Yalign Documentation

Release 0.1.1

Gonzalo García Berrotarán, Rafael Carrascosa, Andrew Vine

Jun 28, 2017

Contents

Yalign is a tool for extracting parallel sentences from comparable corpora.

Statistical Machine Translation relies on parallel corpora (eg.. europarl) for training translation models. However these corpora are limited and take time to create. Yalign is designed to automate this process by finding sentences that are close translation matches from comparable corpora. This opens up avenues for harvesting parallel corpora from sources like translated documents and the web.

CHAPTER 1

Installation

Yalign requires that you install scikit-learn.

After that you can install Yalign from PyPi via pip:

```
sudo pip install yalign
```

CHAPTER 2

Usage

Firstly we need to download and unpack the english to spanish model.

```
wget http://yalign.machinalis.com/models/0.1/en-es.tar.gz
tar -xvzf en-es.tar.gz
```

Now we can use the **yalign-align** script along with the english to spanish model to align two web pages.

```
yalign-align en-es http://en.wikipedia.org/wiki/Antiparticle http://es.wikipedia.org/
↪wiki/Antipart%C3%ADcula
```

Yalign is not limited to any one language pair. By creating your own models you can align any two languages.

CHAPTER 3

Demo

See **yalign** in action right here.

CHAPTER 4

Contents:

Installation

Dependencies

Yalign has the following dependencies:

scikit-learn
docopt
simpleai
nltk
beautifulsoup4
mock
html5lib
lxml

Installing From PyPi

Yalign requires that you first install scikit-learn .

```
sudo pip install yalign
```

Installing From source

The source code is available at the yalign github repository.

```
git clone git@github.com:machinalis/yalign.git
cd yalign
sudo pip install -r requirements.txt
```

Tutorial

In this tutorial you will create a yalign model to align english and spanish documents. This same process can be applied to create models in the languages that you require.

To follow along in this tutorial you will need to download and unpack the tutorial data:

```
wget http://yalign.machinalis.com/tutorial.tar.gz
tar -xvzf tutorial.tar.gz
```

You should now have tutorial folder with two files: **dictionary.csv** and **corpus.en-es**. These two files will be explained shortly.

Let's go and create our model..

Create A Model

During alignment Yalign uses a trained classifier to classify if a sentence is a translation or not. This classifier is dependent on the languages of the sentences to be aligned. This means that we need to train a model for the two languages that we want to align.

Training a model requires two inputs:

1. A dictionary (dictionary.csv)

This is a csv file for the two languages to be aligned. Each line consists of a word, a translation and a translation probability. An english to spanish dictionary (dictionary.csv) is provided in the tutorial folder.

Note: How can I create other dictionaries?

If you have worked with phrase tables before you will recognise that this information can be gleaned from a phrase table of 1-Grams. For conveniance we have included a script, **yalign-phrasetable-csv**, to convert an existing phrase table to a csv file.

2. A parallel corpus (corpus.en-es)

The second requirement for training is an existing parallel corpus.

An english to spanish corpus (corpora.en-es) is provided in the tutorial folder. This corpus is taken from the europarl corpus provided by *RANLP5*. The format of this corpus is plaintext but tmx is also accepted.

Note: Where can I find other parallel corpora?

A good resource for corpora is the opus site.

So now that we have a dictionary and parallel corpus we are ready to use **yalign-train** to train our model.

In the tutorial folder:

Create a folder where the model will be saved:

```
mkdir en-es
```

Run **yalign-train**:

```
yalign-train -a en -b es corpus.en-es dictionary.csv en-es
```

Once the training is finished there should be two files in the model folder (en-es):

- aligner.pickle: A pickle of the yalign model.
- metadata.json: Some metadata and parameters for the model.

We can now use this model to align english and spanish documents.

Align Two Documents

An important part of sentence alignment is making sure that sentences are properly split. To accomplish this Yalign uses the nltk punkt module for sentence splitting in various languages. If you do not have these sentence splitters installed you can install them from the command line with:

```
python -m nltk.downloader -q punkt
```

To perform an alignment we use the **yalign-align** script.

