COMP4650/6490 Document Analysis Assignment 3 – NLP

For this assignment, you will implement several common NLP algorithms: probabilistic context free grammar learning, dependency parsing, and relation extraction in Python.

Throughout this assignment you will make changes to provided code to improve or complete existing models. In addition, you will produce an answers file with your responses to each question. Your answers file must be a .pdf file named u1234567.pdf where u1234567 is your Uni ID. Some questions also require you to submit an output file in a format specified by the question. You should submit a .zip file containing **all the code files**, your **answers pdf** file, and these **requested output files** but **BUT NO OTHER DATA.**

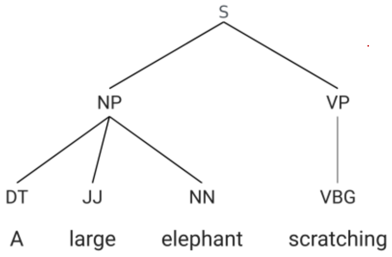
Your answers to coding questions (or coding parts of each question) will be marked based on the quality of your code (is it efficient, is it readable, is it extendable, is it correct). The output files requested will be used as part of the marking for the coding questions.

Your answers to discussion questions (or discussion parts of each question) will be marked based on how convincing your explanations are (are they sufficiently detailed, are they well-reasoned, are they backed by appropriate evidence, are they clear).

This is an individual assignment. Group work is not permitted. Assignments will be checked for similarities.

Question 1: Probabilistic Context Free Grammars (20%)

For this question you will be modifying the starter code in *syntax_parser.py*. You have been provided with a training dataset containing 73638 English language sentences, which have been parsed into constituency trees (*parsed_sents_list.json*). For, example given the sentence: *"A large elephant scratching".* The parse tree is:



For this structure the parsed sentence in *parsed_sents_list.json* is *['S', ['NP', ['DT', 'a'], ['JJ', 'large'], ['NN', 'elephant']], ['VP', ['VBG', 'scratching']]]*. Note that *parsed_sents_list.json* contains a list of parses, each for a different sentence.

Your task is to estimate a probabilistic context free grammar PCFG given this dataset. You should estimate the conditional probabilities of the rules in this grammar (including all rules that express facts about the lexicon) using the maximum-likelihood approach given in lectures. Some examples of rules in this grammar are:

S -> NP
VBN -> parked
ADVP -> IN NP
NP -> NP , NP

Note that some terminals and non-terminals are punctuation characters. There are many more rules than those listed here and you will first have to identify all the rules. All terminals are lower case or punctuation.

When your *syntax_parser.py* code is run, write a list of all the transition rules and their **conditional probabilities** to a file called "q1.json". There is a class called *RuleWriter* in *syntax_parser.py* which will write the rules in the correct format for the automated judging system (do not modify this class). The expected format is (example of format only not actual answers):

[["NP", ["PP", "NP"], 0.8731919455731331], ["NNS", ["locations"], 0.062349802638746935], ….
<many more rules>]

Make sure to submit "q1.json" for marking. Also make sure to **submit your code.**

Hint: if you create a test input file containing only [["A", ["B", ["C", "blue"]], ["B", "cat"]]] your program should output (note: the order the rules are listed in the output does not matter):
[["A", ["B", "B"], 1.0], ["B", ["C"], 0.5], ["B", ["cat"], 0.5], ["C", ["blue"], 1.0]]

Question 2: Dependency Parsing (40%)

Complete both part A and part B of this question.

