# Question 2: Dependency parsing

## Part A:

Return value: True

Dependencies: [(0, 1), (1, 2)]

Return value: True

Dependencies: [(2, 1), (3, 2), (0, 3)]

Return value: True

Dependencies: [(0, 1), (1, 2), (2, 3)]

Return value: True

Dependencies: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]

Return value: True

Dependencies: [(2, 1), (0, 2)]

## Part B:

Action: ['right', 'right']

Features: [['[root]\_stack', 'a\_buffer'], ['a\_stack', 'b\_buffer']]

Success case 1

Sentence: [ROOT] A man dressed in a tux holds a violin .

Actions: ['shift', 'left', 'shift', 'right', 'right', 'shift', 'left', 'right', 'reduce', 'reduce', 'reduce', 'left', 'right', 'shift', 'left', 'right', 'reduce', 'right']

Features: [['[root]\_stack', 'a\_buffer'], ['a\_stack', 'man\_buffer'], ['[root]\_stack', 'man\_buffer']] ....

--------

Success case 2

Sentence: [ROOT] A small cheese pizza is hot on the pan .

Actions: ['shift', 'shift', 'shift', 'left', 'left', 'left', 'shift', 'left', 'right', 'right', 'reduce', 'right', 'shift', 'left', 'right', 'reduce', 'reduce', 'right']

Features: [['[root]\_stack', 'a\_buffer'], ['a\_stack', 'small\_buffer'], ['small\_stack', 'cheese\_buffer']] ....

--------

Success case 3

Sentence: [ROOT] A woman at a restaurant sits in front of a finished plate .

Actions: ['shift', 'left', 'shift', 'right', 'shift', 'left', 'right', 'reduce', 'reduce', 'left', 'right', 'right', 'right', 'right', 'shift', 'shift', 'left', 'left', 'right', 'reduce', 'reduce', 'reduce', 'reduce', 'right']

Features: [['[root]\_stack', 'a\_buffer'], ['a\_stack', 'woman\_buffer'], ['[root]\_stack', 'woman\_buffer']] ....

--------

Fail case 1

['[ROOT]', 'A', 'doughnut', 'that', 'a', 'person', 'is', 'using', 'to', 'see', 'out', 'of', '.']

Actions: ['shift', 'left', 'right', 'shift', 'shift', 'left', 'shift', 'shift', 'left', 'left', 'shift', 'shift', 'left', 'right', 'right', 'right', 'reduce', 'reduce', 'reduce', 'shift']

Features: [['[root]\_stack', 'a\_buffer'], ['a\_stack', 'doughnut\_buffer'], ['[root]\_stack', 'doughnut\_buffer']] ....

Parser Dependencies:('[ROOT]', 'doughnut')('doughnut', 'A')('person', 'a')('using', 'person')('using', 'is')('using', 'see')('see', 'to')('see', 'out')('out', 'of')

Ground Truth Dependencies:('[ROOT]', 'doughnut')('doughnut', 'A')('doughnut', 'using')('doughnut', '.')('person', 'a')('using', 'person')('using', 'is')('using', 'see')('see', 'that')('see', 'to')('see', 'out')('out', 'of')

--------

Fail case 2

['[ROOT]', 'The', 'waters', 'are', 'very', 'choppy', 'today', 'which', 'is', 'making', 'it', 'hard', 'to', 'ride', '.']

Actions: ['shift', 'left', 'shift', 'left', 'right', 'shift', 'left', 'right', 'reduce', 'right', 'shift', 'shift', 'left', 'left', 'reduce', 'reduce', 'shift', 'shift', 'left', 'right', 'shift', 'left', 'right', 'reduce', 'reduce', 'shift']

Features: [['[root]\_stack', 'the\_buffer'], ['the\_stack', 'waters\_buffer'], ['[root]\_stack', 'waters\_buffer']] ....

Parser Dependencies:('[ROOT]', 'are')('waters', 'The')('are', 'waters')('are', 'choppy')('are', 'today')('choppy', 'very')('making', 'which')('making', 'is')('making', 'hard')('hard', 'it')('hard', 'ride')('ride', 'to')

Ground Truth Dependencies:('[ROOT]', 'are')('waters', 'The')('waters', 'making')('are', 'waters')('are', 'choppy')('are', 'today')('are', '.')('choppy', 'very')('making', 'which')('making', 'is')('making', 'hard')('hard', 'it')('hard', 'ride')('ride', 'to')

--------

Fail case 3

['[ROOT]', 'A', 'woman', 'holding', 'three', 'bags', 'stands', 'by', 'the', 'open', 'doors', 'of', 'a', 'subway', 'car', ',', 'through', 'which', 'other', 'people', 'can', 'be', 'seen', '.']

Actions: ['shift', 'left', 'shift', 'right', 'shift', 'left', 'right', 'reduce', 'reduce', 'left', 'right', 'right', 'shift', 'shift', 'left', 'left', 'right', 'right', 'shift', 'shift', 'left', 'left', 'right', 'reduce', 'reduce', 'reduce', 'reduce', 'right', 'shift', 'right', 'shift', 'left', 'shift', 'shift', 'shift', 'left', 'left', 'left', 'reduce', 'left', 'reduce', 'reduce', 'shift', 'shift']

Features: [['[root]\_stack', 'a\_buffer'], ['a\_stack', 'woman\_buffer'], ['[root]\_stack', 'woman\_buffer']] ....

