67750: Advanced Practical Machine Learning 2019/20

Hidden Markov Models

TA: Omri Bloch

1 Hidden Markov Models

1.1 Motivation

Hidden Markov Models are a model based inference tool that is often used to reason about sequential information, where each part of the information stream can tell us something about the other parts. Usually we assume there is some hidden process which we are interested in, but we only see observations emanating from that process.

A few examples of uses of HMMs:

1. **Speech Recognition** - we receive a stream of sound waves and we want to able to understand what is being said. The actual words and phonemes are the hidden process, and we only get to see the sound waves.
2. **Robot Localization** - we have a robot which receives inputs from its sensors each time interval as it moves, and it needs to understand where it is within the house. The location of the robot in time is the hidden process, and we only get to see the input from the sensors (say, the distance from the walls in each direction).
3. **Parts-of-Speech Tagging** - we receive a sentence and we want to be able to label the words with the right PoS tags. The transition between parts-of-speech is our hidden process (some transitions are more likely than others), and we only get to see the words in the sentence. This sounds funny, because we model the world as if we had a transitions between parts-of-speech where each part-of-speech generated a word, which is clearly not how language works. Still, we will see that this kind of model is able to give us surprisingly good results.

1.2 Reminder

1.2.1 Bayes

$$P(A|B) := \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

1.2.2 Maximum a posteriori

Given a sample $x \sim D_y$, the posterior probability is defined as $P(y|x)$. from bayes rule:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

$P(y)$ is called the prior about y, and $P(x), P(y)$ are marginal probabilities.
Let say we want to estimate the parameter y, we can do that by maximizing the posterior:

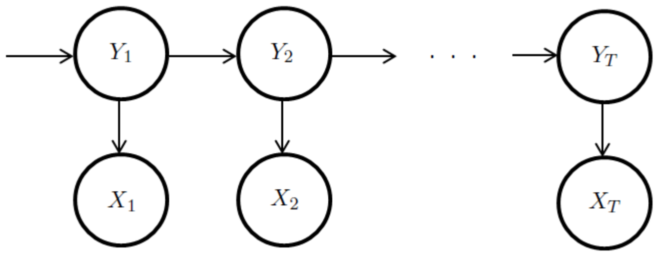
$$\hat{y}_{map} = \arg\max_y P(y|x) = \arg\max_y \frac{P(x|y)P(y)}{P(x)} = \arg\max_y \frac{P(x|y)P(y)}{\int P(x|y')\,dy'} = \arg\max_y P(x|y)P(y)$$

When we have no prior, it is the same as using uniform prior. in this case the MAP is equivalent to the maximum likelihood estimator:

$$\hat{y}_{mle} = \arg\max_y P(x|y)$$

1.3 Assumptions

A Hidden Markov Model is a model where we assume that the sequential data that we want to find out is part of a Markov Chain, but we don't get to observe the actual states - we only see partial evidence emitted by the states.



In order to model some process as a Hidden Markov Model, we need to make the following assumptions:

1. We can model transitions between hidden states as a Markov Chain with the Markov property ($\mathbb{P}(y_t|y_{1:t-1}) = \mathbb{P}(y_t|y_{t-1})$).
2. We can model the emission of an observation from a specific (hidden) state, and that emission depends only on the hidden state which emitted it ($\mathbb{P}(x_t|y_{1:T}, x_{1:t-1}, x_{t+1:T}) = \mathbb{P}(x_t|y_t)$).
3. Usually, it is also assumed that the process is homogeneous (constant in time).

1.4 Notation and Parameters

We will denote the hidden random variables as Y_t and the possible states as integers. We add a "START" state, named y_0, which does not emit an observation. We also add an "END" state, named y_{end}, which also doesn't emit an observation. The possible observations are also random variables that we will denote as X_t. If we omit the time index, we will be referring to the entire series of variables.

