` tuples where `filename` is a relative file name, `tree` is a lxml tree and `info` is a dictionary with annotation data.

get_tree(*path*, *info=None*)
Load a single tree. `path` is a relative path to a file (key in index.json file), `info` is annotation data (value in index.json file).

check()
Check that items in storage are correct; print the problems found. Return the number of errors found.

get_fingerprint(*form*)
Return form fingerprint (a string that can be used for deduplication).

get_form_type_counts(*drop_duplicates=True*, *drop_na=True*, *simplify=False*, *verbose=True*)
Return a {formtype: count} collections.Counter

print_form_type_counts(*simplify=False*)
Print the number annotations of each form types in this storage

generate_filename(*url*)
Return a name for a new file

class formasaurus.annotation.**AnnotationSchema**(*types*, *types_inv*, *na_value*, *skip_value*, *simplify_map*)

na_value
Alias for field number 2

simplify_map
Alias for field number 4

skip_value
Alias for field number 3

types
Alias for field number 0

types_inv
Alias for field number 1

class formasaurus.annotation.**FormAnnotation**
Annotated HTML form

fields
{"field name": "field type"} dict.

fields_annotated
True if form has fields and all fields are annotated.

fields_partially_annotated
True when some fields are annotated and some are not annotated.

field_elems
Return a list of lxml Elements for fields which are annotated. Fields are returned in in order they appear in form; only visible submittable fields are considered.

field_types
A list of field types, in order they appear in form. Only visible submittable fields are considered.

field_types_full
A list of long field type names, in order they appear in form. Only visible submittable fields are considered.

type_full
Full form type name

formasaurus.annotation.**get_annotation_folds**(*annotations*, *n_folds*)
Return (train_indices, test_indices) folds iterator. It is guaranteed forms from the same website can't be both in train and test parts.

We must be careful when splitting the dataset into training and evaluation parts: forms from the same domain should be in the same "bin". There could be several pages from the same domain, and these pages may have duplicate or similar forms (e.g. a search form on each page). If we put one such form in training dataset and another in evaluation dataset then the metrics will be too optimistic, and they can make us to choose wrong features/models. For example, train_test_split from scikit-learn shouldn't be used here. To fix it LabelKFold from scikit-learn is used.

4.5 HTML Processing Utilities

HTML processing utilities

formasaurus.html.**remove_by_xpath**(*tree*, *xpath*)
Remove all HTML elements which match a given XPath expression.

formasaurus.html.**load_html**(*tree_or_html*, *base_url=None*)
Parse HTML data to a lxml tree. `tree_or_html` must be either unicode or utf8-encoded (even if original page declares a different encoding).

If `tree_or_html` is not a string then it is returned as-is.

formasaurus.html.**get_cleaned_form_html**(*form*, *human_readable=True*)
Return a cleaned up version of <form> HTML contents. If `human_readable` is True, HTML is cleaned to make source code more readable for humans; otherwise it is cleaned to make rendered form more safe to render.

formasaurus.html.**get_field_names**(*elems*)
Return unique name attributes

formasaurus.html.**get_visible_fields**(*form*)
Return visible form fields (the ones users should fill).

formasaurus.html.**get_fields_to_annotate**(*form*)
Return fields which should be annotated:

1.they should be visible to user, and

2.they should have non-empty name (i.e. affect form submission result).

formasaurus.html.**escaped_with_field_highlighted**(*form_html*, *field_name*)
Return escaped HTML source code suitable for displaying; fields with name==field_name are highlighted.

formasaurus.html.**highlight_fields**(*html*, *field_name*)
Return HTML source code with all fields with name==field_name highlighted by adding `formasaurus-field-highlighted` CSS class.

formasaurus.html.**add_text_after**(*elem*, *text*)
Add text after elem

formasaurus.html.**add_text_before**(*elem*, *text*)
Add text before elem

formasaurus.html.**get_text_around_elems**(*tree*, *elems*)
Return (before, after) tuple with {elem: text} dicts containing text before a specified lxml DOM Element and after it.

formasaurus.formhash.**get_form_hash**(*form*, *only_visible=True*)
Return a string which is the same for duplicate forms, but different for forms which are not the same.

If only_visible is True, hidden fields are not taken in account.

4.6 Other Utilities

formasaurus.utils.**dependencies_string**()
Return a string with versions of formasaurus, numpy, scipy and scikit-learn.