Parser Dependencies:('[ROOT]', 'stands')('woman', 'A')('woman', 'holding')('holding', 'bags')('bags', 'three')('stands', 'woman')('stands', 'by')('stands', ',')('by', 'doors')('doors', 'the')('doors', 'open')('doors', 'of')('of', 'car')('car', 'a')('car', 'subway')('through', 'which')('people', 'other')('seen', 'through')('seen', 'people')('seen', 'can')('seen', 'be')

Ground Truth Dependencies:('[ROOT]', 'stands')('woman', 'A')('woman', 'holding')('woman', 'seen')('holding', 'bags')('bags', 'three')('stands', 'woman')('stands', 'by')('stands', ',')('stands', '.')('by', 'doors')('doors', 'the')('doors', 'open')('doors', 'of')('of', 'car')('car', 'a')('car', 'subway')('through', 'which')('people', 'other')('seen', 'through')('seen', 'people')('seen', 'can')('seen', 'be')

--------

Total fails: 9, Total success: 19991

**Reasons for Fail Cases:**

**Case 1:**

Three Functional Dependencies are missing compared with Ground Truth,

1. (‘see’, ‘that’)
2. (‘doughnut’, ‘.’)
3. (‘doughnut’, ‘using’)

In order to get the first two of those, we need (‘see’, ’using’) and (‘using’, ’that’) in our rules. But we can’t have both (‘see’, ’using’) and (‘using’, ’that’) as they might void the rule of being a target only for one entity. (‘see’, ’using’), (‘using’, ’that’), (‘see’, ‘that’) and (‘doughnut’, ‘using’) make using and that as target for more than one entity. Automatically, we infer that even (‘doughnut’, ‘.’) one isn’t possible as (‘see’, ‘that’) and (‘doughnut’, ‘using’) can’t be achieved.

**Case 2:**

Two Functional Dependencies are missing compared with Ground Truth,

1. (‘waters’, ’making’)
2. (‘are’, ’.’)

After (‘are’, ’waters’), ‘waters’ was popped out of the stack. So (‘waters’, ’making’) can’t be achieved. After 16th action which is a reduction operation, ‘are’ was popped out of the stack, so (‘are’, ’.’) can’t be achieved.

**Case 3:**

Two Functional Dependencies are missing compared with Ground Truth,

1. (‘woman’, ’seen’)
2. (‘stands’, ’.’)

(‘woman’, ’seen’) isn’t possible, ‘women’ is no more in the stack after the rule (‘stands’, ‘woman’). So, we cant generate this FD, and also (‘stands’, ‘woman’) is in the ground truth. The last two actions are shift and shift, means ‘seen’ and ‘a’ were pushed inside the stack. From this we can infer that, there must be a rule (‘.’ , ‘seen’) to achieve (‘stands’, ’.’). But if we have (‘.’ , ‘seen’), then we can’t have (‘woman’, ’seen’).

# Question 3: Relation Extraction

1. **What pre-processing did you do?**

As part of preprocessing the data before training the model, we did the following methods

* Trim: We trimmed the token sequence such that trimmed list contains the entities and the tokens between them. We also considered having 3 tokens on either side of entity included as it might be resourceful while using bigrams.
* Remove digits & Stop words: Using the dictionary provided which each token sentence, we removed words with digits and stop words from trimmed token list. This is based on an assumption that Person and GPE entities have a very less possibility of containing digits or stop words.
* Creating and Balancing data: For training the model, I considered pairs of entities and their relation. Two step filtering was followed while creating data, entity pair was considered is the entities were either PERSON and GPE. Considering there are nationality relationship between different entities, we are assuming that the number of pairs which such condition is negligible and it’s hard to consider such cases if we want the model to be robust. Second step of filtering was to reduce the negative example which are created by pairs which aren’t related by nationality. So, we considered the pairs as negative pairs if they are from a token sentence with whom I was able to generate only 5 entity pairs.

1. **How did you split the data into training and validation sets?**

The entire data was split into training and testing data set with a ration of 0.9:0.1. Using the GridSearchCV method, I performed cross validation over the data while training. Cross Validation was 3 fold and stratified to maintain the class distribution inside each split.

1. **How did you choose hyperparameters?**

Hyperparameters like C, max\_iteration was found using GridSearchCV method. A dictionary of parameters was provided along with the model, which then GridSearchCV uses to find the parameters with best score. This is a Automated search.

1. **What features did you choose and why did you choose these features?**

Entities and tokens in between and surrounding them:

Entity by itself doesn’t provide any value to the model. So we need to incorporate the context by considering the tokens around the entity. Feature Vectors were created using CountVectorizer. These feature vectors are binary representation of existence of a token in a token sequence. They may be bi-gram or tri-gram. Using n\_gram, I am able to infiltrate the context into the feature vector.

Finding the TFIDF Vector:

Instead of the CountVectorizer, I used TFIDF vectorizer. This doesn’t have better performance as frequency of tokens isn’t necessary or contributing to the performance.

1. **What post-processing did you do?**

Using the Country\_names.txt file provided, I read and stored the country names in a list. After prediction, we have a list of (PERSON, GPE) tuples. From these tuple, we will have to filter the ones if their GPE is in the list of countries that we read. This way we will reducing the number of false positives. Also, we filtered the relations with both GPE’s using the countries list.