

## PPattach Tutorial

System PPAttach tackles the problem of prepositional phrase attachment by incorporating semantic knowledge derived from the lexico-semantic ontologies such as VERBNET and WORDNET. The system assumes input in the form of set of tuples

*T1*: (verb, noun, preposition, noun)

For a given set of tuples PPAttach will return its decision on each tuple on whether it triggers verb or noun attachment. PPAttach uses machine learning methods to implement its decision procedure. Machine learning methods are commonly used for implementing classification procedures called classifiers. In supervised learning, the classifier is first trained on a set of labeled data (training data) that is representative of the domain of interest. Typically labeled data consists of pairs of input objects and a desired output. An input object is often summarized by so called feature vector. The trained classifier is then used to carry out classification decisions for unseen data (testing data). PPAttach uses classic “Ratanaparkhi” dataset, composed of labeled/annotated tuples of the form (*T1*), for “training” and “testing”. Weka – a machine learning tool of the University of Waikato <http://www.cs.waikato.ac.nz/ml/weka/> – is used within the framework to carry out the classification.

Site

<http://www.unomaha.edu/nlpkr/software/ppattach/>

is the project's website which contains a link to the paper on

[1] “*Prepositional Phrase Attachment Problem Revisited: How VERBNET Can Help*” by Daniel Bailey, Yuliya Lierler, Benjamin Susman, In *Proceedings of the 11th International Conference on Computational Semantics (IWCS)*, 2015.

This paper is the best resource for details on the implemented techniques.

This document provides directions on setting up, running, and extending the PPAttach system. The PPAttach system is composed of two main components. One component is responsible for building feature vectors for given tuples of the form (*T1*), another component is responsible for processing these feature vectors and performing the classification itself. The former component is written in python by the authors of the project. The later component relies on Weka.

### *Project Setup:*

Instructions are provided for Linux users (but modulo command line commands these instructions can easily be adapted on Windows).

For PKI-Linux lab users:

% **cp** -R /nlpkr/ppattach/ \_\_the\_directory\_of\_your\_choice\_\_

For general public:

Download and unzip the following file:

<http://www.unomaha.edu/nlpkr/software/ppattach/ppattach.zip>

in \_\_the\_directory\_of\_your\_choice\_\_ .

Command

% **pwd**

will give you the complete path to `__the_directory_of_your_choice__`

We will refer to this path as

`/home/ylierler/`

so that once you copy or download the project files you will have the directory

`/home/ylierler/ppattach`

that includes the following files and directories:

1. `data`
2. `code`
3. `README.docx`
4. `run-naivebayes-weka.sh`
5. `weka`

File structure explanation:

- `Data`
  - *data* directory contains the tuples from the complete Ratanaparkhi's dataset. In this directory, each filename represents a separation of preposition(s). For example, the file named *by* consists only of tuples of which the preposition *by* is used. The file *all* consists of all prepositions and *not\_of* consists of all prepositions except *of*.
- `Code`
  - *code* directory contains one of the main parts of PPattach system. Program `ppattach.py` is responsible for generating the feature vectors given a particular tuple set from *data* in a format that is acceptable by Weka. The output is saved in the *weka* directory. A description of each existing feature in PPattach is summarized in [1].
  - `ppattach.py` – This is the main driver of the program
  - `features.py` – This contains all the implemented features outlined in [1]
  - `summarizer.py` – This transforms the results gathered from each feature into a Weka compatible file format (`.arff`). All resultant files are put in the *weka* directory
  - `additionalFeatures.py` – This is where additional features can be added
- `README.doc`
  - This file
- `run-naivebayes-weka.sh`
  - Runs Weka (on the generated `.arff` files) using the NaiveBayes classifier with 10 fold cross-validation
  - Assumes Weka is installed and `weka.jar` is in the `CLASSPATH`.
- `Weka`
  - Contains all the Weka compatible files generated from the feature vectors established by PPattach

### Command Line Instructions:

% `cd /home/ylierler/ppattach`

% `python code/ppattach.py -h`

will instruct the main source of this project, `code/ppattach.py`, to produce the information on the arguments that the program takes.

