Math 350: An exploration of HMMs through doodles.

Joshua Little (407673)

19 December 2012

1 Background

1.1 Hidden Markov models.

Markov chains (MCs) work well for modelling discrete-time processes, or discrete-time approximations of continuous processes. Hidden Markov models (HMMs) are an extension to Markov chains that allow us to consider processes where the exact state is only indirectly observable. To illustrate this difference, consider the problem of tracking a car by GPS (the more complicated problem of doodles will be introduced in Section 2). A GPS device does get the exact location of where the car is at every time step, but rather a 2D Gaussian of where it likely is. An MC could use a past measurement to guess where the car is likely to be now; however, it has no way of incorporating multiple measurements to refine its estimate. An HMM, on the other hand, could use each successive observation to actively refine its guess as to where the car is.

1.2 The math of HMMs.

Let's get a little more concrete. Suppose we are interested in an MC model of a process $S_1, S_2, S_3, \cdots \in \mathcal{S}$ with starting state S_0 and transition matrix $T \in \mathbb{R}^{|S|\times|S|}$. Given the distribution $P(S_{i-1}) = \pi_{i-1}$, an MC would set the posterior distribution of S_i to be

$$\pi_i = P(S_i \big| S_{i-1}) = \pi_{i-1} T \tag{1.1}$$

Note that the case $S_{i-1} = s$ can be expressed with the above formula using $\pi_{i-1}(s') = \begin{cases} 1 & s'=s \\ 0 & s' \neq s \end{cases}$.

Suppose, now, that we are using an HMM and have a set of observations $O_1, O_2, O_3, \cdots \in \mathcal{O}$, with each O_i independent of all S_j given S_i, and an emission matrix $E \in \mathbb{R}^{|S|\times|O|}$, where $E_{so} = P(O_i = o \big| S_i = s)$. The

observation O_i tells us something about S_i. Using Bayes' rule (and knowing that O_i is independent of S_{i-1} given S_i), we can conditionalize the distribution of S_i on both S_{i-1} and O_i to get the posterior distribution

$$\pi_i = P(S_i \big| S_{i-1}, O_i = o) \propto P(O_i = o \big| S_i) P(S_i \big| S_{i-1}) = E'_o \circ (\pi_{i-1} T) \tag{1.2}$$

where E_o is the column of E corresponding to observation o, and $\circ$ denotes the entrywise product. Normalizing across the states gives us the actual probabilities for π_i. Equation (1.2) is how HMMs pick their next π_i.

Going back to our original scenario, we can now see how exactly HMMs would help GPS locate a car. At each time step, we would supply the HMM with the previous location guess π_{i-1} and the current GPS measurement O_i, and the HMM would give us a new location guess π_i conditionalized on both π_{i-1} and O_i (possibly using car speed and map data to guess the transition probabilities T).

1.3 The plan.

In Section 2, we'll look at what doodles are and how we'll model them. In Section 3, we'll look at how to generate sequences of states and observations, as well as how to calculate the probability of a given sequence of states and probabilities. In Section 3, we'll look at the forward-backward algorithm, which takes a sequences of observations and, for each step i, computes the probability of being in each state at that time. In Section 5, we'll look at the Viterbi algorithm, which takes a sequences of observations and computes the most likely corresponding sequence states. Finally, in Sections 6 and 7, we'll tie the previous two algorithms together with the Baum-Welch algorithm and its applications, which takes sequences of observations (without any states) and uses expectation-maximization to find a transition matrix and emission matrix pair that is likely to generate those observation sequences.

2 Doodles

2.1 Background.

In this paper, we shall examine the inner workings of HMMs through their application to studying doodles. Here, we are using the word doodle to represent any quick drawing consisting of a single path (i.e., drawn

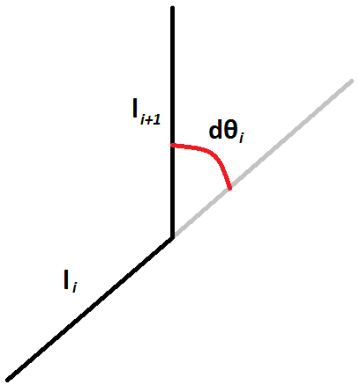


Figure 1: How we measure each $\mathrm{d}\theta_i$. The angle can be negative or positive depending on whether it's rotating clockwise or counter-clockwise.

