Project 3 Writeup

CSC 246 Spring 2021

Authors: Michael Pirall, Sufian Mushtaq, Abdul Moid Munawar, Tianyi Ma

1. Introduction

- Only used samples with less than 100 words
- Used 100 samples in total out of the dataset as training was taking too long on our computers and we needed to try many permutations of training for the report.
- Flourish: Added viterbi decoding and prediction on top of simple prediction method.
- Collaboration Statement:
  Initially, our tasks are divided into the following:
  1. Creating and initializing HMM → Michael Pirrall
  2. Forward-backward algorithm → Tianyi Ma
  3. Log Likelihood → Tianyi Ma
  4. Expectation Maximization → Sufian Mushtaq
  5. Simple prediction method → Tianyi Ma
  6. Viterbi algorithm and predicting using Viterbi → Abdul Moid Munawar
  7. Experiments, writeup and readme → Michael Pirrall

  After we completed our parts individually and combined them, we worked on testing and debugging as a whole so each of us has basically worked on every part of the project in the end.

2. Experiment

In order to determine the parameters that best

a. *Iterations it took to converge*

To determine when our model converges, we used **ε = 0.00001**. If change in log likelihood after an iteration was less than ε then we assumed that the log likelihood converged. We chose this value as log likelihood always hovered in negative hundreds or negative thousands and so this value of ε is extremely small relative to that. The following table summarizes our convergence values:

Number of States	Iterations taken to converge
5	246
10	160
15	257

b. *Hidden states that results in the most accurate*

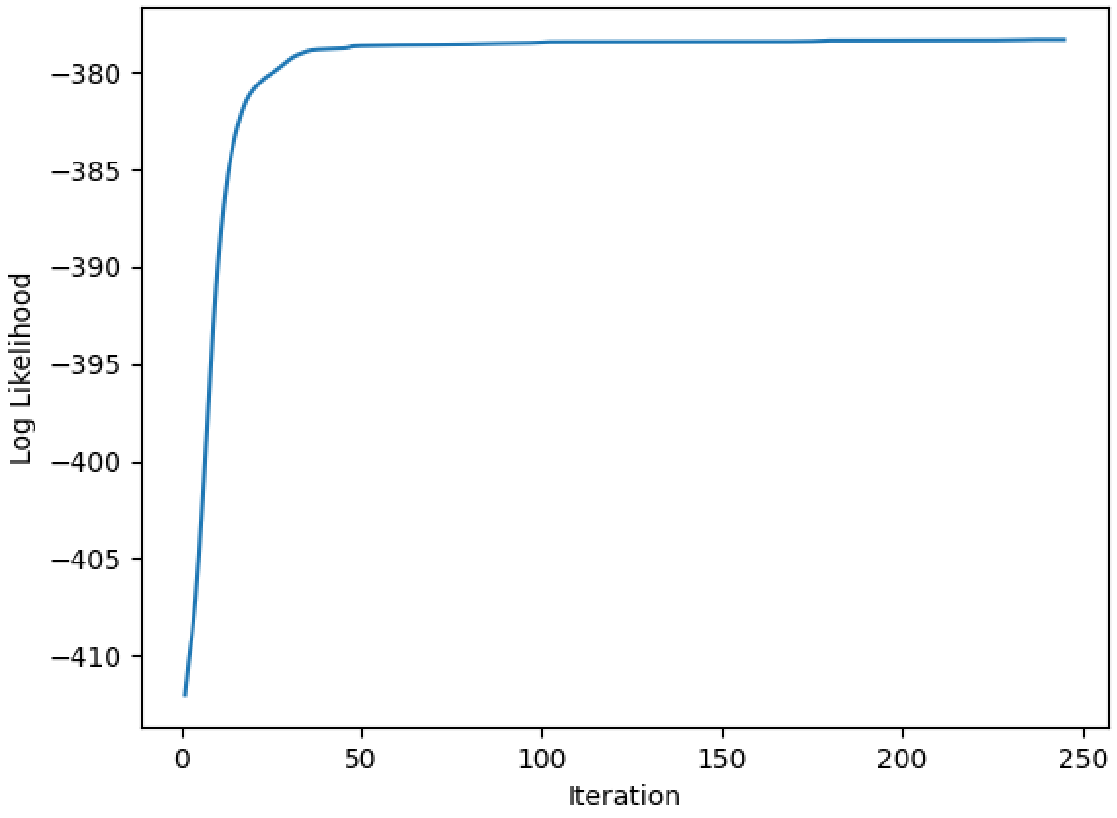

Fig. 1

Fig. 1 shows the log likelihood over time on training data with **5 hidden units**. As aforementioned, this model converged at **245 iterations** with a final log likelihood of **-378.30927896**

