Testing Tagging with Viterbi Algorithm: Hard-Coded Training

<u>Hard-Coded Training:</u>

To train the *POSTagger*, I manually recreated the map from the programming drill. This map is very limited, but can illustrate successful execution of the Viterbi Algorithm with the right sentences.

Note: console output of the *manualTaggingTest* method is located in *manuallyTrainedTaggingOutput.pdf*.

With the information about its processing printed to the console, we can see the iterations over each element of the sentence. The processing shows each set of current and next states with corresponding scores. At the end, the best *finalState* is shown, along with the *backPointerList* and final string of computed tags.

This step by step information was especially helpful during debugging, but now we can use it to trace the successful execution of the Viterbi algorithm.

The first sentence: “I chase the dog .”

This sentence has two words that aren’t in this map, “I” and “the”, so we can see the algorithm default to the *unseenObservationScore* of -100.

As expected, the algorithm evaluates the *currentStates* with their possible *nextStates* and observation to produce the best *nextScores*. Then the subsequent iteration uses the previous *nextScores* as the *currentScores*. The iteration repeats until every observation has been processed. With the iteration information *backPointerList* and *finalState*, “V”, we can confirm that the backPointerList contains the appropriate entries. We can also confirm that the tags are produced correctly by manually reading through the *backPointerList* from the *finalState*.

The only difference between the computed and expected tags is the last entry, which isn’t a period. This is simply because the map doesn’t have the tag for a period in it, so it gives the period another mapped tag.

The accuracy of these computed tags makes sense, 2 right and 3 wrong, because it has only seen the second and fourth elements before, which are the only two it got correct.

The second sentence: “While we watch, you chase the dog and cat .”

Similarly to the first sentence, this one contains many elements that aren’t mapped. But the elements that it does have mapped, it successfully tags them.

In this sentence and the previous, the incorrectly guessed tags are all selected because of the low transition values associated with them. This confirms that the Viterbi Algorithm is making informed guesses about unseen tags.

Testing Training with Example Files

To test training from a pair of files, I used *example-sentences.txt* and *example-tags.txt*. The small maps produced are ideal for checking the training process.

Note: console output of the *exampleTrainTest* method is located in *exampleTrainTestingOutput.txt*.

Similar to the manual testing of the Viterbi Algorithm for tagging, the information printed to the console provides detailed information about the training process. The two main parts of the training are counting and probability calculation.

In the counting part of training, we see the training method examine two corresponding lines, one with a sentence and the other with its tags. Then it increments the appropriate counts for each word and tag. We can look over the incrementing steps to confirm the proper progression of the counting part of training. Once the counting is finished, the maps containing all the counts are printed, giving one last chance to review proper construction of the maps (no empty or missing counts).

Then the method turns all the count values into probabilities with natural logarithmic values of normalized counts. For each tag, we can see the counts before calculation and probabilities after calculation. Notably, only transitions has an entry for the start tag, "#", so the observations map doesn't have any counts processed for that tag. Finally, the fully trained maps for observation probabilities and transition probabilities are printed, giving another chance to confirm proper construction without any empty or missing probabilities.

Tagging Accuracy with Simple Dataset

<u>Console Output</u>

```
Beginning fileTest...
Beginning Tagging...
Tagging Complete!
Out of 5 lines, 0 were tagged correctly and 5 incorrectly.
Out of 37 tags, 32 were correct and 5 were incorrect.
Completed fileTest...
```

Tagging Accuracy with Brown Dataset

<u>Console Output</u>

```
Beginning fileTest...
Beginning Tagging...
```

```
Tagging Complete!
Out of 2390 lines, 1 were tagged correctly and 2389 incorrectly.
Out of 36394 tags, 35109 were correct and 1285 were incorrect.
Completed fileTest...
```

unseenObservationScore **Optimization**

Up to this point, the *unseenObservationScore* has been set to the default -100 as suggested in the problem set instructions. With some guessing and checking, we will attempt to find an optimal value (if it exists) for the *unseenObservationScore* in a tagger trained with the Brown dataset. We can do this by manually changing the value and rerunning the Brown dataset tests.

unseenObservationScore = -21

```
Beginning fileTest...
Beginning Tagging...
Tagging Complete!
Out of 2390 lines, 1 were tagged correctly and 2389 incorrectly.
Out of 36394 tags, 35109 were correct and 1285 were incorrect.
Completed fileTest...
```