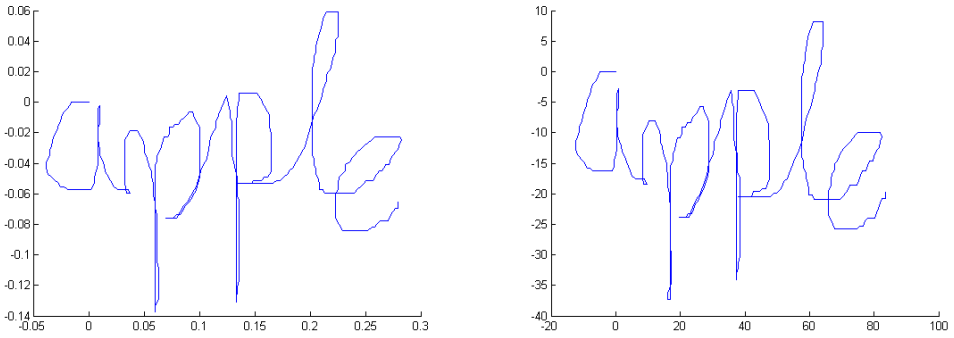
without lifting the pen off the paper). These doodles could range from a random scribble, to a star, to even a single word written in cursive.

2.2 Modelling doodles with HMM.

Before we apply HMMs to anything, we must first decide what will be our observations. The answer for doodles is not immediately obvious. For instance, we could fit a spline to the doodle by partitioning it up into many smooth curves, fitting n-degree polynomials to these curves, and using the parameters of these polynomials as our observations. This may or may not work well (with some tweaking), but sometimes simpler is better. In our investigation, we use the simplest feature we can as our observation that still adequately describes the doodle.

We approximate the doodle D as a connected sequence of line segments $l_1, \ldots, l_n$ and use as our observations the relative angles between successive line segments $\mathrm{d}\theta_1, \ldots, \mathrm{d}\theta_n$, as shown in Figure 1. Since our HMM models are discrete, we bin these $\mathrm{d}\theta$s and make our observations $O_1, \ldots, O_n$ the bin indices, so $O_i \in \{1, \ldots, m\}$ (with $m = 7$ or 11, typically). This representation is translation- and rotation-invariant, so the observations $\mathrm{d}\theta_i$ capture only intrinsic information about D. However, we are throwing away some information by not taking into account the length of each line segment l_i, leading to a slightly distorted view of the doodle, such as in Figure 2.

What the states $S_1, \ldots, S_n$ represent depends on the models. For a zigzag line, the state might represent simply whether the point is on an up-stroke, down-stroke, or corner. For a cursive word, the state could



(a) Doodle (b) Distorted doodle

Figure 2: An example of how doodles are distorted by ignoring line-lengths. Generally, the distortion is not too bad if the doodle is partitioned up fairly evenly.

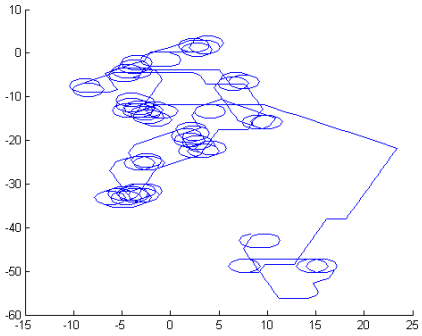
represent something much more sophisticated, such as what letter or part of a letter is being stroked. Indeed, when training an HMM from input data, what the states represent is sometimes the most interesting question.