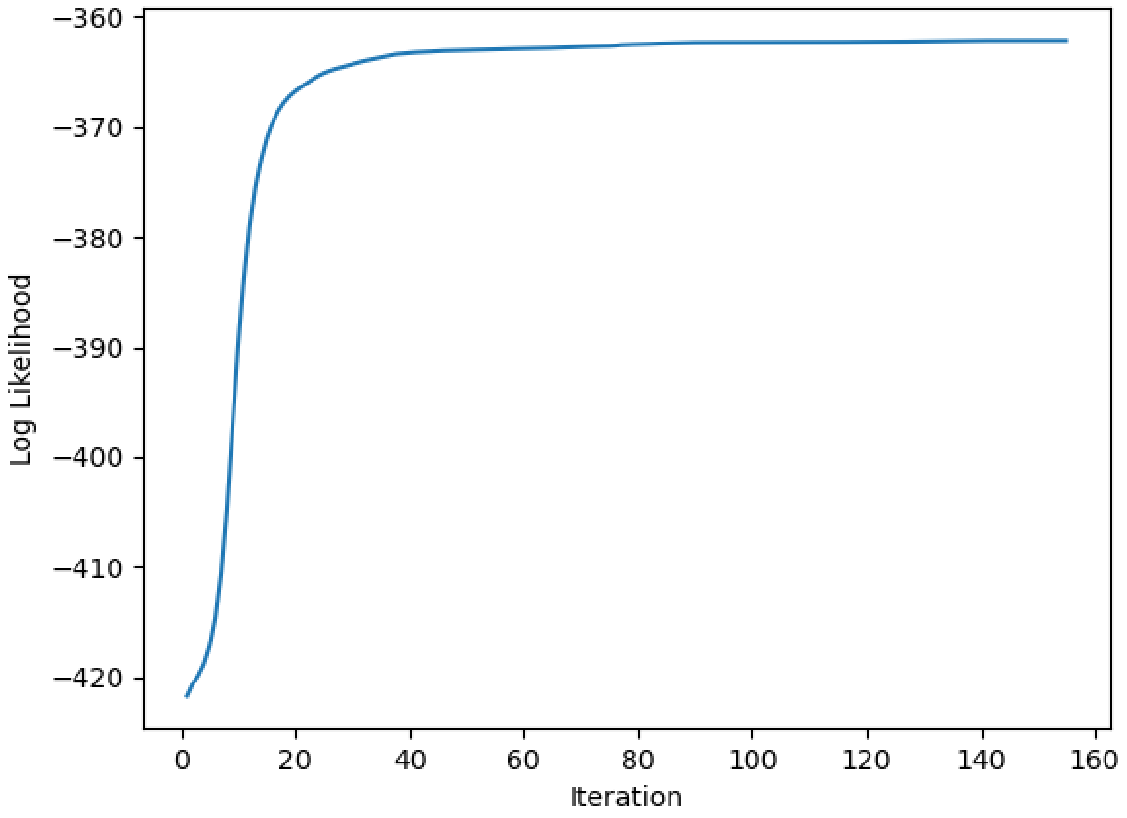

Fig. 2

Fig. 2 shows the log likelihood over time on training data with **10 hidden units**. As aforementioned, this model converged at **160 iterations** with a final log likelihood of around **-362**.

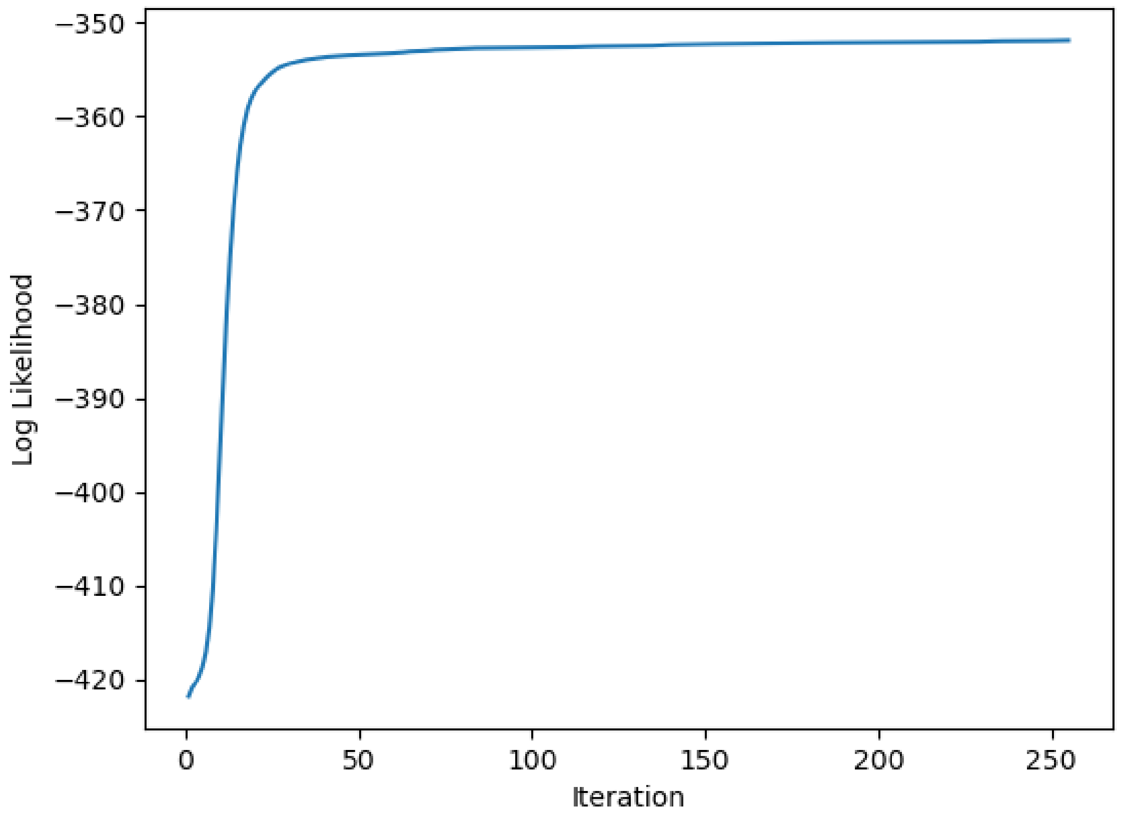

Fig. 3

Fig. 3 shows the log likelihood over time on training data with **15 hidden units**. As aforementioned, this model converged at **257 iterations**, with a final log likelihood of around **-352.**

According to a summary of the results mentioned above, we've determined that the model with 15 hidden units was working the best since it reached a higher log likelihood score even though it took a few more iterations than the other models. Adding more hidden states was not feasible for our model but it is safe to assume that adding more hidden states would increase the log likelihood of our model.

c. *Regarding the likelihood, do the results seem to be theory bound, compute bound, or memory bound? Why? You should be able to answer this question based on your results from the previous question. Be sure to compare likelihood on training and test data. Do you detect any overfitting? What about underfitting?*

We did not try beyond 15 hidden states, but based on the trends in the previous answer we expect that the training would have improved steadily with higher numbers. Due to time constraints, we could not test beyond 3 values of hidden states: 5, 10, and 15. So we would argue that the log likelihood of both the train and test data was computationally bound for us, as the time factor did not allow us to give potentially better conditions to the training and improve the log likelihood of both the train and test data even further.

