1. The required performance was about 74%, which the program is able to reach on most runs. I chose to represent my perceptron weights as a numpy array, and extracted sentences into a Counter object. This way, we did not need to store any 0 feature counts. This algorithm was quite fast and was able to reach near 100% training accuracy each time.
2. The required performance was about 77%, which the program was able to reach as well. The implementation of the Logistic Regression model was quite similar to the perceptron, the only difference being that we used the logistic function on top of our $\mathbf{W}^T\mathbf{X}$ calculation. This algorithm was slower than the perceptron algorithm but resulted in slightly higher average accuracy.
3. When implementing the bigram feature extractor I expected an increase in performance due to being able to track the context of each word, but I found that this approach led to a decreased performance of around 71%. However, I noticed that the training accuracy was always 100%, so perhaps it seems the model overfit the training data.
4. For the better feature extractor, I tried implementing tf-idf to extract features. This required two passes through the data - once to generate the inverse document frequency (idf), and a second time to combine it with the term frequency (tf). After testing, the performance was around 78%.