The script takes a model folder (en-es) and the documents (urls or plantext files) to align as input. The results of the alignment are then written to stdout.

Let's align an english and spanish wikipedia page using our new model.

```
yalign-align -a en -b es en-es http://en.wikipedia.org/wiki/Antiparticle http://es.
↪wikipedia.org/wiki/Antipart%C3%ADcula
```

And that's it. You have successfully created your first alignment model!

These same steps can be followed to create models in other languages.

Implementation

Yalign is implemented using:

- A sentence similarity metric. Given two sentences it produces a rough estimate (a number between 0 and 1) of how likely are those two sentences to be a translation of each other.
- A sequece aligner, such that given two documents (a list of sentences) it produces an alignment which maximizes the sum of the individual (per sentence pair) similarities.

So Yalign's main algorithm is actually a pretty wrapper to a standard sequence alignment algorithm.

For the sequence alignment Yalign uses a variation of the Needleman-Wunch algorithm to find an optimal alignment between the sentences in two given documents. On the good side, the algorithm has polynomial time worst case complexity and it produces an optimal alignment. On the bad side it can't handle alignments that cross each other or alignments from two sentences into a single one (even tough is possible to modify the current implementation to handle those cases).

Since the sentence similarity is a computationally expensive operation, the mentioned "variation" on the Needleman-Wunch algorithm consists in using the A* to explore the search space instead of using the classical dynamic programming aproach (which would always requiere $N * M$ calls to the sentence similarity metric).

After the alignment, only sentences that have a high probability of being translations are included in the final alignment. Ie, the result is filtered in order to deliver high quality alignments. To do this, a threshold value is used such that if the sentence similarity metric is bad enough that pair is excluded.

For the sentence similarity metric the algorithm uses a statistical classifier's likelihood output and adapts it into the 0-1 range.

The classifier is trained to determine if a pair of sentences are translations of each other or not (a binary value). The particular classifier used for this project is a Support Vector Machine. Besides being excelent classifiers, SVMs can provide a distance to the separation hyperplane during classification, and this distance can be easily modified using a Sigmoid Function to return a likelihood between 0 and 1.

The use of a classifier means that the quality of the alignment is dependent not only on the input but also on the quality of the trained classifier.

Library Reference

Main API

Examples of how to use the yalign API:

```
# Load a model from model that was saved to a folder eg.. en-es:

from yalign import YalignModel

model = YalignModel.load('en-es')

# Align text

from yalign import text_to_document

english_text = """Virginia's eyes filled with tears and she hid her head in her hands.
                   The Duke rose and kissed his wife lovingly."""

spanish_text = """¿No tiene ningún lugar donde pueda dormir?
                   Los ojos de Virginia se llenaron de lágrimas y óculto su rostro␣
↪entre los manos."""

english_sentences = text_to_document(english_text, 'en')
spanish_sentences = text_to_document(spanish_text, 'es')

pairs = model.align(english_sentences, spanish_sentences)

# Align html

from yalign import html_to_document

english_html = """<html><body><p>
                   Virginia's eyes filled with tears and she hid her head in her␣
↪hands.
                   The Duke rose and kissed his wife lovingly."
                   </p></body></html>"""

spanish_html = """<html><body><p>
                   ¿No tiene ningún lugar donde pueda dormir?
                   Los ojos de Virginia se llenaron de lágrimas y óculto su rostro␣
↪entre los manos.
```

```
                    </p></body></html>"""

english_sentences = html_to_document(english_html, 'en')
spanish_sentences = html_to_document(spanish_html, 'es')

pairs = model.align(english_sentences, spanish_sentences)

# Align srt

from yalign import srt_to_document

english_srt = """1\n00:00:49,160 --> 00:00:50,992\n
                  <i>Virginia's eyes filled with tears and she hid her head in her␣
↪hands.</i>\n\n
                  2\n00:00:51,734 --> 00:00:53,577\n
                  <i>The Duke rose and kissed his wife lovingly.</i>"""

spanish_srt = """1\n00:00:49,160 --> 00:00:50,992\n
                  <i>¿No tiene ningún lugar donde pueda dormir?</i>\n\n
                  2\n00:00:51,734 --> 00:00:53,577\n
                  <i>Los ojos de Virginia se llenaron de lágrimas y óculto su rostro␣
↪entre los manos.</i>"""

english_sentences = srt_to_document(english_srt, 'en')
spanish_sentences = srt_to_document(spanish_srt, 'es')

pairs = model.align(english_sentences, spanish_sentences)
```