Analysis of log likelihood training vs testing**:** The trend of the log likelihood was always upward for training as iterations progressed, as it should be mathematically, However, we see in figures X, Y, and Z below that for all variants of the hidden states, the log likelihood for the test data gets worse as iterations proceed. This shows that there is **overfitting in our model** as iterations progress. A likely reason for this is that we were limited to 100 samples due to computational constraints, and if we had been able to use 1000+ samples then we may have avoided this problem and would’ve observed an increasing log likelihood in both the testing and training data.

Figures X, Y, and Z show the log likelihood of the trained model on the testing data.

Hidden states = 5

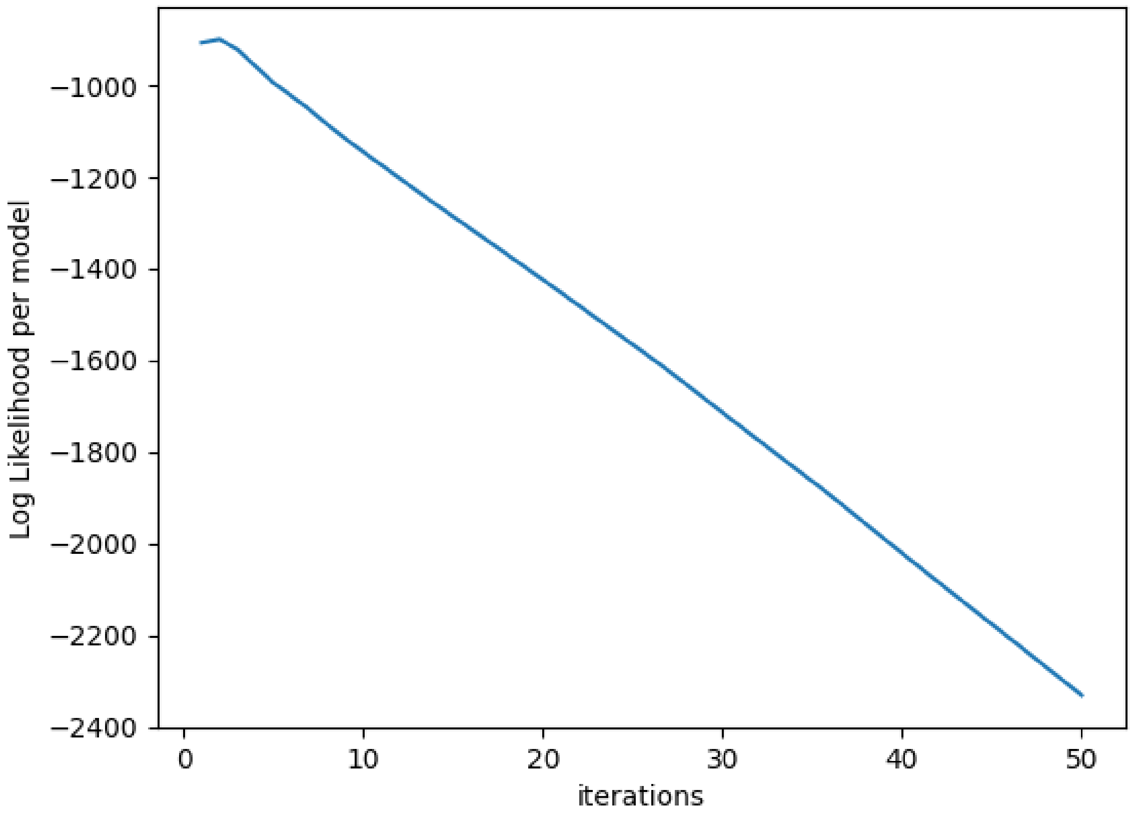

Fig. X: Log likelihood against epoch on the **testing** data using **5** hidden states

