Stat 534 final project

Jeffrey Lee
1826649

Project step:

In this project, we will input 1000 sequences observation data. Each sequence is 40 data long. We need to find the most suitable parameters and use the parameter to predict the hidden state and also the next observation of provided sequences. Therefore, I choose to initialize the model by random emission, transition, start probabilities since we don't know the original distribution of those data. I used cross validation to choose the number of states. With the number of states in mind, I can try to feed in the data into Baum-Welch algorithm to get the training parameters. We can use those parameters to predict the output and distribution from the test set. I will explain more about the algorithm I used in the following report.

Model Training:

In this project, I used the package called hidden_markov from Rahul Ramesh. Can be downloaded by $ git clone https://github.com/rahul13ramesh/hidden_markov.git
To train the data, I first initialize the model object with random start probability (π), emission probability (B), and transition probability (A). Those are generated by the random function therefore it will be different every time. Later, I can use train_hmm function to train the observe sequence. After running for number of iterations, the model finally converge meaning that the model is ready to be used and the parameters has been set.

Number of states:

I used K-fold cross validation method to choose my number of states. In the cross_validation.py code, I implement this by importing the package called sklearn.model_selection. A common number of k is 5 or 10. In this validation I chose 5, meaning I will split the training data into 5 folds. I used 80% and 20% of data as training and validation set respectively. This time, I train the model only by validation part, which is 20%. This will help me get emission probability and transition probability. For the rest 80% of data, I used viterbi function to decode it and obtain the hidden states. Using this hidden states and A, B matrix, I can easily compute the predicted observation (y tilda). By comparing the ground truth of original data. I can compute the loss from different number of states. I tried from 2 to 8. The table is the result.

Number of states (N)	Total loss (100 iterations)
2	32000
3	24833
4	23451
5	21646
6	27096
7	27484
8	30193

With this result, we can clearly see that all the states number have a great amout of loss. That is because I only trained for 100 iterations. However, we can still see that there's a trend that choosing 5 states will give us the best result to train the data, and therefore I used N = 5 in the later implementation.

Baum-Welch Algorithm:

The Baum-Welch algorithm in the package I used is called train_hmm function. Though I used the package, I do modify the code a bit. Like the passing parameter for this function become observe sequences, number of iterations, quantities, and tolerance. Therefore, I will upload the package as well. For the tolerance parameter, I followed the suggestion from professor's handout which is 10^{-4}. This the termination iteration for my project. And from the result, it is also the best prediction I can get. The quantities means that each sequence occurrence time. Also, I return more valuable data like loss and log likelihood to plot the learning curve for the result session.

The transition and emission matrix after 1000 iterations with tolerance: 10^{-4}.:
Emission probability:

```
[[0.382   0.541      0.065   0.0104]
 [0.091   0.152      0.683   0.074]
 [0.468   0.065      0.399   0.0669 ]
 [0.467   0.131      0.199   0.203]
 [0.039   0.199      0.120   0.642]]
```

Transition probability:

```
[[0.0012     0.0567     0.0604   0.1908     0.6908]
 [0.5955   0.1200      0.2226   0.0617     0.0002]
 [0.0166   0.2755      0.4595   0.1871     0.0614]
 [0.0001   0.2859      0.3344   0.0456     0.3340]
 [0.2417   0.2637      0.0371   0.1170     0.3406]]
```

Can also found in the file: modelpars.dat

Viterbi algorithm:

With those parameters from the train set, I can now pass the model into Viterbi algorithm to decode the hidden states. I pass both training data and test data to see the hidden states. It took 5.19 and 0.253 seconds respectively for both data. And now, we have the full sequence of hidden states.

Notation

$\delta_t(i)$	the optimum cost of a state sequence $[q_{1:t-1}, q_t = i]$ $= \max_{q_{1:t-1}} P[q_{1:t-1}, q_t = i, O_{1:t}]$
$\psi_t(i)$	pointer for backtracking $= q_{t-1}$ in the optimal state sequence

Initialization

Optimality principle

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(O_1) \\ \psi_1(i) &= \text{undefined} \end{aligned}$$

Recursion

$$\begin{aligned} \delta_t(i) &= \max_{j=1,\ldots N} [\delta_{t-1}(j)a_{ji}]b_i(O_t) \\ \psi_t(i) &= \operatorname{argmax}_{j=1,\ldots N} [\delta_{t-1}(j)a_{ji}] \end{aligned}$$

Formula 1.

From the recursion part formula 1, we can see that we use the previous state as index and find the argmax from the transition matrix and we can get the most likely output for the next output. The Viterbi is just a recursion part of this process. I can just use once from the last state of predicted output and using the same formula, we can get the predicted next state. In this project, it's the state T = 41.