Part A

The file *dependency_parser.py* contains a partial implementation of Nivre's arc-eager dependency parsing algorithm. Your first task is to implement the missing *left_arc* method, *reduce* method*,* *right_arc* method, and *parse* function. Your implementation should follow the transitions as given by the lecture slides (also some additional instructions are provided as comments in *dependency_parser.py)*. Make sure that your class methods and parse function return True if the operation(s) was/were successfully applied, or else False. The *parse* function should return True if it generates a dependency forest. Your implementation **must** make use of the *DepContext* class which stores the current state of the parser. **Do not modify the *DepContext* class or the name and parameters of the methods and functions you have been asked to implement** because part A of this question will be automatically marked. The auto-marker will inspect the internal state of DepContext to confirm that operations have been applied correctly. You should not use any additional imports for this question (except from python standard libraries). The judging system will use python 3.8.10 in a sandboxed Linux environment. In your **answers pdf** file include the output of *dependency_parser.py* (it runs a small number of test cases and prints them to the terminal). Make sure to submit your **code**.

Hint: As mentioned in lectures we need to make sure that words are uniquely represented (in case there are multiple copies of the same word), to do this we use the position of the word in the sequence, with the [ROOT] token prepended to the sequence. This is handled by DepContext.

Part B

Your second task is to implement the standard oracle for arc-eager dependency parsing in *oracle_parser.py* (specifically you should implement the *parse_gold* function). Make use of the parsing class you developed in Part A of this question -- do not modify your parsing class from part A, all code for this second part should be put in the *oracle_parser.py*. The parsing class from part A has been imported. Additional imports are not permitted (except from python standard libraries). You may add helper functions in *oracle_parser.py.* The oracle you need to implement determines which action to be performed in each state to reconstruct the ground truth dependency parse tree. It returns the list of actions, and the states in which they occur – though in this case you should return only features of the state (using the provided extract_features function -- do not modify the feature extraction code). The details of this algorithm were not covered in lectures, instead you should implement **Algorithm 1** from this paper: https://aclanthology.org/C12-1059.pdf (while this paper covers different oracle methods including a dynamic oracle, you should only implement **Algorithm 1** on page 963). You should implement un-labelled dependency parsing so you will need to modify this algorithm to work without labels. To implement Algorithm 1 you will need to make use of the parsing class you developed in Part A of this question.

Once you have implemented the *parse_gold* function, run *oracle_parser.py* and it will determine the actions that could generate the ground truth dependency parse trees in *gold_parse.json* under the rule set in *rule_set.json*. It will print the first 3 cases where the oracle could reproduce the ground truth, the first 3 cases where it could not reproduce the ground truth, and after a short wait the number of total failure and total success cases. In your **answers pdf** file include the output of *oracle_parser.py* and **explain why** it failed in the three printed cases.

Make sure to submit all your **code** for both parts of this question.

Question 3: Relation Extraction (40%)

Your task is to design and implement a relation extraction model along with its training and testing pipeline. Specifically, your model should extract the relation 'nationality' (denoted /people/person/nationality in our dataset) which holds between a PERSON and a GEOPOLITICAL ENTITY. For example /people/person/nationality(Yann LeCun, France) , /people/person/nationality(Julie Bishop, Australia). The training data is provided in (*sents_parsed_train.json*) the testing data is provided in (*sents_parsed_test.json*). The test data does not have the ground truth set of relations; you program must extract them. Some starter code has been provided (*relextract.py*) which shows you how to read in the data, access its various fields, and write a valid output file.

The final goal is to **extract all relations** of the type *nationality* that exist in the **test data**, not to simply identify if a relation type is in each sentence. Your output should be a list of all the relations in the test data, in CSV format (you should name it q3.csv) -- the starter code provides a function to write this file. The order of lines in this csv does not matter, and duplicate lines will be removed before scoring. You will be scored based of F1. An example judging script (eval_output.py) is provided for you. Since you have not been given the ground truth for the test data you can only run this script using the training data. To do this run (python eval_output.py --train-set). Note that your final output will be evaluated against the test set by the examiner. Thus, make sure that you submit your **output file** named **q3.csv and make sure it contains the output from the test data**.

It is very important that your relation extraction algorithm work for entities **not seen during training**. To clarify, this means that your model should be designed so that it can identify relations such as /people/person/nationality(Julie Bishop, Australia) even if "Julie Bishop" and "Australia" entities are not in the training data. None of the PERSON, GPE pairs that constitute a nationality relation occur in both the training dataset and the test dataset.

