

```

Train training = new Train(trainSentences, trainTags);

writeOutput(unseenPunishment, training, testSentences, outputLocation);

System.out.println(checkTest(outputLocation, testTags));

```

The Lines of code above create probability maps based on the files trainSentences and trainTags. These maps are then used to predict the part of speech (POS) corresponding to each word in the file testSentences. The predicted POS are then written into an output file called outputLocation. The accuracy of the predictions in this file are then tested against an answer key file called testTags.

### Test Results (POS Prediction Accuracy)

	Unseen punishment = -100	Unseen punishment = -75	Unseen punishment = -50	Unseen punishment = -25	Unseen punishment = -0
Test sentences = brown-test-sentences.txt	Viterbi correctly tagged 35109/36394 parts of speech (96.4691982 1948673% of words).	iterbi correctly tagged 35109/36394 parts of speech (96.4691982 1948673% of words).	Viterbi correctly tagged 35109/36394 parts of speech (96.4691982 1948673% of words).	Viterbi correctly tagged 35109/36394 parts of speech (96.4691982 1948673% of words).	Viterbi correctly tagged 2260/36394 parts of speech (6.20981480 4638127% of words).
Test sentences = simple-test-sentences.txt	Viterbi correctly tagged 32/37 parts of speech (86.4864864 8648648% of words).	Viterbi correctly tagged 32/37 parts of speech (86.4864864 8648648% of words).	Viterbi correctly tagged 32/37 parts of speech (86.4864864 8648648% of words).	Viterbi correctly tagged 32/37 parts of speech (86.4864864 8648648% of words).	Viterbi correctly tagged 9/37 parts of speech (24.3243243 24324323% of words).

### The Ideal Unseen Punishment

- Unseen Punishment is the probability used by the viterbi algorithm to approximate the likelihood that a word it is encountering is being used as a POS that it has never seen it be used as before.

The predictions of brown-test-sentences.txt reached their greatest accuracy with an Unseen Punishment of -16

- Accuracy: 35116/36394 correct predictions

The predictions of simple-test-sentences.txt reached their greatest accuracy with an Unseen Punishment of any number  $\leq -5$

- Accuracy: 32/37 correct predictions

Conclusions: The greater the size of the training file the smaller the ideal unseen punishment. This makes sense because the less words an algorithm has encountered in training, the more likely it is to encounter something new when taking input and vice versa.

### **Example Sentences:**

- The/DET quick/ADJ brown/NP fox/NP jumped/VD over/P the/DET log./N
- I/PRO ate/VD pasta/ADV for/P dinner./DET
- I/DET pet/N my/PRO dog./V

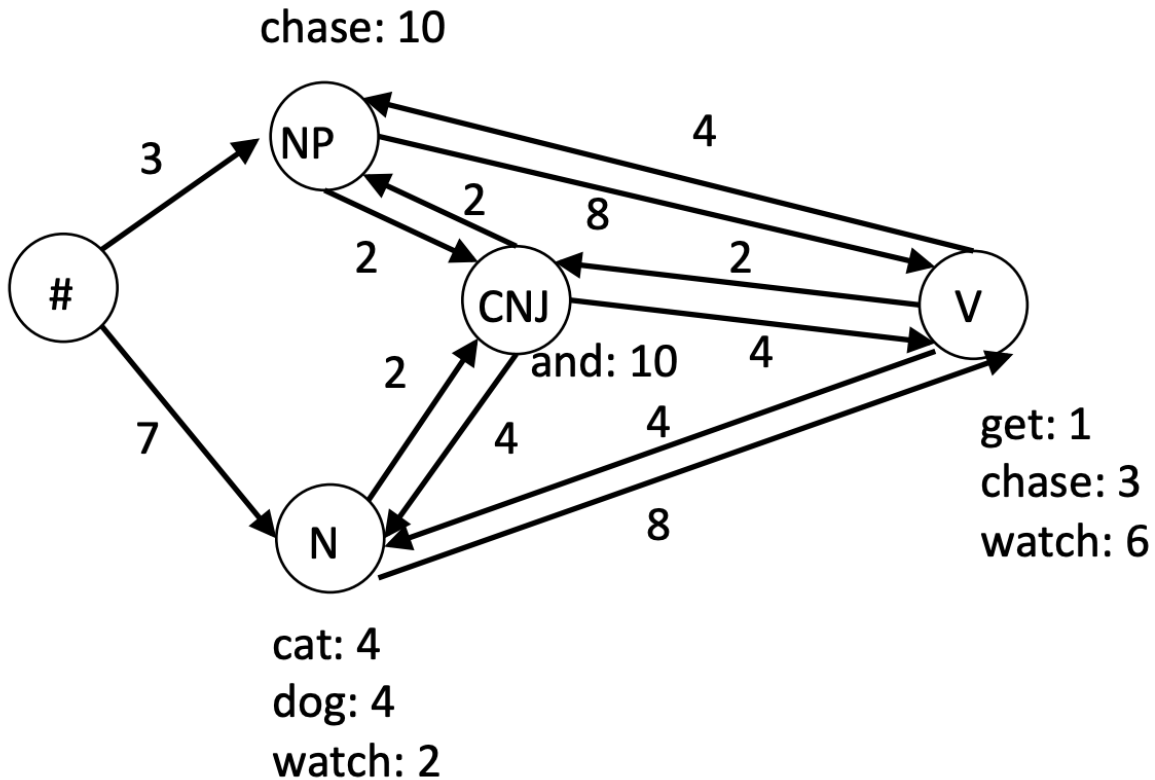
Here are a few example sentences with we tested our program with when it was trained by the brown files. As you can see, it is fairly accurate in tagging adjectives and pronouns, but often gets proper nouns and nouns mixed up and is otherwise not extremely accurate. This program is good at tagging the bulk of words but lacks precision for the minority.

### **Testing the training method:**

- To ensure the accuracy of our training and the resulting Emission and Transition Maps which both hold `<String<String, Double>>` we first made our maps hold only occurrences and not probabilities. We then tested our training method on small handmade input files and printed out our maps. We were then able to count by hand the number of occurrences of a word as a POS as well as the number of transitions from one POS to another and see that our method was collecting information from the input files accurately. We then added to our method and went item by item, changing occurrence counts into probabilities.

### **Hard-Coded Testing:**

For our hard-coded tests, we hard-coded the map we went over in drill into transmission and emission maps.



Emission Map:

{NP={chase=0.0}, V={watch=-0.5108256237659907, get=-2.3025850929940455, chase=-1.2039728043259361}, CNJ={and=0.0}, N={watch=-1.6094379124341003, cat=-0.916290731874155, dog=-0.916290731874155}}

Transmission Map:

{NP={V=-0.2231435513142097, CNJ=-1.6094379124341003}, #={NP=-1.2039728043259361, N=-0.35667494393873245}, V={NP=-0.916290731874155, N=-0.916290731874155, CNJ=-1.6094379124341003}, N={V=-0.2231435513142097, CNJ=-1.6094379124341003}, CNJ={NP=-1.6094379124341003, V=-0.916290731874155, N=-0.916290731874155}}

Because the data set here was very limited, we had low expectations for results. Although the viterbi algorithm was able to correctly identify parts of speech when given combinations of words that it was trained with (e.g. “cat chase dog” is correctly identified as [N, V, N]), it struggles to identify parts of speech of words it has never seen. For example, “Logan ran and jumped” is identified as [N, V, CNJ, V]. Still this exercise reminded us better of how the viterbi algorithm works when it sees new words – Viterbi guesses the next POS solely based on what the most likely transition from the current POS is (the unseen punishment creates a level playing field across all POS it could transition to).