Hidden States = 10

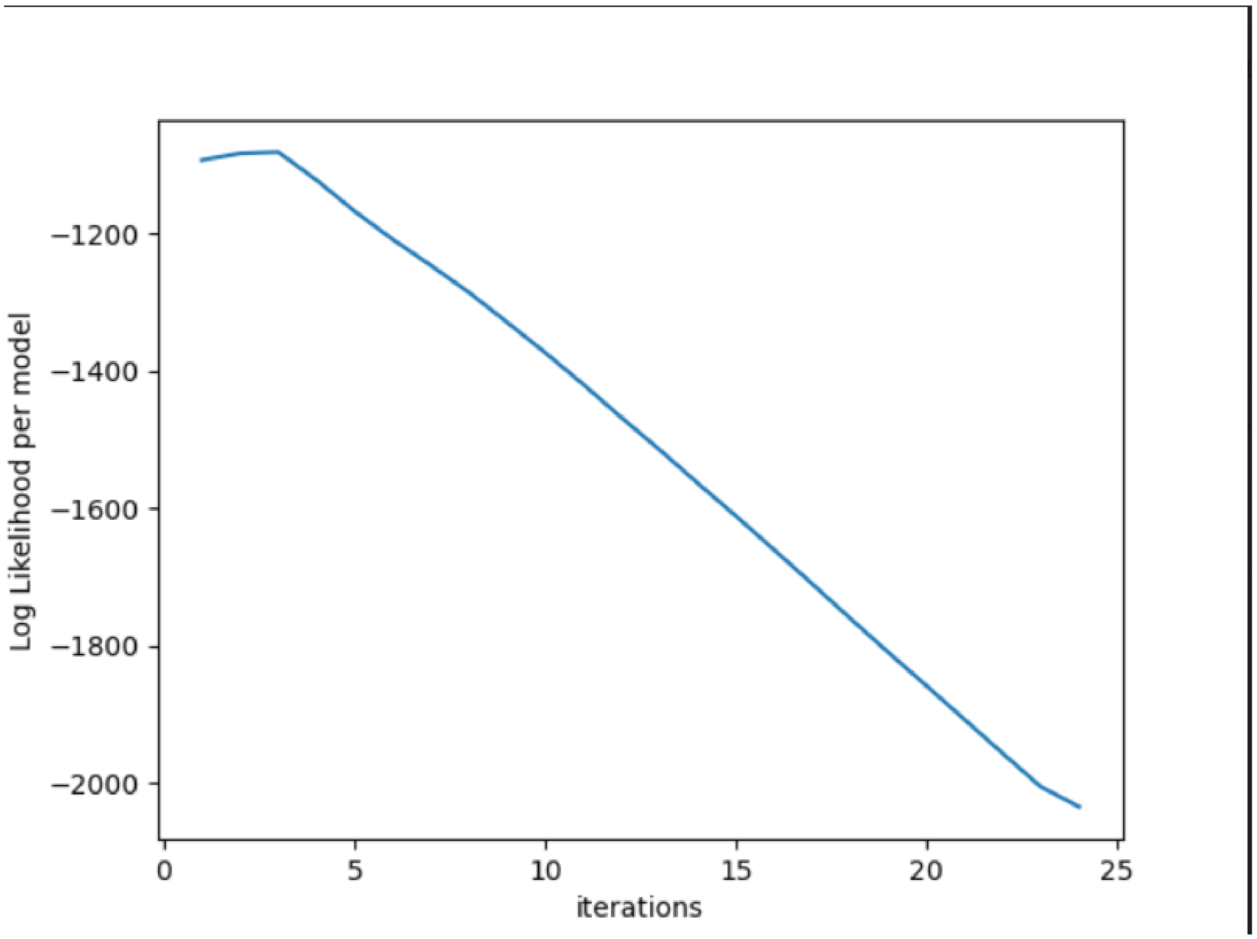

Fig. Y: Log likelihood against epoch on the **testing** data using **10** hidden states

Hidden states = 15

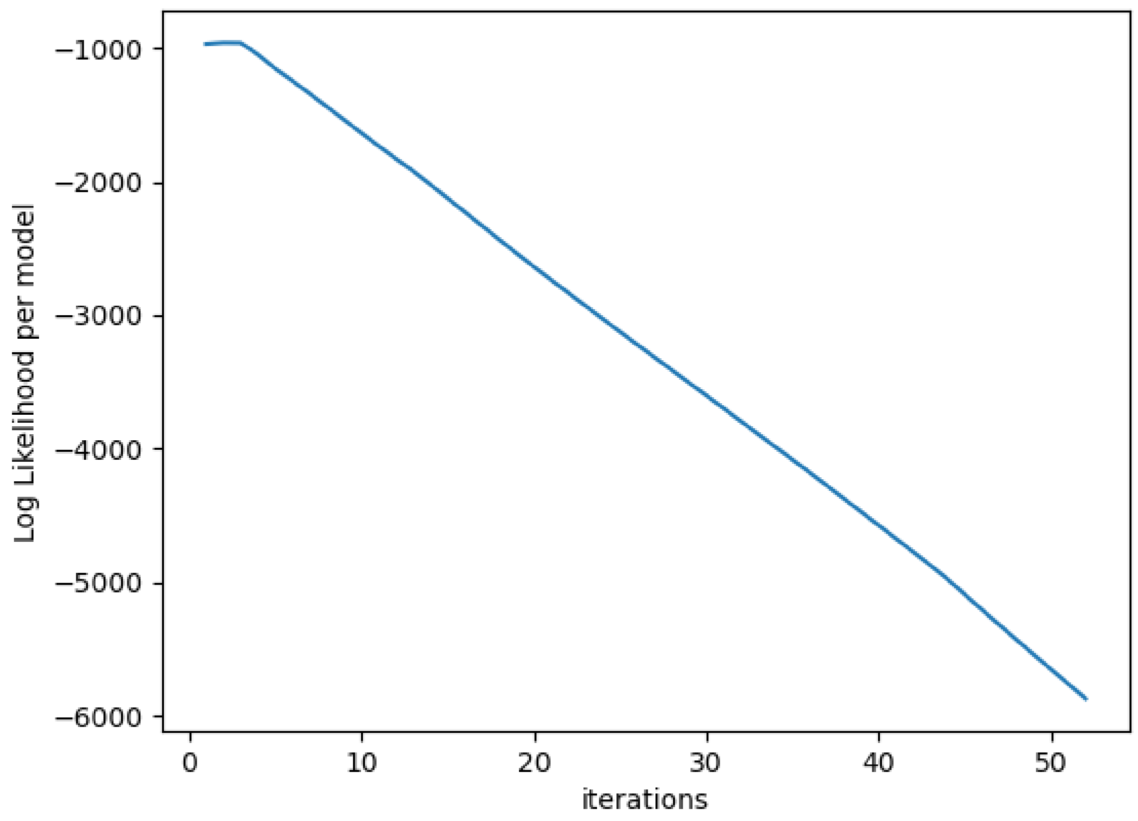

Fig. Z: Log likelihood against epoch on the **testing** data using **15** hidden states

d. *What about predictive power? Evaluate accuracy when predicting into the future. To push further, you may explore how the accuracy drops off when predicting t steps into the future. Is there a limit at which the quality of predictions from your model drops at random chance? How does this relate to the number of hidden states?*

Instead of using simple prediction, we used Viterbi Decoding to predict 1, 3, and 5 words into the future. The Viterbi algorithm chooses the most likely sequence of hidden states, from which we can get a sequence of words. The graphs below predict the accuracies for the training and testing data against the number of words predicted. As we predict more and more into the future, the accuracy seems to increase which is unexpected. The reason for that is unclear to us, but further speculation is below.