3 Simulation and Probability

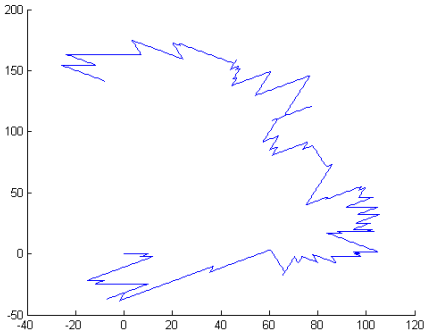
3.1 Simulation.

Given the transition matrix T, the emission matrix E, and some starting state S_0, simulating an HMM is simple: At each step i, sample a next state S_{i+1} using the probability row vector $T(S_i)$, and then sample an new observation O_{i+1} from the probability row vector $E(S_i)$. This generates a sequence of states $S_1, \ldots, S_n$ and observations $O_1, \ldots, O_n$.

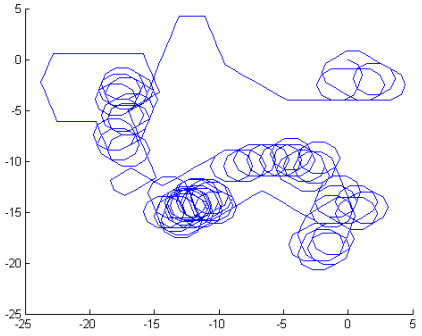
Using the transition and emission matrices in 3(e) gives us doodles like on the left in Figure 3. Note that here an emission value k corresponds to a relative angle of $\mathrm{d}\theta = (k-1)/10 \cdot 2 \cdot \pi - \pi$, giving a range of $-\pi$ to π in increments of $\pi/10$. Using, instead, the transition and emission matrices in 3(f) gives us doodles like on the right in Figure 3.



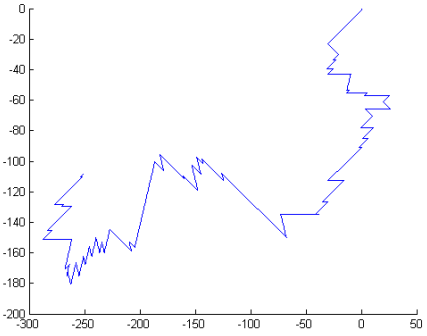
(a) HMM 1, simulation 1



(b) HMM 2, simulation 1



(c) HMM 1, simulation 2



(d) HMM 2, simulation 2

$$T_1 = \begin{pmatrix} 0.99 & 0.01 & 0 \\ 0.05 & 0.85 & 0.10 \\ 0 & 1. & 0 \end{pmatrix} \quad E_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0.9 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0.9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \end{pmatrix}$$

(e) HMM 1, for the left figures.

$$T_1 = \begin{pmatrix} 0.9 & 0.1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.9 & 0.1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad E_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 \end{pmatrix}$$

(f) HMM 2, for the right figures.

Figure 3: Two HMM models and their simulations. The columns of each E correspond to equally space angles between $-\pi$ and π. HMM 1 alternates between going in circles with state 1, and going mostly straight in state 2 with the occasional corner from state 3. HMM 2 produces a zigzag pattern, with the long lines of states 1 and 3 broken up by the alternating sharp corners of states 2 and 4. Note that in HMM 2, states 1 and 3 produce identical observations, differing only in what state they lead into. This would be impossible with a standard Markov model, where the output is the state.

3.2 Probability.

Given T and E again, we can compute the probability $P(S, O \big| P, E)$ of generating some sequence of states $S = \{S_i\}$ and observations $O = \{O_i\}$:

$$P(S, O \big| P, E) = P(O_n \big| S_n, T, E) \cdot \prod_{i=1}^{n-1} \left[P(O_i \big| S_i, T, E) \cdot P(S_{i+1} \big| S_i, T, E) \right] \tag{3.1}$$

$$= E(S_n, O_n) \cdot \prod_{i=1}^{n-1} \left[E(S_i, O_i) \cdot T(S_i, S_i + 1) \right] \tag{3.2}$$

Each term i in the product takes into account the probability of a single transition and emission, so the entire product must be $P(S, O \big| P, E)$. Most of the time this calculation is done in log-space, both to save on computation time, but also because the probability decreases exponentially with the size of the sequence and will quickly become zero in simulations. When comparing sequences of different length, the longer sequence will generally be much less likely simply because it is longer. So one should take the n^{th} root of each probability (where n is the length of the sequence) giving the geometric mean of the per-step probabilities. (Conveniently, in log-space this is simply the arithmetic mean.)

