Homework 2: Parsing and Machine Learning

COMS W4705_001: Natural Language Processing
Prof. Kathleen McKeown, Fall 2017
Due: Saturday, October 14th, 2017, 2:00 PM

This assignment will consist of tasks in unlabeled dependency parsing. You are required to complete this assignment in python 3.

For this assignment there will be dedicated Piazza posts pinned for each of the subtopics - Submission Logistics, Part 1, Part 2, Provided Code, and Parsing Theory and Terminology. To post a question, identify the topic it falls under and post a follow-up on that post. If a similar question has been asked but you require more information, post a reply in that thread.

For questions that pertain to your implementation specifics you can post a private question.

Part 1 - Arc Standard Parser

In this part of the assignment you are given a corpus of sentences in different genres. Your task is to complete a machine learning program that can *learn* a parser for the corpus. Note that typical learned parsers available in the NLP community are learned from the Penn Treebank which includes Wall Street Journal text and thus, these parsers work well on formal language, but not on text that we typically find on the web. Your learned parser will be tuned for sentences that are typically found on the web.

You will be provided with a parsing and machine learning framework that does part of the job for you. We will be using SVM, and all parameters have already been set for you. The parsing framework takes the training data as input and from this will learn a grammar and parser. What is missing in the parsing framework is the code to compute the parser's transitions and the features that the machine learner uses. You thus have two tasks: 1) write the code that can compute transitions and 2) experiment with different possible features so that you can get the best results on the dev set.

Corpus: Web TreeBank - a corpus of text from 5 different genres from the web containing 16,622 sentences. The 5 genres are weblogs, newsgroups, emails, reviews, and Yahoo! Answers. The data is in the CoNLL-X format (See appendix). There are 3 datasets: train, dev and test. You are provided with the train and dev sets. You will be evaluated on the test set, which does not have any genre information.

Provided Code:
Look at the file for each module in order to see what it does. You should focus on 4 of these, which you will have to modify. The remainder are given just so that you have some idea of how the whole program works, but you don’t need to understand each one.

```
hw2/
      README.txt
      src/
            dataset.py
            dependency.py
            domain_adaptation.py
            domain_adaptation_eval.py
            en-ud-dev.conllu
            en-ud-train.conllu
            eval.py
            feature_extraction.py
            feature_extraction_da.py
            train.py
            utils.py
            validate_transitions.py
```

In part 1, you will have to make changes only in 2 files: **dependency.py** and **feature_extraction.py**. When completed, answer the first 5 questions in README.txt. **Only these 2 files along with your answers will be used for grading this part of your assignment. Do not modify any other files.**

Instructions:

1. Download a copy of the code into your w4705 repo. You should see all the files mentioned above. In your repo, the folder structure should look like this:
```
w4705-01_<uni>/
                  hw2/
                        src/
```

2. Complete the ‘shift’, ‘right_arc’, and ‘left_arc’ functions in dependency.py.
   The stack, buffer and dgraph are given as arguments to each function.
   Implement the code that performs the operation and updates the stack, buffer and dgraph accordingly. See appendix for arc-standard example.
   a. Input - stack, buffer, dgraph
      i. Stack - List of word indices into the input sentence..
         Ex: `[0,3,4,6]` with an input sentence of “John went to the NLP class on Tuesday.” The tokens “ROOT”, “to”, “the”, “class” are on the stack.
      ii. Buffer - List of word indexes. It will be in decreasing order.
         Ex: `[7,5].` “On” and “NLP” are in the buffer.

        iii. Dgraph - List of arcs till now.
        Ex: `[(1,2),(2,3)].` There is an arc from “John” to “went” and another arc from “went” to “to”.

The tuple of (stack, buffer, dgraph) will be referred to as the configuration from here on.

3. Complete the ‘oracle_std’ function in dependency.py.
   The oracle_std predicts the next transition to be made given the current stack, buffer, dgraph and the gold_arcs. You should be implementing the arc-standard algorithm here.
    a. Input - stack, buffer, dgraph, gold_arcs
        i. Stack - List of word indexes. Ex: `[0,3,4,6]`
        ii. Buffer - List of word indexes . Ex: `[5]`
        iii. Dgraph - List of arcs till now. Ex: `[(1,2),(2,3)]`
        iv. Gold_arcs - List of arcs from the training data. Ex: `[(1,2),(2,3)(3,4)(5,0)(6,5)]`
    b. Output - transition
        i. Transition - string. One of `[‘shift’, ‘right-arc’, ‘left-arc’]`

4. Run ‘python3 validate_transitions.py’
   validate_transitions.py will pick a valid projective tree from ‘en-ud-train.conllu’. It will output the gold arcs and the set of transitions according to your implementation. Use this output to validate your implementation of ‘transitions.py’.
   It is important that you have the right implementation for the ML output to be correct. See Appendix for a run through of an example.