When printing the sequence of words predicted into the future, for 5 words into the future with 15 hidden states, we got the predicted sequence of words as [and, the, and, the, and] when the model had converged. “and” and “the” are extremely common words, in fact they are likely the most common words to be seen in the training samples. For our rather small model, it makes sense that it would choose the most common words in a sequence, even if it isn’t the most realistic prediction. This is further exacerbated by the fact that we had to throw away many of the words in the test data as they were not in the vocab of the training data. This likely made the model perform even worse than normal. When testing against the training data that it was trained with, it still performed poorly though, likely due to the fact that it was just choosing the most likely words “and” and “the,” which, while the most common in the set, were still relatively unlikely to appear as the exact next word. The increase in accuracy with more words predicted into the future is rather small, only 1-2% which may come from the fact that in a longer sequence of words the model was more likely to get a correct guess that the word was “and” or “the.”

Hidden States = 5

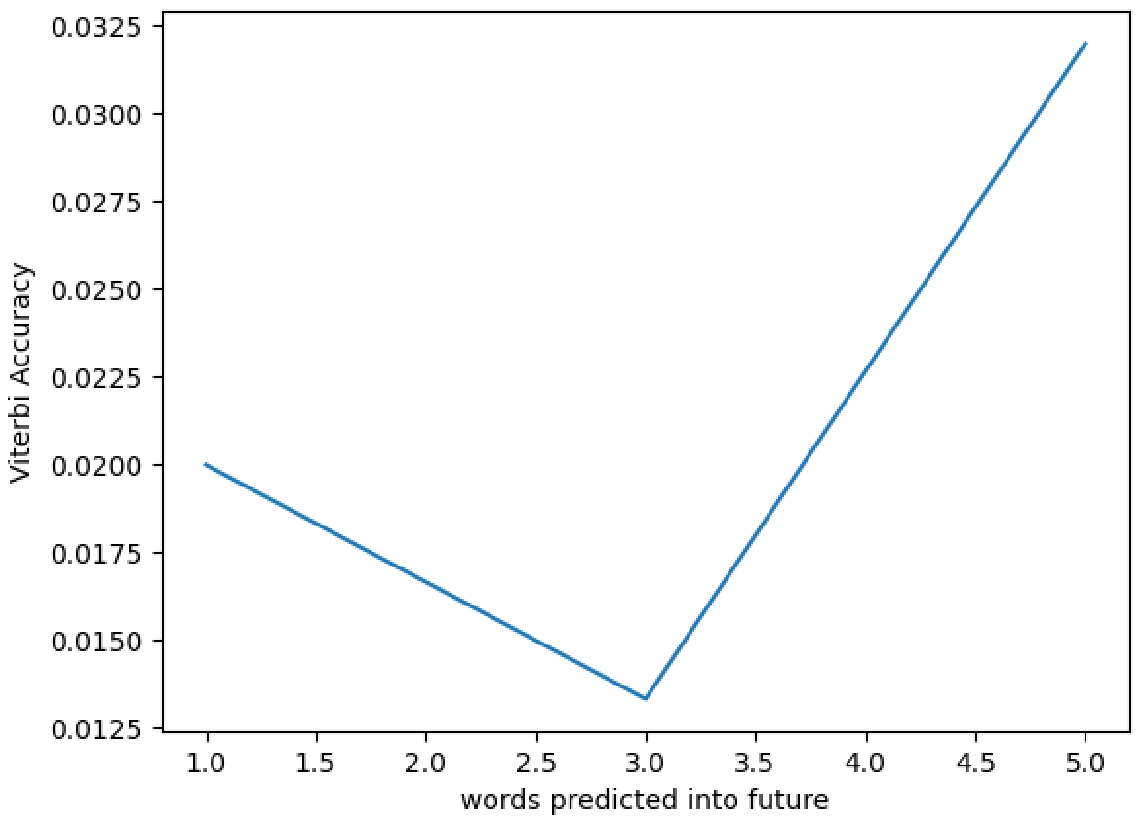

Fig A: Prediction accuracy using Viterbi Decoding on **testing** data for 5 hidden states