Using this equation, we can calculate the log probabilities for the doodles on the left in Figure 3, giving us -737.7635 and -537.3647 respectively, meaning that the bottom doodle is more likely and hence a better representation of the class. Similarly, for Figure 3, we get probabilities of -999.5230 and -441.6115 respectively, meaning that the top doodle is more likely.

4 Forward-Backward Algorithm

4.1 Background.

Given a transition matrix T, an emission matrix E, and a sequence of observations $O = \{O_i\}$, the forward-backward algorithm gives us $P(S_i \big| O, T, E)$, the probability distribution of over the possible states, for each time i. This is useful, for instance, if we're only interested in what the most likely state is at any point in time. If our states represent letters in our doodle, we could use the probabilities to do an intelligent search through a dictionary to find the most likely actual word. These probabilities are computed in three steps.

4.2 Forward algorithm.

First, we compute the forward probabilities for the sequence, $\pi_i^F = P(S_i | O_1, \ldots, O_i, T, E)$ (note that F_i is a vector with one entry per possible state). These probabilities give us the most likely state at each step only taking into account the previous and current observations. Given some initial distribution π_0^F, we can calculate these probabilities iteratively using Equation (1.2), which already computes exactly that

$$\pi_i^F \propto E(O_i)' \circ (\pi_{i-1}^F T) \tag{4.1}$$

We can see example results from the forward algorithm in Figure 4(a). When the observations are streaming in in real-time (so that no future observations are known), such as in the GPS example, this is often the best estimation of state one can get.

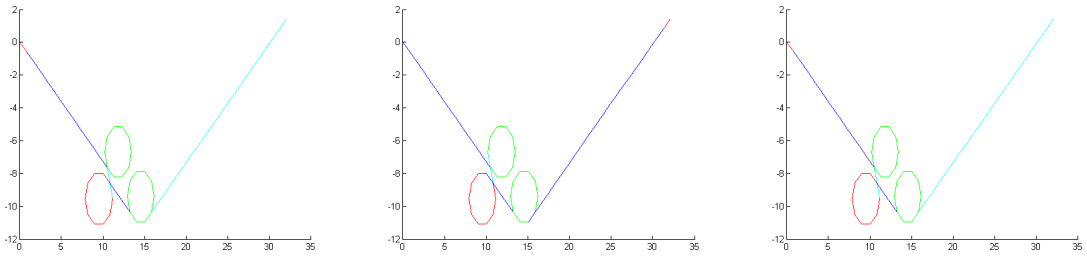
4.3 Backward algorithm.

Second, we compute the backward probabilities, $\pi_i^B = P(S_i | O_{i+1}, \ldots, O_n, T, E)$. These probabilities give us the most likely state at each step only taking into account future observations. We compute these probabilities iteratively as well. No observations come after step n, so we start with the uniform distribution for π_n^B. (In theory, we could try to compute some stationary or prior distribution for π_n^B, but this is not done in practice, since we're mainly interested in the middle of the sequence where it doesn't make a difference anyway.) We can derive this update process similar to how we did for Equation (1.2). Using Bayes theorem, we can compute

$$P(S_{i+1} | O_{i+1}, \ldots, O_n, T, E) \propto P(O_{i+1} | S_{i+1}, O_{i+2}, \ldots, O_n, T, E) P(S_{i+1} | O_{i+2}, \ldots, O_n, T, E) \tag{4.2}$$

which is simply π_{i+1}^B weighted by the emission probabilities for O_{i+1}.

$$P(S_{i+1} | O_{i+1}, \ldots, O_n, T, E) \propto E(O_{i+1}) \circ \pi_{i+1}^B \tag{4.3}$$



