HMM project Writeup

By David Lau

Introduction

For this project, I did most of the running of it on Ubuntu for Windows. I chose the Linux OS for the project because on Linux, the numpy longdouble datatype is a float128, where on windows is a float64. Being able to use this precision helped a ton when I ran into problems with underflow in my probabilities, as is common when working with HMMs.

```
Machine parameters for float64
---------------------------------------------------------------
precision =  15   resolution = 1.0000000000000001e-15
machep =    -52   eps =        2.2204460492503131e-16
negep =     -53   epsneg =     1.1102230246251565e-16
minexp =  -1022   tiny =       2.2250738585072014e-308
maxexp =   1024   max =        1.7976931348623157e+308
nexp =       11   min =        -max
```

```
Machine parameters for float128
---------------------------------------------------------------
precision =  18   resolution = 1.0000000000000002641e-18
machep =    -63   eps =        1.084202172485504434e-19
negep =     -64   epsneg =     5.42101086242752217e-20
minexp = -16382   tiny =       3.36210314311209350663e-4932
maxexp =  16384   max =        inf
nexp =       15   min =        -max
---------------------------------------------------------------
```

Figure 1: data type info for windows(top) and Linux(bottom)

I did the actual coding for the project in PyCharm.

Model Choices

I decided to use a word based model for the project, with the words being represented as integers as opposed to one-hot vectors. I did word based because I didn't want my model to generate gibberish by prediction, which character based models are weak to, and using integers allowed me to access arrays and values via the direct values of a sample.

I wanted to try and implement log probabilities so that I wouldn’t have to worry about underflow, but I decided that trying to make the math work was too ambitious of a venture for the for time coding the algorithm. Instead I used regular probabilities, and limited the length of the reviews to 1000 words long. I figured this was ok seeing as this only excludes around 1% of reviews.

My E-M algorithm updated the parameters based on estimations from the entire dataset all at once. I mention this because I wasn’t completely sure whether I should be updating as I went or not, but regardless, that's how I implemented the algorithm.

Experiments

For most of the experiments, I used a subset of 5000 sequences. I thought that this allowed me to get results in a reasonable time.

1:

My models tended to converge quite rapidly regardless of the number of hidden states and samples, usually only taking 2-4 iterations before no change was obviously noticeable. For this reason, and seeing as the log sort of makes changes in probabilities smaller, I made my epsilon extremely small, at e = 0.0000000000001 difference in log likelihood.

2:

I felt that 6 hidden states was the best balance between model complexity and the power of my rig. It took about 15 minutes to train my model to converge with this amount. Above 10 states seemed like it would take too long to be convenient.

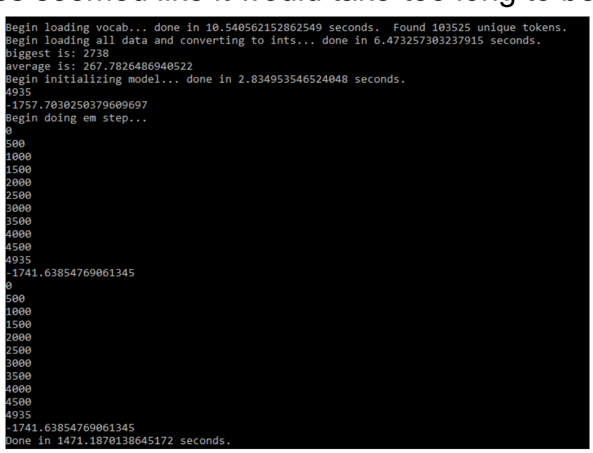

Figure 2: 10 hidden states takes 23 minutes

3:

The results seem very theory bound. A consistent likelihood could be settled on within a reasonable amount of time and more hidden states stop being unilaterally helpful eventually, so there’s not a lack of computing. Additionally, the memory used would max out at around 2 GB on my ram, which for my computer is a very reasonable ask. Therefore, the only thing really in the way of making the likelihood even better is most likely theory based.