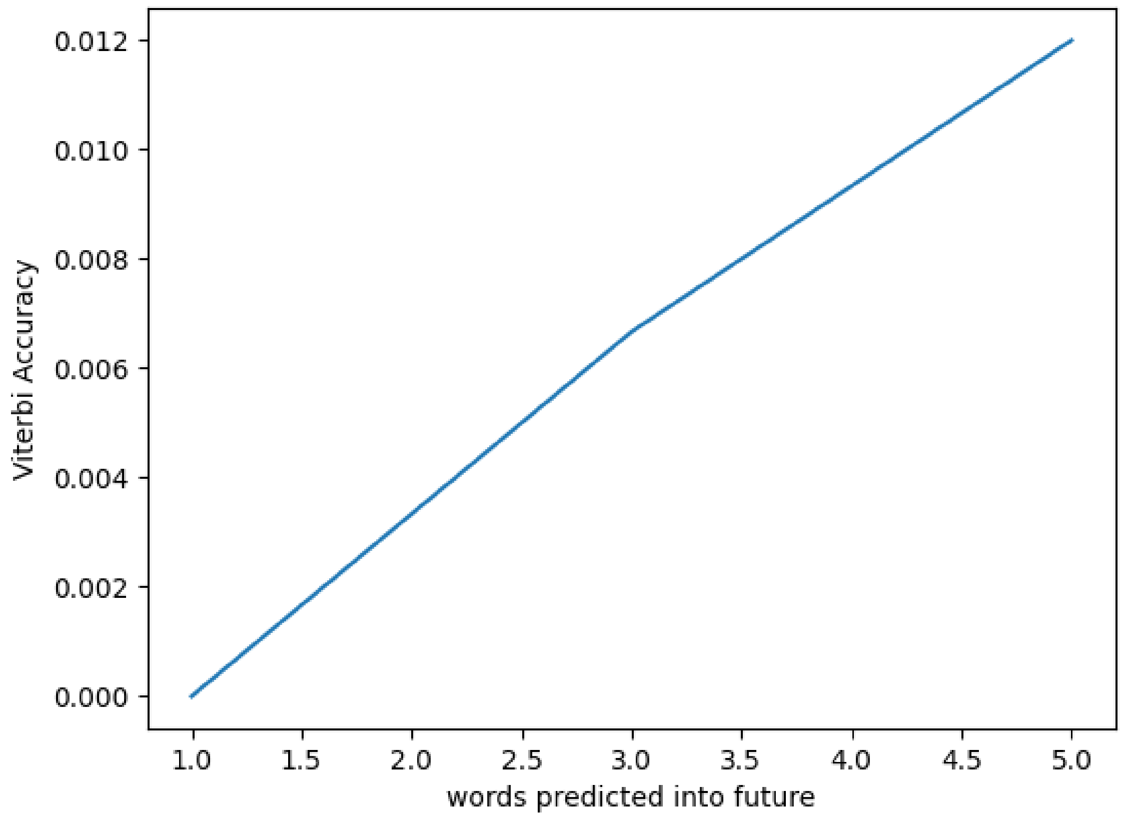

Fig B: Prediction accuracy using Viterbi Decoding on **training** data for 5 hidden states

Hidden States = 10

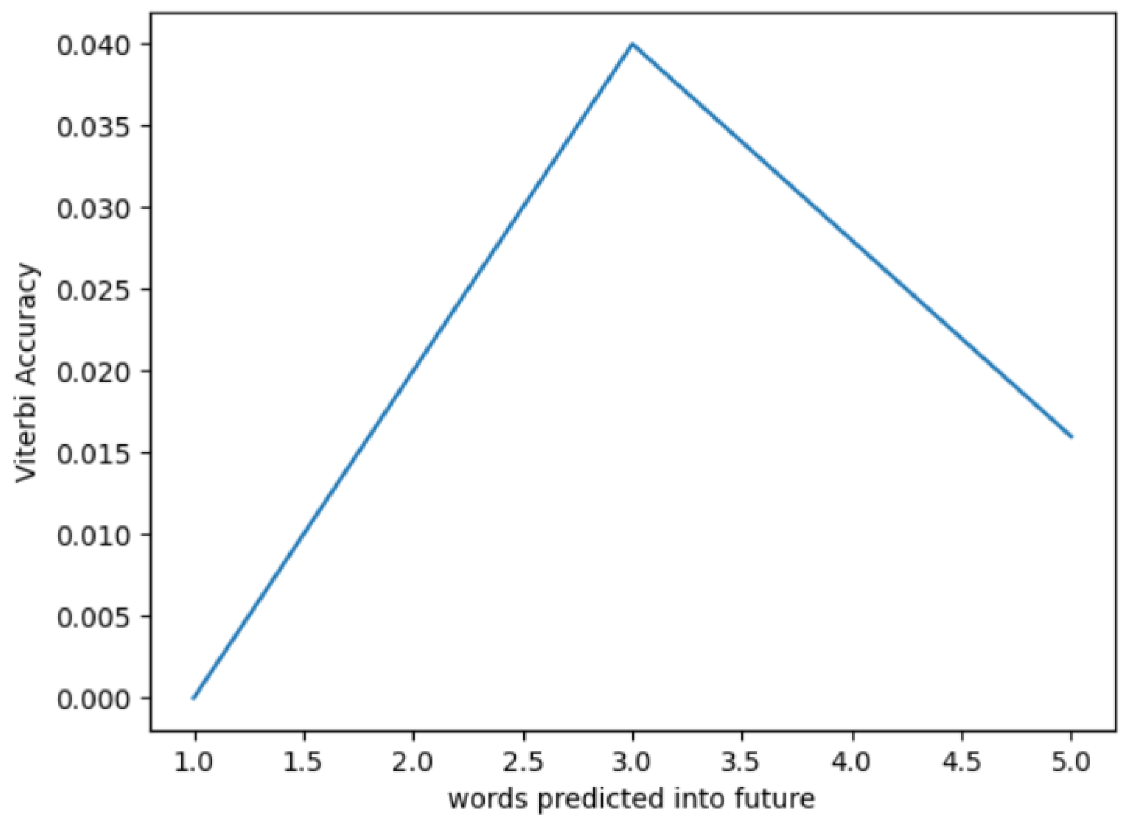

Fig C: Prediction accuracy using Viterbi Decoding on **testing** data for 10 hidden states