(a) Best forward (b) Best backward (c) Best combined

$$T_3 = \begin{pmatrix} 0.95 & 0 & 0.05 & 0 \\ 0 & 0.95 & 0 & 0.05 \\ 0 & 0.05 & 0.95 & 0 \\ 0.05 & 0 & 0 & 0.95 \end{pmatrix} \quad E_3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(d) HMM

Figure 4: The most-likely states given by the forward-backward probabilities over a simple doodle. The color of each line corresponds to the state that generated its starting angle, with red, green, blue, and cyan representing states 1, 2, 3, and 4. The generated sequence alternate between straight lines from states 3 and 4 and different orientation circles from states 1 and 2. Notice how the backwards algorithm gets the state for the final line (going right) because it hasn't seen a circle yet.

Where, again, $\circ$ denotes the element-wise product. And to convert this to the probabilities for S_i, we simply have to apply the transition matrix in reverse.

$$\pi_i^B = P(S_i | S_{i+1}, O_{i+1}, T, E) \tag{4.4}$$

$$\propto T \times (E(O_{i+1}) \circ \pi_{i+1}^B) \tag{4.5}$$

We can see example results from the backward algorithm for the same figure as before in Figure 4(b).

4.4 Combined probabilities.

The combined forward-backward probabilities are then simply the forward probabilities at each step weighted by the corresponding backward probabilities, which then takes into account all of the observations.

$$P(S_i | O, T, E) \propto \pi_i^F \circ \pi_i^B \tag{4.6}$$

We can see an example of these combined results in Figure 4(c). Noticed how the combined result fixes the errors from both the forward algorithm and backward algorithm. Note: don't mistake the output of the forward-backward algorithm to give you the most likely *path*. Due to oddly distributed emission probabilities

and the fact that $P(S_i|O,T,E)$ takes into account all possible ways to reach each state, two adjacent highest-probabilities states S_i and S_{i+1} might have actually have a really low or zero transition probability between them (i.e. $T(S_i, S_{i+1}) = 0$). To compute the most-likely path, we need to use the Viterbi algorithm.

5 Viterbi Algorithm

5.1 Background.

Given a transition matrix T, an emission matrix E, and a sequence of observations $O = \{O_i\}$, the Viterbi algorithm calculates the single most-likely sequence of states $S = \{s_i\}$ given O. That is, it calculates

$$\underset{S}{\operatorname{argmax}} P(S|O,T,E) \tag{5.1}$$

This algorithm gives, in effect, a "most-likely explanation" for the observation data, and is used in applications such as speech recognition where the observations might be sound samples and the observations different phonemes (word-sounds) being said. The Viterbi is a fairly straight-forward algorithm, with its roots in dynamic programming.