Can find the most likely output for each sequence from the file called: Viterbi.dat

Can find the distribution for 41 data set in 50 sequence for the test set in the file: predict.dat

Forward and backward algorithm:

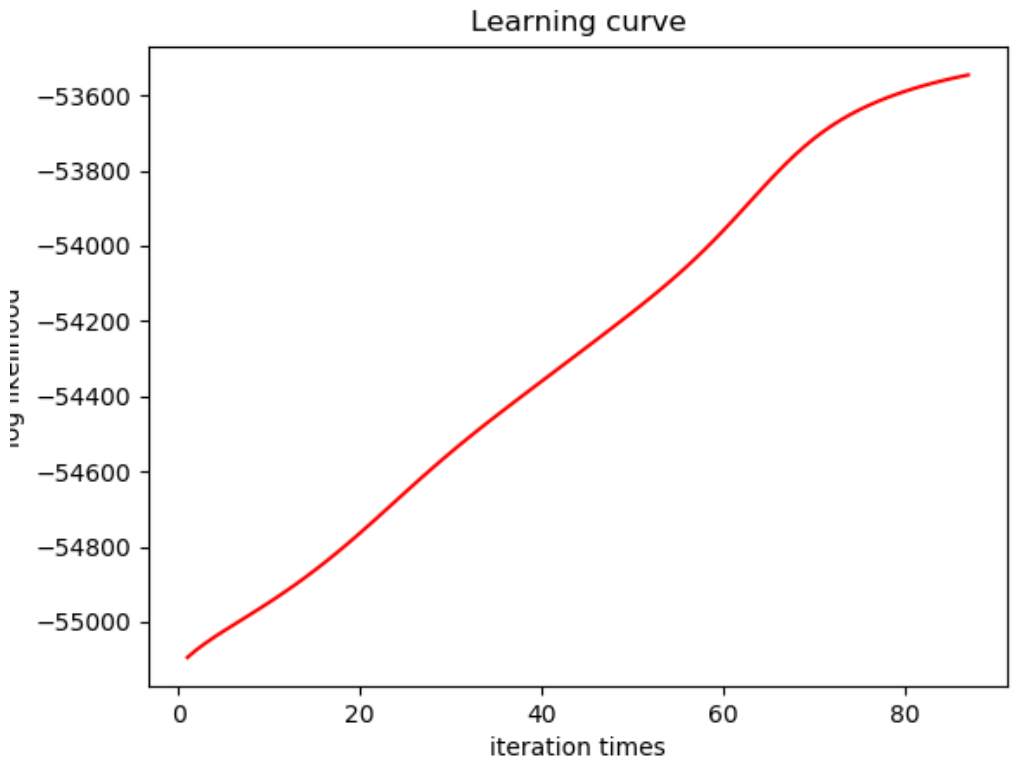
In this package, I can call the function call log_prob to easily calculate the log likelihood each training iteration. The first log likelihood is -59514.15, which is unlikely, and after training we can get the -53545.68. This shows that we actually learn from the model and the likelihood of prediction has improved.

The test set log likelihood is: -2725.23

Can also be found in the file: loglik.dat

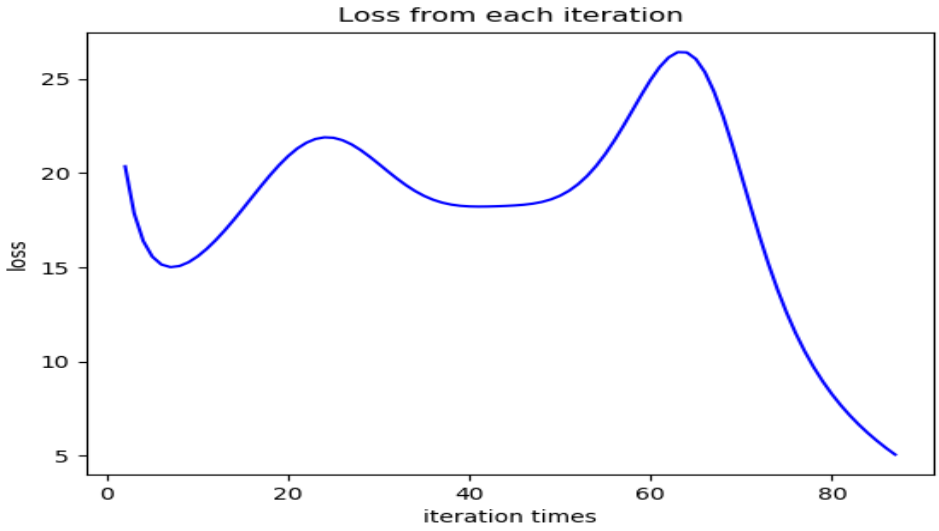
Result

Learning Curve:



From the plot we can see the log likelihood gradually increasing and eventually when the difference of the log likelihood did not over the tolerance, the program will terminate. The iteration time for this training is about 90 times for the training process to converge

Loss:



From the loss data plot, we can see that the loss for the data will vary during the time. In the middle of the training, it will sometimes become worse. However, as time past, we will able to find the model parameter and the loss for each iteration gradually diminish lower than tolerance.

Reference:

[1] **https://github.com/rahul13ramesh/hidden_markov.git**
[2] https://web.stanford.edu/~jurafsky/slp3/A.pdf
[3] https://machinelearningmastery.com/k-fold-cross-validation/
[4] Professor Marina's lecture handouts