When running “`python code/ppattach.py`” you have the option to set your own command line arguments. The prepositions to test against can be specified via the `'-p'` directive. The `'-f'` directive

specifies features one would like to consider.

For example, to test the preposition “by” with the “verbNetFull” and “nominalization” features, I could do so by altering the command line (program) arguments as follows:

```
% python code/ppattach.py -p by -f verbNetFull nominalization
```

*Note:* Each run of PPattach effectively replaces the contents of the *weka* folder. In the previous example, *by-generic.arff* and *by-generic-plus.arff* would be generated (and overwrite the file if it were already present)

## Testing with Weka

Once your data has been populated, you are ready to test your results with the classifier.

### Linux

Run the following:

```
% cd /home/ylierler/ppattach  
% sh run-naivebayes-weka.sh
```

The output can be redirected to a file if you'd like.

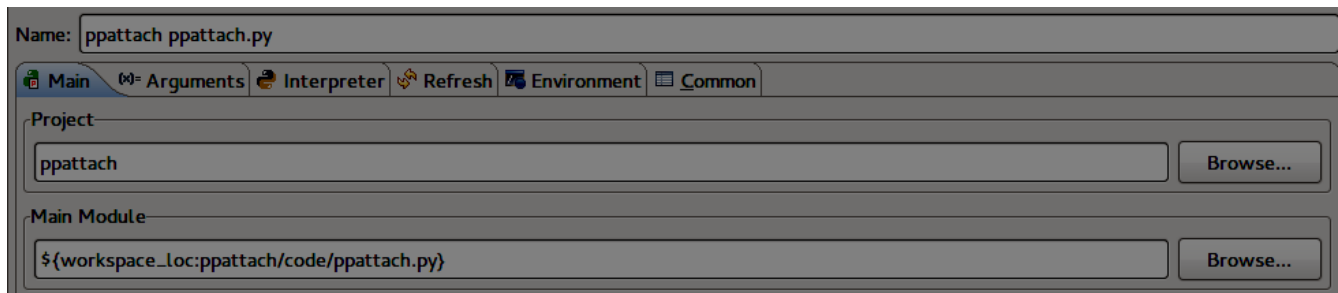
### Windows

In Explorer (or the command line) go to */home/ylierler/ppattach/*.

- Run “run-naivebayes-weka.bat”.
- The output will be stored in “results.txt” or an output file can be specified as a command line parameter.

## Eclipse Setup:

1. Open Eclipse.
2. Go to: File → New → Project.. → PyDev -> PyDev Project  
**Create a Python Project**  
Project name: *ppattach*  
**Uncheck** Use default location  
Location: */home/ylierler/ppattach/*  
**Ensure** Grammar Version is 2.7  
(In case you are prompted to configure Interpreter: use autoconfig option)
3. Go to: Run → Run As → Python Run → New  
**Change** Name to *ppattach ppattach.py*  
**Change** Project to *ppattach*  
**Change** Main Module to *\${workspace\_loc:ppattach/code/ppattach.py}*



4. Now go to the Arguments Tab → Working directory

**Change** Working directory to **Other:** `${workspace_loc:ppattach}`

**Click** Apply button and then Close button

## *Running in Eclipse:*

### Getting Started

Start the system by

Go to: Run → Run As → 1 Python Run → Select code/ppattach.py

Command line arguments can be added by going to the menu:

Run → Run Configurations -> Arguments Tab → Program arguments

In this area, for example, you can type “-h”, then click “Apply” and “Run”

An explanation of valid command line arguments should be listed in the console. This is your main way to interface with PPattach.

## *SAMPLE ASSIGNMENT for a natural language processing project:*

### *Feature Development*

All development should be done in code/additionalFeatures.py. A dummy feature has been given in this file. You may call the feature(s) whatever you want, but ensure that the *results* dictionary uses the *features*' name as a key. The python dictionary *results* is an instance variable of code/features.py and is inherited by code/additionalFeatures.py

Ideas for feature development may include:

- Analyzing a specific preposition and creating relevant features to capture this analysis (what was done with 'with' [1]):
  - in
  - for
  - on

- from
- to
- Utilizing or improving on existing lexical ontologies in creating new features
  - Wordnet,
  - Nomlex,
  - NomBank,
  - Propbank ...