5.2 The algorithm.

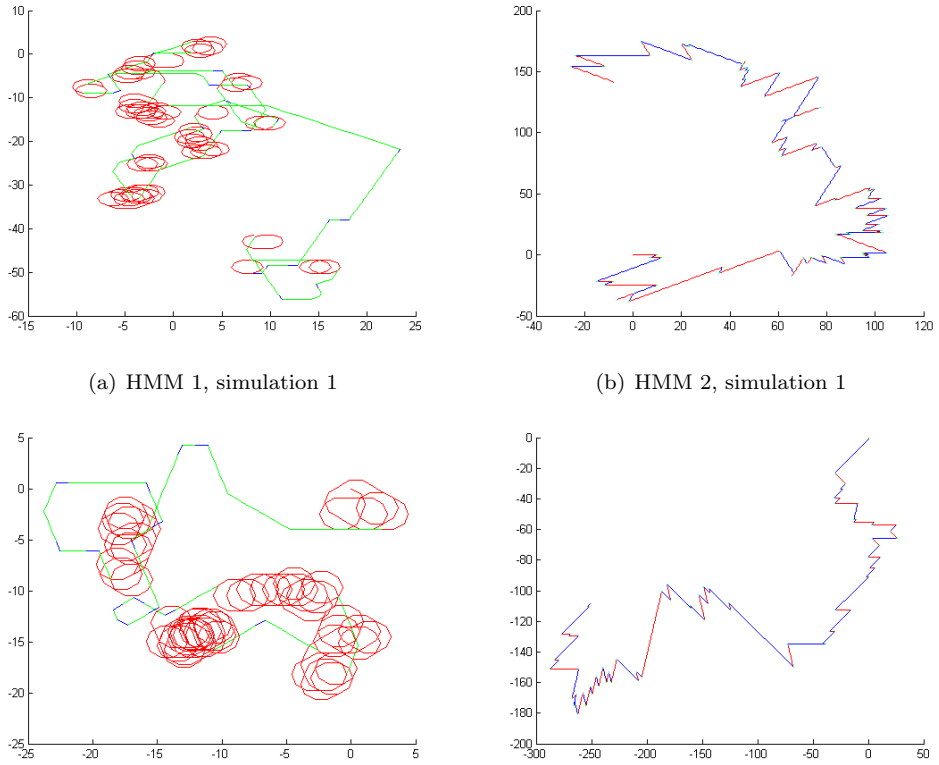
Given some starting distribution π_0, the Viterbi algorithm the most-likely path iteratively, similar to with the forward algorithm. However, instead of calculating π_i as $P(S_i|S_{i-1}, O_i, T, E)$, the total probability of being in each state given the evidence, it instead calculates π_i as

$$[\pi_i]_{s'} = \max_s P(S_i = s'|S_{i-1} = s, O_i, T, E) \tag{5.2}$$

$$\propto \max_s \left(E(s', O_i) \times T(s) \cdot [\pi_{i-1}]_s\right) \tag{5.3}$$

the maximum probability of reaching each state from any one other state. We also keep track of which state s gave the maximal probability for $[\pi_i]_{s'}$ as $B(i, s') = s$.

Once we have calculated π_n, finding the most-likely path is simple. We find the most-likely end state $s_n = \operatorname{argmax}_s [\pi_i]_s$, and for each state s_i we iteratively find the most-likely previous state s_{i-1} by retrieving



(a) HMM 1, simulation 1

(b) HMM 2, simulation 1

(c) HMM 1, simulation 2

(d) HMM 2, simulation 2

Figure 5: The result of running the Viterbi algorithm on the doodles from Figure 3, with the color of the line denoting the state that generated the starting angle. Again, colors red, green, blue, and cyan represent states 1, 2, 3, and 4, respectively. On the left, Viterbi correctly determines that the circles come from state 1, the long lines from state 2, and the sharp corners from state 3. On the right, Viterbi correctly sees the alternating states for the long lines, with the sharp corners attributed to the correct states.

which state led to s_i, i.e. $s_{i-1} = B(i, s_i)$. This process gives us the most-likely path, given our observations O.

$$s_0,\ s_1,\ \ldots,\ s_{n-1},\ s_n \quad = \quad B(1, s_1),\ B(2, s_2),\ \ldots,\ B(n, s_n),\ s_n \tag{5.4}$$

Figure 5 shows the results of running the Viterbi algorithm on the doodles we generated in Section 3. The results are indeed what we would expect given the matrices used to generate them.

6 Baum-Welch Algorithm

6.1 Background.

Given one or more sequences of observations $O = \{O_i\}$, a set of possible states $\mathcal{S}$, and a set of possible observations $\mathcal{O}$ (and some, possibly random, initial guess of a transition matrix T_0 and emission matrix E_0), the Baum-Welch algorithm ties the previous two algorithms together to provide a guess at the most likely matrices $T \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ and $E \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{O}|}$ to generate the sequence O. In general, finding the optimal T and E is an intractable problem, but Baum-Welch uses expectation-maximization to arrive at some locally-optimal guess (which gets better the more training data we have).

6.2 The algorithm.

As an EM-algorithm, Baum-Welch has two steps that are repeated iteratively: the expectation step and the maximization step. In the expectation step, the algorithm calculates what the expected path $S = \{s_i\}$ for each observation sequence O is. This is simply the most-likely path given the current T and E, which is obtained from (5.4) using the Viterbi algorithm.

