Extra credit questions:

1. [Up to 5 points] Do feature engineering to further improve the accuracy. (You cannot change the classifier or dataset.)

I introduced four features for the parser:

1. Stack size - returns the current length of the stack per transition
2. Buffer size - returns the current length of the buffer per transition
3. Top stack word - returns the current top of the stack (word index) per transition
4. Top buffer word - returns the current top of the buffer (word index) per transition
5. Index - returns the input index (current offset of the word under consideration)
6. Index plus stack top - returns the addition of index and current top of the stack
7. Index plus buffer top - returns the addition of index and current top of the buffer
8. Sentence length - returns addition of the stack size and buffer size

Feature Engineering Process:

- I initially added Stack length and Buffer length features to the transitions, achieving an accuracy of 0.6338.
- Recognizing that the left_arc transition relies heavily on the top element of the stack, I introduced the Top stack word feature, which raised accuracy slightly to 0.6340. It might be because I am taking the top of the stack after popping the top element of the stack which is the element related to the left_arc transition.
- Similarly, understanding that right_arc depends heavily on the top element of the buffer, I added the Top buffer word feature, further improving accuracy to 0.6403 which was a slight increment. Here again, the minor gain may be because I am taking the top of the buffer after popping the top element of the buffer which is the element related to the right_arc transition.
- From these observations, I realized that in both left_arc and right_arc, we need to consider the actual word under processing which is the input index to the function. So, I added a new feature called Index, which returns the current offset of the word being considered. This addition resulted in a significant accuracy increase to 0.8832.
- These were my primary features. Building on them, I introduced derived features: Index plus stack top (boosting accuracy to 0.9093), Index plus buffer top (increasing accuracy to 0.9098), and Sentence length.
- However, adding Sentence length caused accuracy to drop to 0.9039, so I discarded it.
- After feature engineering, I had seven features in total, with a final accuracy of 0.9098.

Below is the accuracy matrix from this final configuration.

```
  ==> Training (25 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
            1          -1.09861        0.546
            2          -0.74879        0.631
            3          -0.64382        0.712
            4          -0.57112        0.753
            5          -0.51728        0.791
            6          -0.47558        0.828
            7          -0.44221        0.842
            8          -0.41486        0.856
            9          -0.39201        0.866
           10          -0.37263        0.877
           11          -0.35597        0.885
           12          -0.34149        0.891
           13          -0.32879        0.896
           14          -0.31755        0.900
           15          -0.30754        0.903
           16          -0.29855        0.905
           17          -0.29044        0.906
           18          -0.28307        0.908
           19          -0.27636        0.909
           20          -0.27021        0.910
           21          -0.26455        0.911
           22          -0.25932        0.912
           23          -0.25448        0.913
           24          -0.24997        0.914
        Final          -0.24577        0.915
Held-out Classification Accuracy: 0.9098
Held-out Attachment Accuracy:     0.6797
```

Please note that I have implemented the feature_extractor function for this task.

2. [Up to 5 points] Complete the code to compute attachment accuracy. This requires three things: implement the `attachment_accuracy` function, implement the `sentence_attachment_accuracy` function, and use the `classifier_transition_sequence` function to generate the transitions from the classifier. You might need to harden some code elsewhere to account for impossible / incomplete transitions.

Implemented the **sentence_attachment_accuracy, attachment_accuracy** functions.

3. [Up to 5 points] Do an error analysis to understand what the classifier is getting wrong. What types of transitions does it mispredict (i.e., what kinds of situations does it get confused by)?

I analyzed the percentages of mispredicted transitions relative to the total number of transitions. There are six types of misprediction scenarios, each with specific misclassification counts, ratios and ranks as shown in the table below. In the table:

- **Error count** represents the total number of errors in each scenario.
- **Error ratio** shows the percentage of errors relative to the total transitions.
- **Error rank** indicates the rank I assigned to each scenario based on error count.

Mispredicted scenario	Error count	Error ratio	Error rank
Predict shift as a left arc	10	0.00057	5
Predict shift as a right arc	7	0.00040	6
Predict left arc as a shift	66	0.00377	3
Predict left arc as a right arc	584	0.03339	2
Predict right arc as a shift	42	0.00240	4
Predict right arc as a left arc	869	0.04969	1

The analysis shows that most mispredictions stem from classifying a right arc as a left arc, and the second most is the vice versa. This suggests that the classifier struggles to distinguish between edge types, particularly between right and left arcs.

The third and fourth largest error sources are due to predicting both left arc and right arc transitions as shifts. Finally, the smallest error contributions come from shift transitions being misclassified as either left or right arcs.

Overall, the highlighted scenarios (rank 1 and 2) significantly contribute to the overall misclassification error.