Datatypes

Module of some basic data types.

class `yalign.datatypes.`**`ScoreFunction`**(*min_bound*, *max_bound*)
Bases: `object`

Abstract Base class for callable objects that provide a real value score. The min_bound and max_bound are used to assert the score range.

class `yalign.datatypes.`**`Sentence`**(*iterable=None*, *text=None*)
Bases: `list`

`check_is_tokenized`()

`to_text`()

class `yalign.datatypes.`**`SentencePair`**(*sentence_a*, *sentence_b*, *aligned=None*)
Bases: `list`

An association of two sentences with one attribute to indicate if they are considered aligned.

Evaluation

`evaluation` Module

Module for the evaluation of sequence alignment accuracy.

`yalign.evaluation.`**`F_score`**(*xs*, *ys*, *beta=0.01*)
Returns the F score described here: http://en.wikipedia.org/wiki/F1_score for list *xs* against to the list *ys*.

Make beta smaller to give more weight to the precision.

`yalign.evaluation.`**`alignment_percentage`**(*document_a*, *document_b*, *model*)
Returns the percentage of alignments of *document_a* and *document_b* using the model provided.

- *document_a* and *document_b* are two lists of Sentences to align.
- *model* can be a YalignModel or a path to a yalign model.

`yalign.evaluation.`**`classifier_precision`**(*document_a*, *document_b*, *model*)
Runs a ten-fold validation on the classifier and returns a value between 0 and 100. Higher is better.

`yalign.evaluation.`**`correlation`**(*classifier*, *dataset=None*)
Calculates the correlation of the attributes of a classifier.

For more information see:

- http://en.wikipedia.org/wiki/Correlation_and_dependence

`yalign.evaluation.`**`evaluate`**(*parallel_corpus*, *model*, *N=100*)
Returns statistics for N document alignment trials. The documents are generated from the parallel corpus.

- *parallel_corpus*: A file object
- *model*: A YalignModel
- *N*: Number of trials

`yalign.evaluation.`**`precision`**(*xs*, *ys*)
Precision of list *xs* to list *ys*.

`yalign.evaluation.`**`recall`**(*xs*, *ys*)
Recall of list *xs* to list *ys*.

Input Conversion

A module of helper functions for dealing with various inputs.

`yalign.input_conversion.`**`generate_documents`**(*filepath*, *m=20*, *n=20*)
Document generator. Documents are created from the parallel corpus and will be between m and n lines long.

`yalign.input_conversion.`**`html_to_document`**(*html*, *language='en'*)
Returns html text as list of Sentences

`yalign.input_conversion.`**`parallel_corpus_to_documents`**(*filepath*)
Transforms a parallel corpus file format into two documents. The Parallel corpus has:

- One sentences per line.
- One line of each language.
- Sentences are tokenized and tokens are space separated.
- The file encoding is UTF-8

For example:

> This is a sentence . Esto es una oración . And this , my friend , is another . Y esta , mi amigo , es otra .

`yalign.input_conversion.`**`parse_training_file`**(*training_file*)
Reads SentencePairs from a training file.

`yalign.input_conversion.`**`srt_to_document`**(*text*, *lang='en'*)
Convert a string of srt into a list of Sentences.

`yalign.input_conversion.`**`text_to_document`**(*text*, *language='en'*)
Returns string text as list of Sentences

`yalign.input_conversion.`**`tmx_file_to_documents`**(*filepath*, *lang_a=None*, *lang_b=None*)
Converts a tmx file into two lists of Sentences. The first for language lang_a and the second for language lang_b.