The model can now be described with two distributions:

$$t_{i,j} = \mathbb{P}(Y_t = j | Y_{t-1} = i)$$

$$e_{i,x_j} = \mathbb{P}(X_t = x_j | Y_t = i)$$

This gives us the following probability function (we clump all of the probability distributions of the model into a parameter vector Θ):

$$\mathbb{P}(X_{1:T}, Y_{1:T}; \Theta) = \prod_{t=1}^{T} t_{y_{t-1},y_t} \prod_{t=1}^{T} e_{y_t,x_t}$$

2 HMM queries

What questions do we typically ask an HMM (or any model for that matter), given an observation X?

1. *What is the probability of the observation? (Likelihood query)* given the model, what's the chance to see the observations we're seeing:

$$\mathbb{P}(X = x;\, \Theta)$$

2. *What is the probability for some hidden states given the observations? (Posterior query)* given the model and the observations, what is the probability that the data was generated from these hidden states:

$$\mathbb{P}(Y = (y_1 ... y_T) | X;\, \Theta)$$

3. *What is the most probable assignment of all the hidden states, given an observation? (MAP query)* if we believe the model, and the data, what are the most probable hidden states:

$$\underset{y}{argmax}\big(\mathbb{P}(Y = (y_1 ... y_T) | X;\, \Theta)\big)$$

Can we calculate these things efficiently?

The nice thing about HMMs and the Markov property in general, is that the chain structure allows us to create recursive formulas for most calculations, which then lead to nice dynamic programming algorithms that can calculate summations and maximizations over exponentially large state spaces.

2.1 Viterbi Algorithm for MAP Estimation

In class, we saw a reduction of the MAP estimation problem to a shortest weighted path problem, which led to a nice dynamical programming algorithm (Viterbi). We will now derive the Viterbi algorithm algebraically.

Assume we have a HMM and an output sequence of length 4 and we want to calculate the most probable input sequence. Naively, this is by definition:

$$y = \underset{y_1,y_2,y_3,y_4}{argmax}\left(\mathbb{P}(y_{1:4} | x_{1:4})\right)$$

At the moment, this calculation is exponential in T, since we have to maximize over all of the possible assignments of all of the hidden variables. Luckily, we can use the Markov property and take advantage of the chain structure of our model:

$$\underset{y_1,y_2,y_3,y_4}{argmax}\left(\mathbb{P}(y_{1:4}|x_{1:4})\right) = \underset{y_1,y_2,y_3,y_4}{argmax}\left(\mathbb{P}(x_{1:4}|y_{1:4})\mathbb{P}(y_{1:4})\right) = \underset{y_1,y_2,y_3,y_4}{argmax}\prod_{i=1}^{4}\mathbb{P}(y_i|y_{i-1})\mathbb{P}(x_i|y_i) =$$

$$= \underset{y_1,y_2,y_3,y_4}{argmax}\,\mathbb{P}(y_4|y_3)\mathbb{P}(y_3|y_2)\mathbb{P}(y_2|y_1)\mathbb{P}(y_1|y_0)\mathbb{P}(x_4|y_4)\mathbb{P}(x_3|y_3)\mathbb{P}(x_2|y_2)\mathbb{P}(x_1|y_1)$$

Now, we can push some of the argmaxes over the hidden variables inside the product:

$$= \underset{y_4}{argmax}\,\mathbb{P}(x_4|y_4)\,\underset{y_3}{argmax}\,\mathbb{P}(y_4|y_3)\mathbb{P}(x_3|y_3)\,\underset{y_2}{argmax}\,\mathbb{P}(y_3|y_2)\mathbb{P}(x_2|y_2)\,\underset{y_1}{argmax}\,\mathbb{P}(y_2|y_1)\mathbb{P}(x_1|y_1)\mathbb{P}(y_1|y_0)$$

Now, as long as we do the maximization in the right order, we are able to reduce the complexity of our calculations to being $O(|Y|^2)$ instead of $O(|Y|^T)$.