5. Run ‘python3 eval.py’.
   This script outputs 3 things.
    a. Precision - Number of correct predicted arcs/Number of predicted arcs.
    b. Recall - Number of correct predicted arcs/Number of gold arcs.
    c. F1 - Harmonic mean of precision and recall.

   As you can see our ML model isn’t doing too well with predicting the arcs. Looks like we need more features.

6. Add feature types in ‘get_features’ in ‘feature_extraction.py’.
   Add sufficient feature types to achieve good results. Your score will be evaluated relative to other students’ scores.
    a. Input - config, sentence_dictionary
        i. config - tuple of (stack, buffer, dgraph)
        ii. sentence_dictionary - dictionary of lists where the key is a CoNLL field header, and the list (value) is the CoNLL field values for the sentence. (See appendix)
    b. Output - list of string features
       `[‘Feature_X_val1’ , ‘Feature_Y_val1’, ‘Feature_X_val2’` ……]

For this you are required to develop an intuition on what information is required to learn transitions and produce the dependency arcs. Add your own feature types and label them accordingly. Append your feature types to the features list.

For example, the actual text (“form”) of the word on top of the stack may be informative. This feature type has been implemented for you as a sample. The feature type here is ‘top of the stack wordform’. We label it is as ‘TOP_STK_FORM_<FORM_VAL>’.

Feature Type Example:
In the input below, config includes stack, buffer and arcs in that order. Sentence_dict is a representation of the input and the gold arcs. It includes the POS tag sequence, the actual words in the sentence, the text as a list (FORM), the head of each word in the list (HEAD) (e.g., the head of “watch” is 0 which is the root and the head of “airlift” is “watch”), the coarse-grained parts-of-speech (CPOSTAG), and the lemmas for each word in the sentence (LEMMA).

```
Input - config : ([0,1],[3,2],[])
      sentence_dict: {
                       'POSTAG': [‘ROOT’,'VB', 'DT', 'NN'],
                       'TEXT': ' Watch the airlift\n',
                       'FORM': [‘ROOT’,'Watch', 'the', 'airlift'],
                       'HEAD': [‘ROOT’,'0', '3', '1'],
                       'CPOSTAG': [‘ROOT’,'VERB', 'DET', 'NOUN'],
                       'LEMMA':    [‘ROOT’,    'watch',    'the',
                       'airlift'],
                      }
Output - [‘TOP_STK_FORM_watch’]
```

NOTE - Do not use HEAD while creating new feature types. HEAD is the index of a token’s parent in the dependency tree, which we would be trying to predict.

We call it a feature type as it translates into a number of boolean features, and each input configuration can have only one of them set to true. Each feature type will increase your feature vector size by the number of possible values the type can have. (E.g., the above example will increase your feature vector size by the number of forms that can possibly be on top of the stack--approximately the vocabulary size.)

7. Run ‘python3 eval.py’ again.
   To see how your additional features are doing, run evaluate.py.

8. For the first 5 questions in your README.txt, note down your answers. You will submit these answers as a pdf on Courseworks. The questions are repeated below for your reference.
   a. Output of your validate_transitions.py
   b. List all the feature types you implemented. [Ex: ‘TOP_STK_TOKEN_’ , ‘...’, ‘....’] For our reference.
   c. Explain any 6 feature types in detail. What does the feature type represent? Why do you think this is useful for transition predictions?
   d. How do you interpret precision and recall in context of this task?
   e. What is your final F1 score on the dev set?
   You should report your final F1 score on the dev set. Since you will be evaluated on your F1 score on the test set, the scores you report are a reference. We will be running your code separately for grading.