`yalign.input_conversion.`**`tokenize`**(*text*, *language='en'*)
Returns a Sentence with Words (ie, a list of unicode objects)

Sentence Pair Score

Module for code related to scoring sentence pairs.

class `yalign.sentencepairscore.`**`SentencePairScore`**
Bases: *`yalign.datatypes.ScoreFunction`*

This class provides a score of how close two sentences are to being translations of each other.

`SCORE_MULTIPLIER` **= 3**

`logistic_function`(*x*)
See: http://en.wikipedia.org/wiki/Logistic_function

`train`(*pairs*, *word_score_function*)
Trains the sentence pair likelihood score using examples. *pairs* is an interable of *SentencePair* instances. *word_score_function* is an instance of ScoreFunction, perhaps even an instance of *WordPairScore*.

`word_pair_score`

class `yalign.sentencepairscore.`**`SentencePairScoreProblem`**(*word_pair_score*)
Bases: `simpleai.machine_learning.models.ClassificationProblem`

Provides the classifier attributes.

`number_of_word_pair_scores`(*sentence_pair*)
The number of the word pair scores divided by the number of words of the longest sentence.

`ratio_of_character_count`(*sentence_pair*)
The ratio of the sentence with the least characters over the sentence with the most characters.

`sum_of_word_pair_scores`(*sentence_pair*)
The sum of the word pair scores divided by the word count of the longest sentence.

`target`(*sentence_pair*)
Returns if these sentences are translations of each other

Sequence Aligner

Module for handling sequence alignment.

class `yalign.sequencealigner.`**`SequenceAligner`**(*score*, *gap_penalty*)
Bases: `object`

Aligns two sequences.

class `yalign.sequencealigner.`**`SequenceAlignmentSearchProblem`**(*xs*, *ys*, *score*, *gap_penalty*)
Bases: `simpleai.search.models.SearchProblem`

Represents and manipulates the search space for a sequence alignment problem. Used by simpleai's graph search algorithm.

`actions`(*state*)
Returns the next actions from a given state. A state is an alignment (tuple of indexes from either sequence). An action is a the next alignment to consider with a score for that alignment.

`cost`(*state1*, *action*, *state2*)
Cost of this action.

`heuristic`(*state*)
A heuristic for A* type searches. Currently we return The distance of this state from the diagonal in a N*M lattice where N and M are the lengths of the two sequences.

`is_goal`(*state*)
Are we finished aligning? True when our when state is the alignment (N, M) where N and M are the lengths of the two sequences.

`result`(*state*, *action*)
Returns the next state for this state, action pair.

SVM

Module for code dealing with the classifier.

class `yalign.svm.`**`SVMClassifier`**(*dataset*, *problem*)
Bases: `simpleai.machine_learning.models.Classifier`

A Support Vector Machine classifier to classify if a sentence is a translation of another sentence.

`classify`(*sentence_pair*)
Classify if this SentencePair *sentence_pair* has sentences that are translations of each other.

`learn`()
Train the classifier.

`score`(*data*)
The score is positive for an alignment.

Tokenizers

`tokenizers` Module

Module providing tokenizers for various languages.

`yalign.tokenizers.`**`get_tokenizer`**(*language*)
Get a tokenizer for a two character language code.

Train Data Generation

Module to generate training data.

`yalign.train_data_generation.`**`training_alignments_from_documents`**(*document_a*, *document_b*)
Returns an iterable of SentencePairs to be used for training. The inputs *document_a* and *document_b* are both lists of Sentences made from a parallel corpus.

`yalign.train_data_generation.`**`training_scrambling_from_documents`**(*document_a*, *document_b*)

Returns a tuple *(scrambled_a, scrambled_b, correct_alignments)* where:

- *scrambled_a* is a scrambled version of document_a.
- *scrambled_b* is a scrambled version of document_b.
- ***correct_alignments* are all the correct sentence alignments that exist** between *scrambled_a* and *scrambled_b*.

Utils

Module for miscellaneous functions.

class `yalign.utils.`**`CacheOfSizeOne`**(*f*)

Bases: `object`

Function wrapper that provides caching.