In the second step, the maximization step, transition probabilities T and emission probabilities E are recalculated using the weighted frequencies of their appearance in each path (S, O). These weighted frequencies are simple the sum of the forward-backward probabilities at the steps where the transition or emission happens. That is, we compute

$$T(s, s') \propto \sum_{\substack{i \\ S_i = s \\ S_{i+1} = s'}} \left[\pi_i^F\right]_s \cdot T(s, s') \cdot \left[\pi_{i+1}^B\right]_{s'} \tag{6.1}$$

$$E(s, o) \propto \sum_{\substack{i \\ S_i = s \\ O_i = o}} \left[\pi_i^F\right]_s \cdot \left[\pi_i^B\right]_s \tag{6.2}$$

where π_i^F and π_i^B are calculated for each sequence using Equations (4.1) and (4.5). We then normalize so the probabilities in each distribution add up to 1. With multiple observation sequences O, the two sums are over each step in each sequence.

After each maximization step, the probability of the expectation (S, O) should be greater than it was before[1].

[1] A formal proof of this is outside the scope of this paper. See *A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains*, by Leonard E. Baum, Ted Petrie, George Soules and Norman Weiss for

These two steps are repeated until either the matrices T and E converge to some local optimum, or some maximum number of iterations is reached. Since T and E are only local optima, the entire Baum-Welch is often repeated using different initial T_0 and E_0, and the resulting T and E with that give the highest probability expectations (S, O) is chosen as the output.

We can see example results from this algorithm in the next section.

7 Analysis with Trained HMMs

7.1 Single-class classification.

One reason trained HMMs are interesting is that they can be used for single-class classification. Most classification algorithms in machine learning require examples of two or more classes and simply seek to distinguish between these classes. HMMs attempt to describe a single class of sequences without requiring negative examples (and, in fact, has no simple way of integrating negative examples). This is useful when the types of things outside your class of interest are far to accurately model.

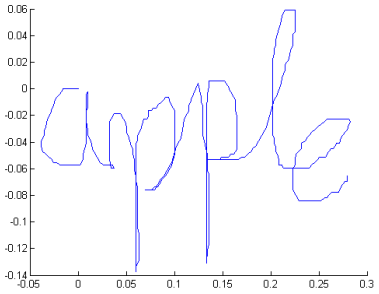
Consider the case of recognizing cursive word doodles. The class cursive words, such as those on the left in Figure 6, has a lot of structure to it, but the class of doodles that are not cursive words, such as those on the right in Figure 6, includes everything from scribbles to stars to elaborate, one-stroke renditions of the Taj Mahal.

By training an HMM on one of the cursive words, we use the probabilities (generated using equation (3.2)) to get a rough estimate for how similar each of the other doodles are to it. We can rank these doodles by their similarity ratings and, with enough data, choose some cutoff value for classifying a doodle as a word. Figure 6 shows example similarity ratings. Notice how the likelihood of the non-words is much lower than the words (especially considering this is log space). If we set the cutoff at 375, we would classify each doodle correctly with a large margin.

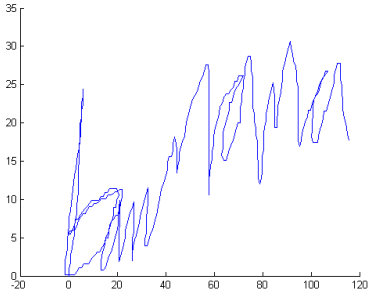
7.2 Difference with bag-of-words.

As an interesting endnote, one might wonder how this HMM technique stacks up with the simpler bag-of-words technique. With bag-of-words, one generates histograms from the observations found in the training

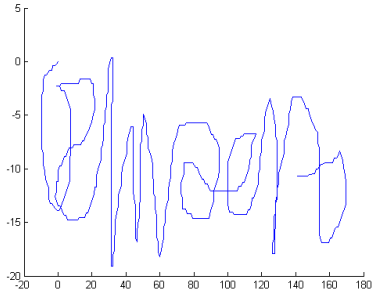
a more formal analysis of this type technique.