Deliverables:

- **Completed dependency.py (4 functions).**
- **Additional feature types in ‘feature_extraction.py’.**
- **Answers to first 5 questions in the README.txt.**

Grading:

dependency.py	Shift	3
	Right Arc	6
	Left Arc	6
	Oracle standard	10
README questions	1. Validate_transitions output	4
	3. Explanation of 6 feature types	6*2
	4. Interpretation of precision and recall	2+2
F1 score (test)		25
Total		70

Part 2 - Domain Adaptation

For this part of the assignment you will use the learned parser from Part 1 and carry out experiments to learn how well a parser trained on one domain performs on another. You will modify the training set to include text just from the newsgroups genre and test how well it performs on weblogs and Yahoo Answers!.

Note that some features that you used for Part 1 will generalize well from one domain to another and some will not. You are also asked to experiment with different features to find which ones do perform well across domains.

In part 2, you will have to make changes only in 2 files: **domain_adaptation.py** and **feature_extraction_da.py**. When completed, note answers to the 2 questions in README.txt. **Only these 2 files along with your answers will be used for grading this part of your assignment. Do not modify any other files.** However, you are required to go through 'domain_adaptation_eval.py' to understand how the evaluation will run.

Instructions:

1. Modify 'domain_adaptation.py' and 'feature_extraction_da.py' to run your domain adaptation experiments described above.
   domain_adaptation.py is expected to write the files 'model_da.pkl' and 'feature_dict_da.pkl' that will be called for evaluation. You can reference the provided code for your implementation.

2. Run 'python3 domain_adaptation_eval.py'.
   The script will output your score on each on the test genres.

3. For the questions 6,7 under the heading 'Part 2 - Domain adaptation' in README.txt, note down your answers. Again, these answers are to be submitted on Courseworks. The questions are repeated below for your reference.
   a. Average F1 score from 10 runs of domain_adaptation_eval.py.
      As it is a smaller training set you **may** notice varying F1 scores. Write the average score over 10 runs. The scores you report are a reference. We will be running your code separately for grading.
      Format for each line:
      'train_genre : <train_genre>, test_genre : <test_genre>, Avg F1 : <F1_score>'
   b. Provide an explanation of the performance of the feature types in domain adaptation. What features generalized well? What features did not?

Deliverables:

- **Completed 'domain_adaptation.py'.**
- **Completed 'feature_extraction_da.py'.**
- **Answers to questions 6,7 in the README.txt.**

Grading:

README question	7. Domain adaptation explanation	10
Avg F1 score (dev)		20
Total		30

Code Submission

Run the following commands to commit and tag your homework.

```
a. git add dependency.py
b. git add feature_extractor.py
c. git add domain_adaptation.py
d. git add feature_extractor_da.py
e. git commit -m “<your comment>”
f. git tag -m "I finished HW2" done_HW2
g. git push origin master
h. git push origin done_HW2
```

