COS 402 – P4

Toy Robot:

1. A sample of the results for robot_no_momemtum.data is:

   errors: 32 / 200 = 0.16
   errors: 42 / 200 = 0.21
   errors: 45 / 200 = 0.225
   errors: 36 / 200 = 0.18
   errors: 54 / 200 = 0.27
   errors: 41 / 200 = 0.205
   errors: 25 / 200 = 0.125
   errors: 39 / 200 = 0.195
   errors: 37 / 200 = 0.185
   errors: 40 / 200 = 0.2
   errors: 49 / 200 = 0.245
   errors: 37 / 200 = 0.185
   errors: 37 / 200 = 0.185
   errors: 58 / 200 = 0.29
   errors: 53 / 200 = 0.265
   errors: 42 / 200 = 0.21
   errors: 22 / 200 = 0.11
   errors: 55 / 200 = 0.275
   errors: 30 / 200 = 0.15
   errors: 42 / 200 = 0.21
   errors: 22 / 200 = 0.11
   errors: 45 / 200 = 0.225
   errors: 23 / 200 = 0.115
   errors: 24 / 200 = 0.12

   From the results we see that the Vitberbi algorithm has about a 20% error rate for this problem. The relatively high error rate is likely due to the random nature of the sensor model (if the robot doesn't get the right color on a square it guesses a random one). This means if the observed sequence was one of the random colors it is difficult to tell if it was because the robot guessed wrong, especially if the random color is on an adjacent square. In general, the HMM-based approach did okay in solving the problem.

   A sample of the results for robot_with_momemtum.data is:

   errors: 18 / 200 = 0.09
   errors: 30 / 200 = 0.15
   errors: 27 / 200 = 0.135

errors: 35 / 200 = 0.175
errors: 21 / 200 = 0.105
errors: 35 / 200 = 0.175
errors: 44 / 200 = 0.22
errors: 22 / 200 = 0.11
errors: 38 / 200 = 0.19
errors: 21 / 200 = 0.105
errors: 31 / 200 = 0.155
errors: 12 / 200 = 0.06
errors: 36 / 200 = 0.18
errors: 24 / 200 = 0.12
errors: 34 / 200 = 0.17
errors: 22 / 200 = 0.11
errors: 24 / 200 = 0.12
errors: 40 / 200 = 0.2
errors: 32 / 200 = 0.16
errors: 37 / 200 = 0.185
errors: 40 / 200 = 0.2
errors: 16 / 200 = 0.08
errors: 40 / 200 = 0.2
errors: 28 / 200 = 0.14
errors: 29 / 200 = 0.145
errors: 14 / 200 = 0.07

From the results we see that the HMM-based approach did much better on the robot problem with momentum than on the one without, with an error rate of about a little more than 10%. I infer that the reason the Viterbi algorithm did better on this one than the previous one is because this problem avoids the problem that the first one had. In particular, the previous problem was that the robot might randomly (and incorrectly) report a color which an adjacent square has. Whereas in this problem the robot is very likely to end up in a square not adjacent to its current square (because of the momentum) meaning if the robot incorrectly reports a color and moves two squares away and reports correctly we'll likely be able to infer that the previous one was incorrectly reported. In general Viterbi did better on the problem with momentum, but has a big spread.

2.

The assumptions made using an HMM is that where the robot is unobservable and where the robot is at each step is based only on where it was at the previous step. These assumptions match the actual process of generating data perfectly, since where the robot is at the next step depends on where it is now (it can only move one square at a time) and we randomly select whether it reports the correct square color based on the sensor model. This is because the problem was made specifically for an HMM, hence it is a "toy" problem.

3.

The performance was not hurt by making these assumptions since the assumptions are valid. For example, let's say the robot is at square (3, 3) and it moves to square (2, 3), it doesn't matter how the robot got to square (3, 3), it's probability of going to square (2, 3) from (3, 3) is the same. This is because its next move is solely dependent on where it is now (moving adjacently).

Correcting Typos Without a Dictionary:

1. The error rate for typos20.data is:

   errors: 2093 / 20064 = 0.10431618819776714

   And the error rate for typos10.data is:

   errors: 1048 / 20064 = 0.05223285486443381

   Naturally, we expect data with less noise to recover it better because there is less randomness in the set and therefore we are more certain of the true sequence. The HMM-based approach did fairly well at solving the problem with a 5% error on the data set with 10% noise. The reason I say the Viterbi algorithm did only fairly well is because for this kind of problem we place a greater emphasis on a lower error rate. This makes intuitive sense because for correcting typos we tend to either consider the problem unsolved (there are typos) or solved (there are no typos) in which case there is no error. On a numerical scale, however, the Viterbi algorithm did well with relatively small error percentage. If the first-order Markov model were extended to a second or higher order Markov model, the error would likely decrease dramatically because the higher order models are able to capture the word a letter is in, whereas the current one can't.
2. In this problem we assume the next letter in the text depends only on the previous letter and the corruption of the observed sequence is independent of each other. The assumption doesn't match the actual generation process as well as the toy robot problem did. This is because the next letter in the sequence has been deterministically chosen (it is transcribed from a chosen text, in this case, Unabomber's Manifesto) rather than randomly assigned based on a transition matrix of characters. The observed sequence generation was generated pretty accurately since, at each step, the observed sequence is randomly corrupted with some probability.
3. The performance was drastically hurt by the assumption that the next letter is solely dependent on the current letter. In actuality, this is not the case. The next letter depends on a variety of factors. In particular, it depends on what word is being typed (which in turn depends on the subject of the sentence, where in the sentence the word is, etc). Because we assume that the next letter depends on the current letter we introduce a lot of noise that should not necessarily be in the problem. For example, if I am typing probabilistic and the state is currently the last "i" the model uses the same transition matrix for the letter without considering what word it is in

(or the last three letters, for example). In this case a letter such as “m” would almost certainly not follow “i”, but because the transition matrix is the same for all i’s, m has a much higher probability of appearing next than it should. This skews the data and the most likely path. The weakness of the first-order Markov model in this HMM severely hurt the performance of the Viterbi algorithm on this problem.

Tracking a Changing Topic:

1. The error rate for topics.data is:

   errors: 74542 / 476168 = 0.15654558895179851

   From the error we can see that the Viterbi algorithm did fairly well, given the nature of the problem. For the most part, the algorithm was able to infer when the topic had changed based on words read in. From the data a fair amount of the error is when the word is a common word that appears in two or more of the topics (generic words such as “and”). Words that are topic specific (“shoot”) were virtually always categorized correctly.
2. In the context of the problem we assume that a change in topic depends only on what topic the state is currently on and the overserved sequence is dependent solely on the current state. The assumptions match the data-generating process fairly well. All articles were randomly permuted, therefore this is no bias on the probability of a topic switch, although in reality some topics are more likely to follow than others (medicine might be more likely to follow gun in real-life applications). The observation sequence generation matches the assumption perfectly (since each word is tied to the state).
3. Based on the data-generation method, the performance was likely not hurt by the assumptions made, since the testing articles were also randomly permuted. The reason for the 15% error was due to how the problem was formulated. If there is a sequence of generic words that isn’t naturally tied to a topic (such as this sentence), it is difficult to infer which topic it belongs to. This is because these words would appear in all the topics during training with relatively equal probability. In general the assumptions didn’t hurt the algorithm, instead the appearance of generic words was probably what caused the error by the Viterbi algorithm.