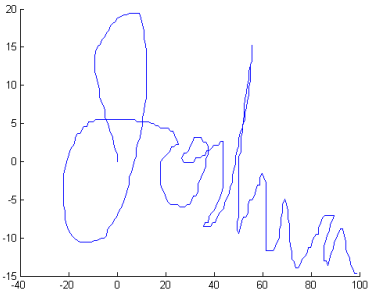
(a) Likelihood: -1.1582 (Training example)



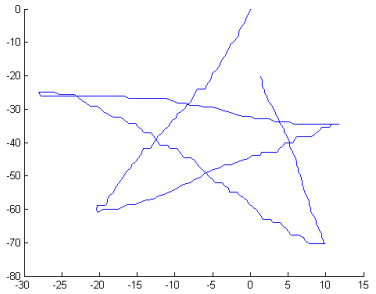
(b) Likelihood: -347.4649



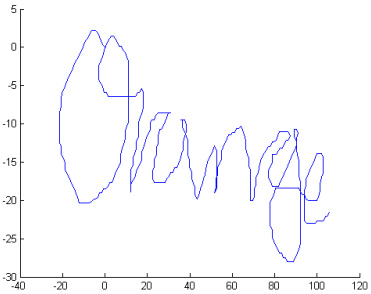
(c) Likelihood: -428.4324



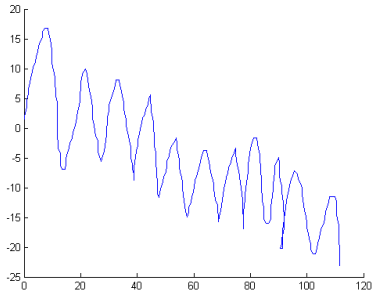
(d) Likelihood: -271.8618



(e) Likelihood: -415.9992

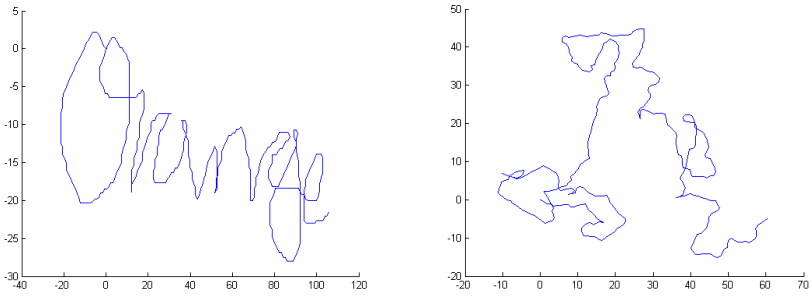


(f) Likelihood: -129.2902



(g) Likelihood: -415.7359

Figure 6: Example words (left) and non-words (right), along with their associated log likelihood (similarity) normalized by the length of the sequence. The HMM was trained on doodle (a) with 10 states and the angles were divided into 7 bins.



(a) A word (b) Definitely not a word

Figure 7: An example word (a) (likelihood -129.2902), and that same word with its angles scrambled (b) (likelihood -412.0987).

data and compares that with the histograms of those in what their testing. With our cursive-word doodle example above, this would amount to taking a histogram of the angles in the training word and comparing that to histograms of the angles in the other doodles, and then ranking them using some histogram distance function.

This technique is significantly simpler than the HMM approach (the hardest decision is likely the choice of distance function), and may work just fine if the distribution of doodles or other things you need to test is nice. However, it disregards any information supplied by the order the angles are in, which can lead to odd situations such as in Figure 7. Both doodles have the same angles in them, but the one on the right had its angles shuffled randomly. Our HMM model from before can easily tell the difference between the two doodles, ranking the one on the left as a lot more likely. But bag-of-words would have given them the same rank, since they give identical histograms. Bag-of-words is simpler and faster, but in the end HMMs can be more powerful.