Written Submission

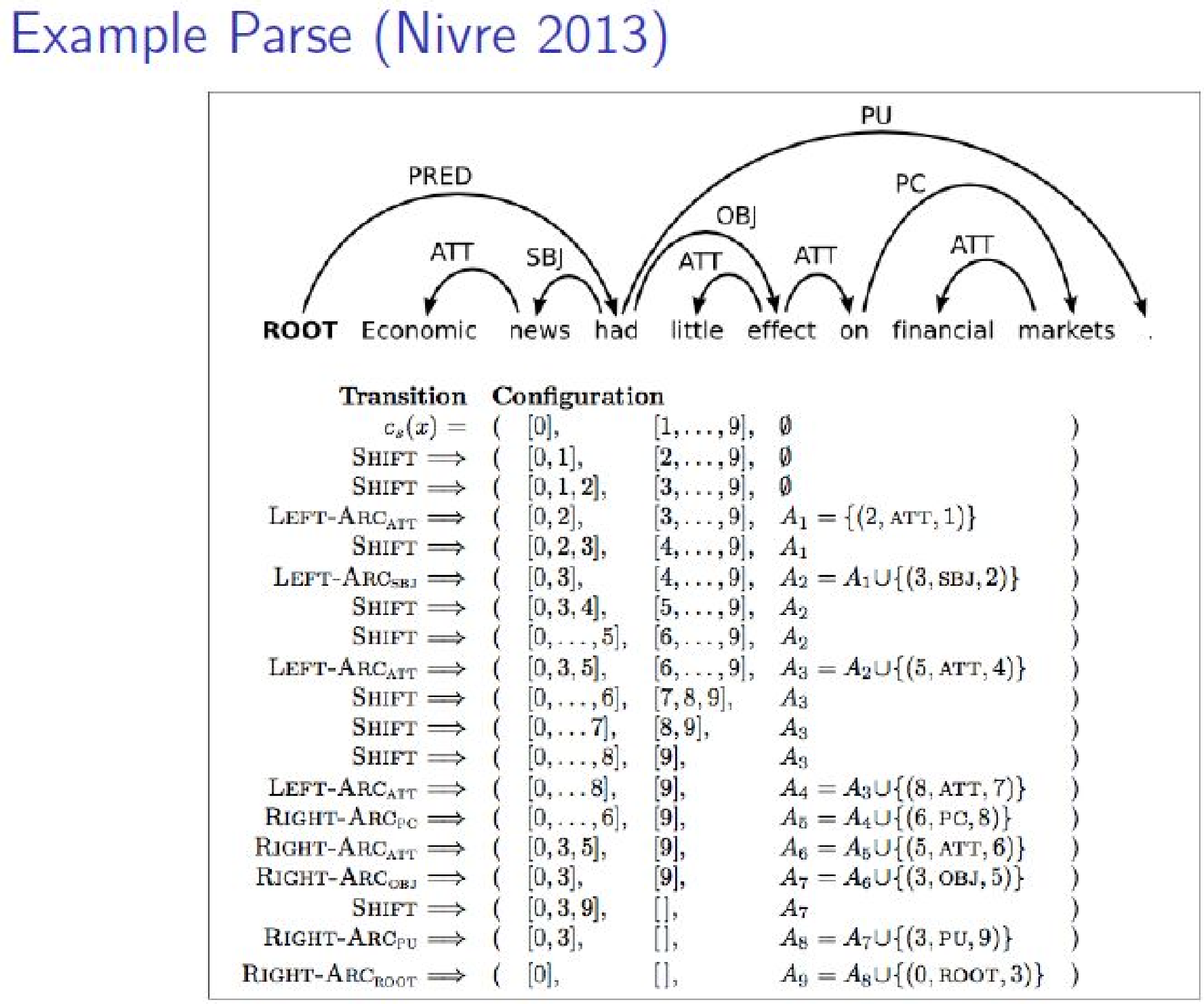
Compile your answers along with the 7 questions into a file called ‘w4705_01_hw2_<uni>.pdf’. Make sure to note your name and uni on the first page.

Submit your answers as a pdf on Courseworks.

APPENDIX

Arc-Standard example

The below example uses the arc-standard algorithm to produce the dependency parse. For this assignment you will not be looking to learn the label/relation of the arcs. The arc set will be (i,j) instead of (i,l,j), where i and j are word indices and l is a label.



Example Parse (Nivre 2013)

Transition	Configuration			
$c_s(x) =$	$(\; [0],$	$[1,\ldots,9],$	$\emptyset$	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,1],$	$[2,\ldots,9],$	$\emptyset$	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,1,2],$	$[3,\ldots,9],$	$\emptyset$	$)$
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	$(\; [0,2],$	$[3,\ldots,9],$	$A_1 = \{(2,\textsc{att},1)\}$	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,2,3],$	$[4,\ldots,9],$	A_1	$)$
$\textsc{Left-Arc}_{\textsc{sbj}} \Longrightarrow$	$(\; [0,3],$	$[4,\ldots,9],$	$A_2 = A_1 \cup \{(3,\textsc{sbj},2)\}$	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,3,4],$	$[5,\ldots,9],$	A_2	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,\ldots,5],$	$[6,\ldots,9],$	A_2	$)$
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	$(\; [0,3,5],$	$[6,\ldots,9],$	$A_3 = A_2 \cup \{(5,\textsc{att},4)\}$	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,\ldots,6],$	$[7,8,9],$	A_3	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,\ldots 7],$	$[8,9],$	A_3	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,\ldots,8],$	$[9],$	A_3	$)$
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	$(\; [0,\ldots 8],$	$[9],$	$A_4 = A_3 \cup \{(8,\textsc{att},7)\}$	$)$
$\textsc{Right-Arc}_{\textsc{pc}} \Longrightarrow$	$(\; [0,\ldots,6],$	$[9],$	$A_5 = A_4 \cup \{(6,\textsc{pc},8)\}$	$)$
$\textsc{Right-Arc}_{\textsc{att}} \Longrightarrow$	$(\; [0,3,5],$	$[9],$	$A_6 = A_5 \cup \{(5,\textsc{att},6)\}$	$)$
$\textsc{Right-Arc}_{\textsc{obj}} \Longrightarrow$	$(\; [0,3],$	$[9],$	$A_7 = A_6 \cup \{(3,\textsc{obj},5)\}$	$)$
$\textsc{Shift} \Longrightarrow$	$(\; [0,3,9],$	$[\,],$	A_7	$)$
$\textsc{Right-Arc}_{\textsc{pu}} \Longrightarrow$	$(\; [0,3],$	$[\,],$	$A_8 = A_7 \cup \{(3,\textsc{pu},9)\}$	$)$
$\textsc{Right-Arc}_{\textsc{root}} \Longrightarrow$	$(\; [0],$	$[\,],$	$A_9 = A_8 \cup \{(0,\textsc{root},3)\}$	$)$