`f` **= None**

class `yalign.utils.`**`Memoized`**

Bases: `collections.defaultdict`

`yalign.utils.`**`host_and_page`**(*url*)

Splits a *url* into the hostname and the rest of the url.

`yalign.utils.`**`read_from_url`**(*url*)

GET this *url* and read the response.

`yalign.utils.`**`write_tmx`**(*stream*, *sentence_pairs*, *language_a*, *language_b*)

Writes the SentencePair's out in tmx format,

Word Pair Score

Module for scoring pairs of words.

class `yalign.wordpairscore.`**`WordPairScore`**(*dictionary_file*)

Bases: *`yalign.datatypes.ScoreFunction`*

Provides the probability that two words are translations of each other.

Yalign Model

class `yalign.yalignmodel.`**`MetadataHelper`**(*metadata*)

Bases: `dict`

class `yalign.yalignmodel.`**`YalignModel`**(*document_pair_aligner=None*, *threshold=None*, *metadata=None*)

Bases: `object`

Main Yalign class. It provides methods to train a alignment model, to load a model from a folder and to align two documents.

`align`(*document_a*, *document_b*)

Try to detect aligned sentences from the comparable documents *document_a* and *document_b*. The returned alignments are expected to meet the F-measure for which the model was trained for.

align_indexes(*document_a*, *document_b*)
Same as *align* but returning indexes in documents instead of sentences.

classmethod load(*model_directory*)
This method to loads an existing YalignModel from the path to the folder where it's contained.

optimize_gap_penalty_and_threshold(*document_a*, *document_b*, *real_alignments*)
Given documents *document_a* and *document_b* (not necesarily aligned) and the *real_alignments* for that documents train the YalignModel instance to maximize the target F-measure (the quality measure).

real_alignments is a list of indexes (i, j) of *document_a* and *document_b* respectively indicating that those sentences are aligned. Pairs not included in *real_alignments* are assumed to be wrong alignments.

save(*model_directory*)
Store a serialization of a YalignModel instance in a given folder. Metadata is stored in a separate file.

sentence_pair_score

word_pair_score

yalign.yalignmodel.**apply_threshold**(*alignments*, *threshold*)

yalign.yalignmodel.**basic_model**(*corpus_filepath*, *word_scores_filepath*, *lang_a=None*, *lang_b=None*)
Creates and trains a *YalignModel* with the basic configuration and default values.

corpus_filepath is the path to a parallel corpus used for training, it can be:

- a csv file with two sentences and alignement information, or
- a tmx file with correct alignments (a regular parallel corpus), or
- a text file with interleaved sentences (one line in language A, the next in language B)

word_scores_filepath is the path to a csv file (possibly gzipped) with word dictionary data. (for ex. "house,casa,0.91").

lang_a and *lang_b* are requiered for the tokenizer in the case of a tmx file. In the other cases is not necesary because it's assumed that the words are already tokenized.

yalign.yalignmodel.**best_threshold**(*real_alignments*, *predicted_alignments*)
Returns the best F score and threshold value for this gap_penalty

yalign.yalignmodel.**pre_filter_alignments**(*alignments*)

yalign.yalignmodel.**random_sampling_maximizer**(*F*, *min_*, *max_*, *n=None*)

yalign.yalignmodel.**score_with_best_threshold**(*aligner*, *xs*, *ys*, *gap_penalty*, *real_alignments*)

Yalign is a Machinalis project. You can view our other open source contributions here.

The Yalign Team:

Laura Alonso Alemany
Elías Andrawos
Rafael Carrascosa
Gonzalo García Berrotarán
Andrew Vine

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Bibliography

[RANLP5] Jörg Tiedemann, 2009, News from OPUS - A Collection of Multilingual Parallel Corpora with Tools and Interfaces. In N. Nicolov and K. Bontcheva and G. Angelova and R. Mitkov (eds.) Recent Advances in Natural Language Processing (vol V), pages 237-248, John Benjamins, Amsterdam/Philadelphia

Python Module Index

Index