Saved scikit-learn models may be not compatible between different numpy/scipy/scikit-learn versions; a string returned by this function can be used as a part of file name.

formasaurus.utils.**add_scheme_if_missing**(*url*)

```
    >>> add_scheme_if_missing("example.org")
    'http://example.org'
    >>> add_scheme_if_missing("https://example.org")
    'https://example.org'
```

formasaurus.utils.**get_domain**(*url*)

```
>>> get_domain('example.org')
'example'
>>> get_domain('foo.example.co.uk')
'example'
```

formasaurus.utils.**inverse_mapping**(*dct*)
Return reverse mapping:

```
>>> inverse_mapping({'x': 5})
{5: 'x'}
```

formasaurus.utils.**at_root**(**args*)
Return path relative to formasaurus source code

formasaurus.utils.**thresholded**(*dct*, *threshold*)
Return dict `dct` without all values less than threshold.

```
>>> thresholded({'foo': 0.5, 'bar': 0.1}, 0.5)
{'foo': 0.5}
```

```
>>> thresholded({'foo': 0.5, 'bar': 0.1, 'baz': 1.0}, 0.6)
{'baz': 1.0}
```

```
>>> dct = {'foo': 0.5, 'bar': 0.1, 'baz': 1.0, 'spam': 0.0}
>>> thresholded(dct, 0.0) == dct
True
```

formasaurus.utils.**download**(*url*)
Download a web page from url, return its content as unicode.

formasaurus.text.**tokenize**()
Tokenize text

formasaurus.text.**ngrams**(*seq*, *min_n*, *max_n*)
Return min_n to max_n n-grams of elements from a given sequence.

formasaurus.text.**token_ngrams**(*tokens*, *min_n*, *max_n*)
Return n-grams given a list of tokens.

formasaurus.text.**normalize_whitespaces**(*text*)
Replace newlines and whitespaces with a single white space

formasaurus.text.**normalize**(*text*)
Default text normalization function

formasaurus.text.**number_pattern**(*text*, *ratio=0.3*)
Replace digits with X and letters with C if text contains > ratio of digits; return empty string otherwise.

4.7 IPython Annotation Widgets

IPython widgets for data annotation.

formasaurus.widgets.**AddPageWidget**(*storage*)
Widget used to add a new web page to dataset.

formasaurus.widgets.**MultiFormAnnotator**(*annotations*, *annotate_fields=True*, *annotate_types=True*, *save_func=None*)
A widget with a paginator for annotating multiple forms.

formasaurus.widgets.**FormAnnotator**(*ann*, *annotate_fields=True*, *annotate_types=True*, *max_fields=80*)
Widget for annotating a single HTML form.

formasaurus.widgets.**FormTypeSelect**(*ann*)
Form type edit widget

formasaurus.widgets.**FieldTypeSelect**(*ann*, *field_name*)
Form field type edit widget

formasaurus.widgets.**RawHtml**(*html*, *field_name=None*, *max_height=500*, ***kwargs*)
Widget for displaying HTML form, optionally with a field highlighted

formasaurus.widgets.**HtmlCode**(*form_html*, *field_name=None*, *max_height=None*, ***kwargs*)
Show HTML source code, optionally with a field highlighted

formasaurus.widgets.**HtmlView**(*form*, *field_name=None*)
Show both rendered HTML and its simplified source code

formasaurus.widgets.**get_pager_elements**(*min*, *max*)
Return (back, forward, slider) widgets.

CHAPTER 5

Contributing

5.1 Development

- Source code: https://github.com/TeamHG-Memex/Formasaurus
- Issue tracker: https://github.com/TeamHG-Memex/Formasaurus/issues

Feel free to submit ideas, bugs reports and pull requests.

In order to run tests install tox, then type

```
tox
```

from the source checkout.

The easiest way to improve classification quality is to add more training examples. Use "Add New Pages" and "Annotate" IPython notebooks for that.

If you want to improve Formasaurus ML models check *How It Works* section.

5.2 Authors

- Mikhail Korobov <kmike84@gmail.com>

5.3 License

License is MIT.

CHAPTER 6

Changes

6.1 0.5 (2015-12-19)

This is a major backwards-incompatible release.

- Formasaurus now can detect field types, not only form types;
- API is changed - check the updated documentation;
- there are more form types detected;
- evaluation setup is improved;
- annotation UI is rewritten using IPython widgets;
- more training data is added.

6.2 0.2 (2015-08-10)

- Python 3 support;
- fixed model auto-creation.

6.3 0.1 (2015-07-09)

Initial release.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

f

A

B

C

D

E

F

T

W