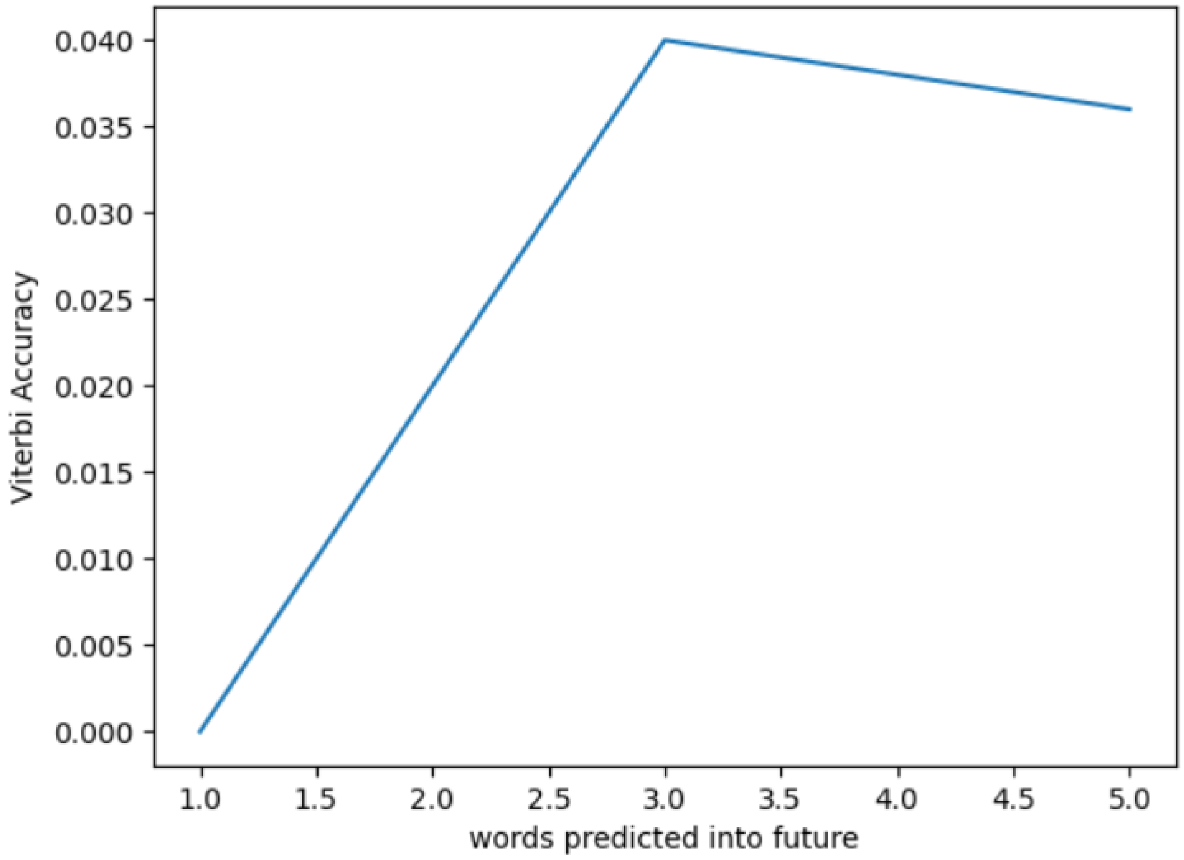

Fig D: Prediction accuracy using Viterbi Decoding on **training** data for 10 hidden states

Hidden States = 15

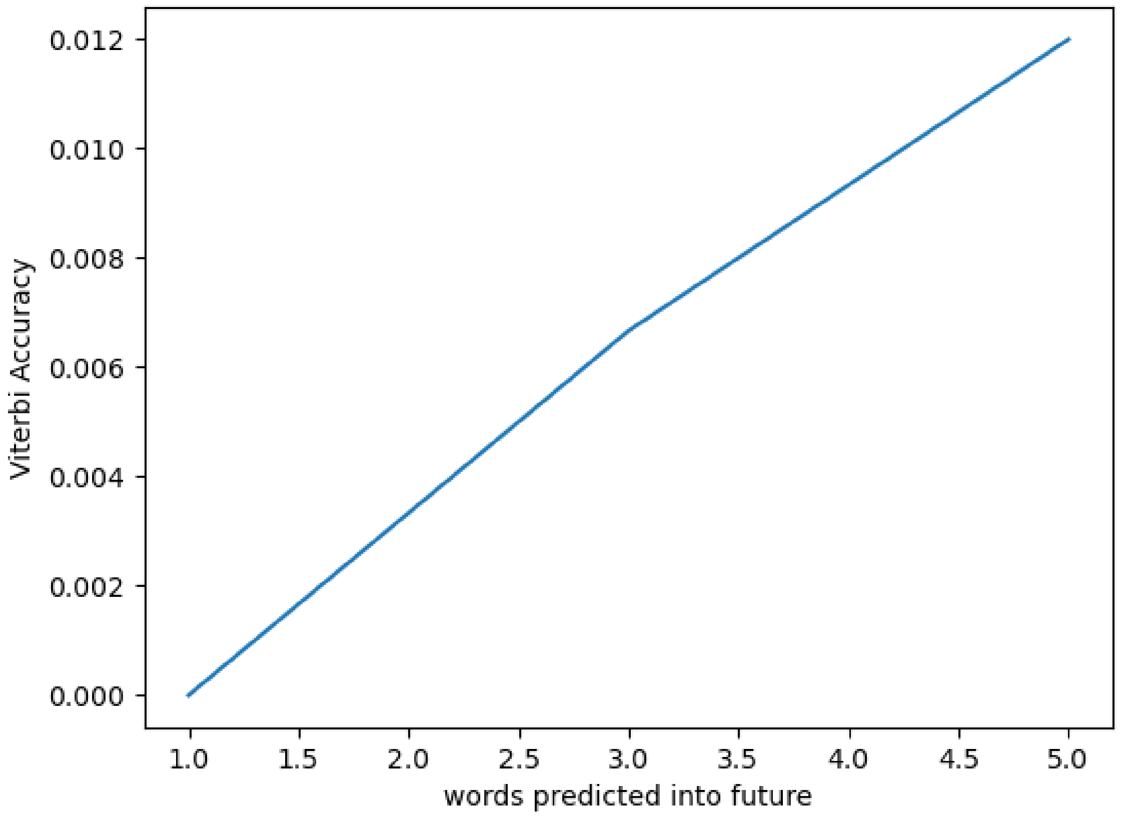

Fig E: Prediction accuracy using Viterbi Decoding on **testing** data for 15 hidden states