The focus of this question is to develop a good training/validation pipeline and to explore different lexical, syntactic or semantic features. You should use the 'Feature-based supervised relation classifiers' technique introduced in the Speech and Language Processing textbook (3rd edition draft 30th December 2020) Section 17.2.2 -- and figure 17.6. You do not have to implement all the suggested features, but an attempt to implement a variety of them is expected. You will likely need to do substantial pre-processing of the input data as well as some post processing of the output list to get a good result. You should use LogisticRegression from sklearn as the classifier. **Neural networks and vector features generated by neural networks are not to be used for this question.** You may use SpaCy to extract additional features from your text; however, several features are already provided for you. It is possible to get full marks using only the features provided; however, these features will likely need some processing before they can be used appropriately as part of a classification pipeline.

Permitted Libraries: sklearn, numpy, pandas, python standard libraries, nltk, spacy, json.
You should **not** use other libraries, datasets, or ontological resources. Except for the provided country_names.txt.

Answer the following questions in your **answers pdf file** (please answer in plain English, dot points are preferred, do not include code snippets or screenshots of code as these will not be marked):

1. What pre-processing did you do?

2. How did you split the data into training and validation sets?
3. How did you choose hyperparameters?
4. What features did you choose and why did you choose these features?
5. What post-processing did you do?

Submit your **code**. Submit your **output file** named **q3.csv** (**make sure this is the output from the test data**). Submit **your answers** to the questions.

Marking:

Part of the mark for this question will be based on the f1 score of your q3.csv file on the held-out test data. The rest will be based on an examination of your code and answers to the questions in your answers pdf file.

Description of the data:

The data you have been given is described in detail below. Briefly, it is a set of sentences, each of which contain a relation between two entities. You may use the relations that are not the target relation "/people/person/nationality" to generate negative examples on which to train your model.

sents_parsed_train.json and *sents_parsed_test.json* each contain a list of dictionaries. Each dictionary corresponds to a different example.

Each example dictionary has the following fields:

"tokens": a tokenised sentence that contains a relation

"entities": a list of the named entities in the sentence

"relation": a description of the relation in the sentence

Each *entity* is a dictionary with the following fields:

"start": the token index where the entity starts

"end": the token index where the entity ends

"label": the entity type. PERSON = a persons, GPE = a geopolitical entity, and several others

Each *relation* is a dictionary with the following fields:
Note: the relation field has been removed from *sents_parsed_test.json*

"a": first entity that is part of the relationship (tokenized into a list)

"b": the second entity that is part of the relationship (tokenized into a list)

"a_start": the token index where the first entity starts

"b_start": the token index where the second entity starts

"relation": the type of relation, note we are interested in the "/people/person/nationality" relation for this assignment.

There are also some **additional features** that may or may not be helpful (each of the features below is a list, the array indices match to those of the tokenized sentence):

"lemma": the tokenized text that has been lemmatized.

“pos”: part of speech tags for each word

“dep”: the dependency relationships

“dep_head”: the head of each dependency relationship (allowing you to reconstruct the full dependency tree)

“shape”: shape features for each word. This preserves things such as word length and capitalization while removing much of the content of the word.

“isalpha”: is each word only alphabetic.

“isstop”: is each word in the spacy stop word dictionary (and thus likely to be common)

A file containing a list of country names (country_names.txt) is also provided for you to use.

All features were extracted using SpaCy 2.3.5 en_core_web_md models. The data you have been provided was constructed automatically by matching ground truth relations to text. For the training data any sentence that had both the named entities that participated in a ground truth relation was marked as containing that relation. No manual checks were done to ensure accuracy. You might like to consider how this will affect the data quality.

您所提供的数据是通过将ground truth relations与text进行匹配而自动构建的。对于训练数据，任何同时包含两个参与ground truth relations的命名实体的句子都被标记为包含该关系。没有人为检查以确保准确性。您可能想要考虑这将如何影响数据质量