CoNLL data format

Dependency parsing is a well-known problem in Natural Language Processing. It was the shared task of CoNLL, a conference for evaluating machine learning algorithms, for two consecutive years, and provides a fun introduction to more complex parsing algorithms.

The CoNLL data format is a tab-separated text file, where the ten fields are:
1) ID - a token counter, which restarts at 1 for each new sentence
2) FORM - the word form, or a punctuation symbol.
3) LEMMA - the lemma or the stem of the word form, or an underscore if this is not available
4) CPOSTAG - course-grained part-of-speech tag
5) POSTAG - fine-grained part-of-speech tag
6) FEATS - unordered set of additional syntactic features, separated by |
7) HEAD - the head of the current token, either an ID or 0 if the token links to the root node. The data is not guaranteed to be projective, so multiple HEADs may be 0.
8) DEPREL - the type of dependency relation to the HEAD. The set of dependency relations depends on the treebank.
9) PHEAD - the projective head of the current token. PHEAD/PDEPREL are available for some data sets, and are guaranteed to be projective. If not available, they will be underscores.
10) PDEPREL - the dependency relationship to the PHEAD, if available.

DEPREL, FEATS, PHEAD and PDEPREL are not applicable for our assignment.

This data is stored as a dictionary of label to list of values for every sentence and made available for training.

Data Format:

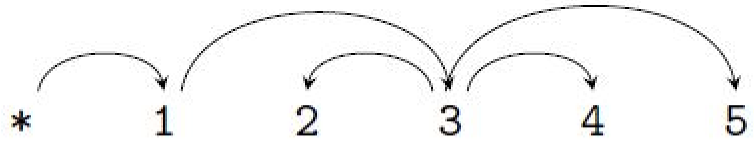
```
     sentence_dict : {
                     'POSTAG': [‘ROOT’,'VB', 'DT', 'NN'],
                     'TEXT': ' Watch the airlift\n',
                     'FORM': [‘ROOT’,'Watch', 'the', 'airlift'],
                     'HEAD': [‘ROOT’,'0', '3', '1'],
                     'CPOSTAG': [‘ROOT’,'VERB', 'DET', 'NOUN'],
                     'LEMMA': [‘ROOT’, 'watch', 'the', 'airlift'],
                    }
```

Projective Trees

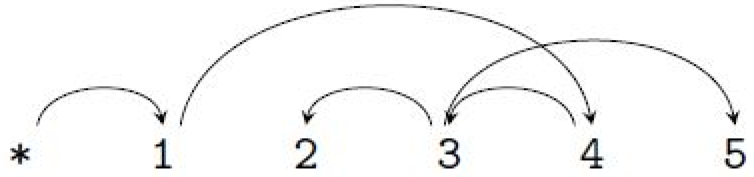
Arc-standard can parse only projective dependency trees. We filter the sentences in the train data and use only the projective trees for our training and testing.

A valid dependency tree is projective if for every arc (i,j) there is a path from i to k for all i < k < j. Here i,k,j are word indices.

Example of a projective dependency tree -



Example of a non-projective but valid dependency tree -



Precision+Recall

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

Visualizing the CoNLL dependency tree

While working on features you can use the MatlParser jar to visualize the CoNLL data. This would make it easy to think about dependency parsing in general and also the features.

If you are using Google Cloud resources for this assignment then X-forwarding, which you set up in Homework 0, and java is required for this to work. You will have to clean the data and remove some examples to get this working. The MaltParser is **not** required for completion of the assignment, but it is a useful tool.

Usage: `java -jar MaltEval.jar -g en-ud-train_clean.conllu -v 1`