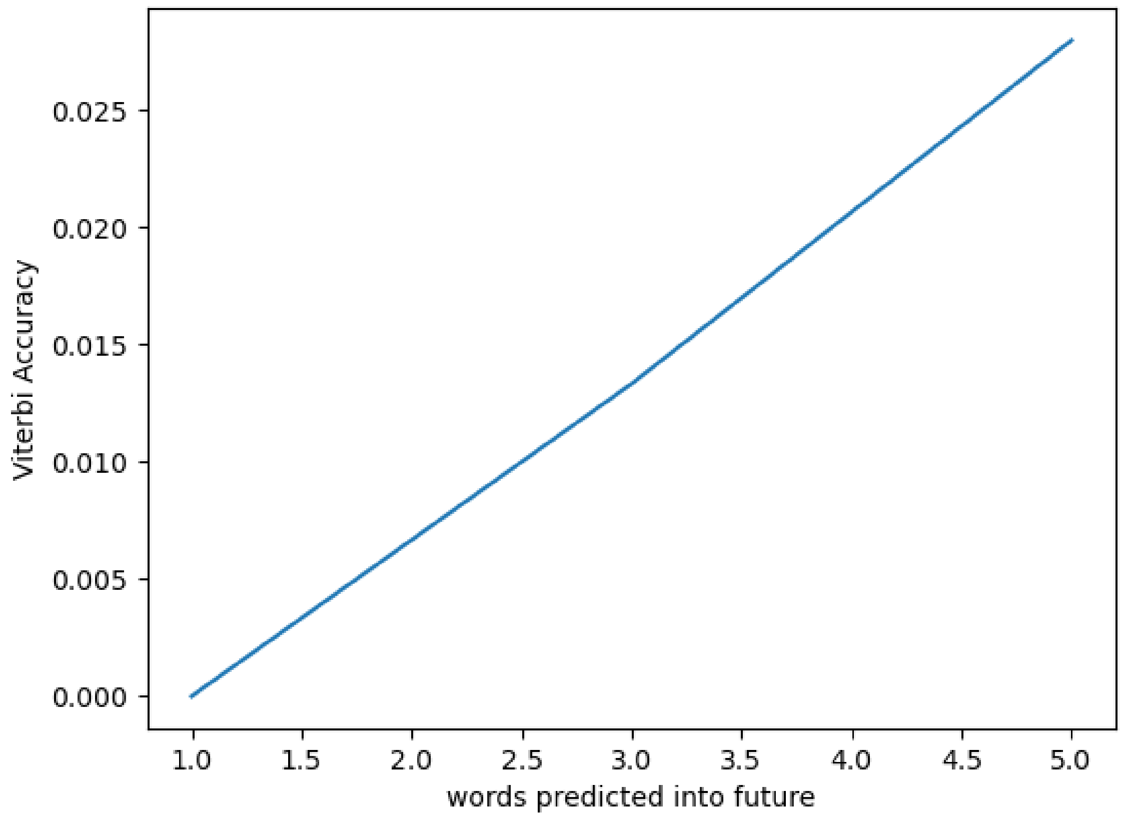

Fig F: Prediction accuracy using Viterbi Decoding on **training** data for 15 hidden states

Flourish: Comparison of viterbi prediction vs simple prediction

- Viterbi prediction is significantly faster than our simple prediction implementation, so much so that we can predict multiple permutations of words into the future using viterbi in seconds, but simple prediction takes an hour to predict one word into the future.
- We were able to use viterbi for prediction 1,3 and 5 words into the future (for both train and test data and for all hidden states) and plotted the accuracies in the above section. With simple prediction we only did one word into the future for (5 hidden states on test data) and (10 hidden states on train data) as it was too slow to try all permutations.
- Simple prediction picks up the word with the highest log likelihood when being appended to the end of the sample. When predicting multiple words into the future, it predicts one word, attaches it to the end of the sample, and predicts the next word.

- Simple prediction was more accurate for one word into the future and that it all we could test with it. The main advantage of viterbi is how fast it is, especially for multiple words into the future.

  Here is a comparison table of our viterbi and simple prediction values:

Number of States	Viterbi Accuracy for one word	Simple Pred. Accuracy for one word
5	0.02 (test data)	0.4 (test data)
10	0.0 (train data)	0.38 (train data)

Conclusions:

Due to the limitations of training time, we were only able to train our model on 100 smaller samples with 5, 10, and 15 hidden states. Obviously the model is not great, however the math and implementation is correct, as can be seen by the log likelihood convergence, so the theoretical basis of the model was a success at least. When it comes to predicted accuracy, viterbi was faster but performed horribly, getting around 2% accuracy. The simple prediction accuracy, however, was a good bit better, getting around 40% accuracy. Given how few samples the model was trained on, these are some pretty good results, especially compared to viterbi.

While the HMM is correct mathematically speaking, the run time limited us greatly in our experimentation and model building, resulting in subpar models and experimentation results. If we had more time available, we would have liked to train models for longer periods of time, thus likely getting higher accuracy and better results. Overall though, we are happy to have gotten our HMM working correctly, despite the subpar models.