-21 was the greatest integer value for the unseenObservationScore I could find that produced the same accuracy as every value lower than it. Decreasing the score beyond it, even to -300, saw no change in the accuracy of the tagger.

unseenObservationScore = -20

```
Beginning fileTest...
Beginning Tagging...
Tagging Complete!
Out of 2390 lines, 1 were tagged correctly and 2389 incorrectly.
Out of 36394 tags, 35111 were correct and 1283 were incorrect.
Completed fileTest...
```

-20 is the first integer score that changed the accuracy. This was an increase in accuracy, confirming that there is indeed an optimal value for the *unseenObservationScore*.

unseenObservationScore = -15.625

```
Beginning fileTest...
Beginning Tagging...
Tagging Complete!
Out of 2390 lines, 1 were tagged correctly and 2389 incorrectly.
Out of 36394 tags, 35120 were correct and 1274 were incorrect.
Completed fileTest...
```

After some rigorous guessing and checking in a bisecting manner, I managed to manually find the value -15.625 as an optimal *unseenObservationScore* for this dataset. Other values greater than or less than this one did not produce better accuracy.

unseenObservationScore = -14

```
Beginning fileTest...
Beginning Tagging...
Tagging Complete!
Out of 2390 lines, 1 were tagged correctly and 2389 incorrectly.
Out of 36394 tags, 35069 were correct and 1325 were incorrect.
Completed fileTest...
```

This is to prove that scores greater than -15.625 decrease the accuracy. The accuracy get worse as you continue to increase it.

Take-Aways about *unseenObervationScore*

Significantly low values (less than or equal to -21) all equally impact accuracy. I believe this is the case because significantly low values would lead the Viterbi algorithm to strongly prefer states that have any score for the observation, regardless of the transition scores. When calculating a *nextScore*, . This helps explain why the value is -21 because that is close to the sum of worst transition and observation scores. If the unseenObservationScore is low enough to overcome the worst possible combination of transition and observation scores, it would heavily encourage the algorithm to prefer even the worst possible score as long as the observation is not unseen at that state.

As the value approaches -15.625, the accuracy approaches its maximum with 35120 correct tags. I believe that this score is best because it is only slightly worse than the worst observation score. Unlike scores less than it (approaching -21), this score discourages the algorithm from states that haven't seen the score, but gives room for the transition score of that state to make an impact. If the transition score is given room for impact, then it can override an unseen observation score with a very good transition score. In those cases, the algorithm could avoid incorrect tagging by preferring states with better transitions (sentence context) where the observation is rare.

Values greater than -15.625 quickly produce very inaccurate tags. I believe that this is because a high *unseenObservationScore* confuses the algorithm by producing unseen observation scores better than mapped observation scores. In those cases, the algorithm would produce an incorrect tag because it preferred a state that hasn't seen the observation over one that has.

Console-Entered New Sentence Tagging Accuracy with Brown Dataset Training

```
I wonder what the weather will be tomorrow .
Tagged Sentence: PRO V WH DET N MOD V N .
```

This relatively simple sentence was perfectly tagged!

The elephants are stampeding !
Tagged Sentence: DET N V . .

This is an example of a sentence with punctuation that the tagger could not handle. It also couldn't handle the word stampeding, which it likely never saw before. It failed on the word stampeding not only because it was unseen, but because there wasn't enough other context in the sentence.

Tomorrow , the weather will be partly cloudy , with a high of 91 degrees and a low of 69 degrees .
Tagged Sentence: N , DET N MOD V DET N , P DET ADJ P DET N CNJ DET ADJ P NUM N .

In this case, the tagger had trouble with the description of the weather. For the words partly and cloudy, the tagger likely rarely or never saw these words during training. On top of that, we will see that the rest of the sentence doesn't provide solid context. In the case of the words high and low, they should have been tagged as nouns. It makes sense that the tagger failed on these because the use of high and low as nouns really only occurs in weather descriptions. If the tagger wasn't trained with weather descriptions, then it wouldn't be prepared to interpret high and low as known. The number 91 was also tagged incorrectly. This is likely because the number was never seen during training. Since the number 91 wasn't mapped, the tagger had to rely entirely on sentence context. But as we know, the sentence context is confusing enough due to uncommon language involved in weather descriptions.