Natural Language Processing - Ex4

Please submit a single zip file.
The zip file should contain your code, a README txt file,
a single pdf file for the reported results and any additional file required for running your code

Due: 16.1.20 23:55

In this Python programming exercise, we will implement the MST (Maximum Spanning Tree) parser for *unlabeled* dependency parsing, using the perceptron algorithm.

1. Use the NLTK toolkit for importing the *dependency_treebank* (using the commands: *nltk.download()* and *from nltk.corpus import dependency_treebank*). Load the parsed sentences of the corpus (given by dependency_treebank.parsed_sents()). Then, divide the obtained corpus into training set and test set such that the test set is formed by the last 10% of the sentences.

2. **The feature function:**
   Assume the input sentence is $s = \{w_1, ..., w_n\} \in S$ (S is the set of possible sentences), so the nodes of the parse tree are $V = \{w_1, ..., w_n, ROOT\}$. Write a Boolean feature function $f : V^2 \times S \to \{0,1\}^d$ that encodes the following features:

   - **Word bigrams:** For a potential edge between the nodes $u, v \in V$, the feature function will have a feature for every pair of word forms (types) w, w', which has a value of 1 if the node u is the word w and the node v is the word w'.
   - **POS bigrams:** For a potential edge between the nodes $u, v \in V$, the feature function will have a feature for every pair of POS tags t, t', which has a value of 1 if the node u has the POS tag t and the node v has the POS tag t'.

   Remark: The *ROOT* node can be assumed to have the POS tag *ROOT*.

3. **The perceptron algorithm:**
   The scoring function is defined to be the dot product of the feature function by a weight vector w. Implement the averaged perceptron algorithm for learning w from the training set. Use 2 iterations (i.e., two traversals over the examples) and a learning rate equal to 1. Traverse the training instances in a random order to avoid artefacts.

   For Inference (computing the MST), use the Chu-Liu-Edmonds algorithm. You can use the attached python implementation - *Chu_Liu_Edmonds_algorithm.py*, note that this version is for **minimum** spanning tree.

   For the feature function components based on POS tags, use the part of speech tags given in the test set (no need to run a PoS tagger).

4. **Distance features:** Augment the feature function so that it has another feature, such that given $(u, v) \in V$, has a value of 1 if the node u immediately precedes the node v in the sentence, 0 otherwise. Add similar features for the case where there is one word between u and v, where there are two words between them, and where there are three or more words between them.
   Note: The distance features are not symmetric. $f(u, v, s)$ is not necessarily equal to $f(v, u, s)$.
   Remark: The size of this feature vector should be the size of the original vector + 4.

5. **Evaluation:** For each of the two feature functions you created, compute the (unlabeled) attachment score for the learned w (i.e., the number of edges shared by the predicted tree and the gold standard tree divided by the number of words; see lecture notes), averaged over all sentences in the test set. Report your results in the pdf file.

6. (5 pts) **Bonus Question:** Design an additional set of features for the MST parser and show that the model with the additional features achieves better attachment score. Describe your choice of features in the pdf file.