2.1.1 Viterbi Reminder

We can formalize the above as a dynamic programming algorithm, called the "Viterbi algorithm". We'll define the following value which will be the table that we fill:

$$\pi_t(i) = \underset{\{y|y_t=i\}}{max}\left(\mathbb{P}(y_{1:t}|x_{1:t})\right)$$

In words, $\pi_t(i)$ is the maximum probability among all possible sequences of hidden states up to time t that end in $y_t = i$. We can calculate these values recursively:

$$\pi_t(i) = \underset{j}{max}(\pi_{t-1}(j)t_{j,i}e_{i,x_t})$$

The initial value for π_1 is also simple using the definition:

$$\pi_1(i) = \underset{\{y|y_1=i\}}{max}\left(\mathbb{P}(y_1|x_1)\right) \propto t_{y_0,i}e_{i,x_1}$$

So, to calculate the MAP assignment we just need to calculate the π variables while saving the argmax in every stage. Once they are all calculated, we just go backwards from T to 1 and retrieve the argmaxes to get the MAP assignment.

This doesn't have to be coded recursively: we can initialize 2 tables (one for probabilities, one for argmax) and fill them while iterating over t.

2.1.2 Viterbi with Log Probabilities

The fun we had with log probabilities in exercise 1 is back...

Since the emission probabilities in PoS tagging are small (since there are many possible words to observe), we will need to use log probabilities to avoid numerical problems. Luckily, it is easy to convert the Viterbi algorithm to work with log probabilities:

$$\pi_t(i) = \max_{\{y|y_t=i\}} \Big(log\big(\mathbb{P}(y_{1:t}|x_{1:t})\big)\Big)$$

$$\pi_t(i) = \max_j(\pi_{t-1}(j) + log(t_{j,i}) + log(e_{i,x_t}))$$

$$\pi_1(i) = log(t_{y_0,i}) + log(e_{i,x_1})$$

3 Learning The Model

So far we assumed that we know what the transition and emission probabilities are, but how do we learn them given training data?

For this we have the tool that we often use when learning a probability distribution, and that is the MLE. In a HMM we assume that all the probability distributions are multinomial, which makes for a relatively simple log-likelihood function. our training set $S = \{x^{(i)}, y^{(i)}\}_{i=1}^N$ is pairs of sequences of observations and their corresponding hidden states. We are looking for parameters t and e which maximize the log likelihood of our data:

$$\ell(S,\theta) = \sum_{i=1}^N log(p(y^{(i)}, x^{(i)}; \theta)) = \sum_{i=1}^N log(\prod_{t=1}^{|x^{(i)}|} t_{y_{t-1}^{(i)}, y_t^{(i)}} \prod_{t=1}^{|x^{(i)}|} e_{y_t^{(i)}, x_t^{(i)}})$$

$$\ell(S,\theta) = \sum_{i=1}^N \sum_{t=1}^{|x^{(i)}|} log(t_{y_{t-1}^{(i)}, y_t^{(i)}}) + \sum_{i=1}^N \sum_{t=1}^{|x^{(i)}|} log(e_{y_t^{(i)}, x_t^{(i)}})$$

Now that we've separated the two parameters we want to estimate from one another, we can maximize over each of them separately (exercise).

$$\hat{t}_{i,j} = \frac{\#(y_i \to y_j)}{\sum_k \#(y_i \to y_k)}$$
$$\hat{e}_{i,x_j} = \frac{\#(y_i \to x_j)}{\sum_x \#(y_i \to x)}$$

And this makes sense, as MLEs usually do... We just count the relevant quantities, normalize and that's our MLE.

4 Maximum Entropy Markov Model

In class we introduced an additional model for PoS tagging - the MEMM. Instead of using only transitions and emissions to model a sentence, it uses a general mapping from a transition and emissions to a feature space. This allows us to take advantage of word prefixes and suffixes, along with capitalization of letters and anything else.

For a mapping $\phi : \mathcal{X}^T \times \mathcal{Y} \times \mathcal{Y} \times \mathbb{R} \to \mathbb{R}^d$, the probability distribution of the model is parameterized by a vector $w \in \mathbb{R}^d$ and is written as:

$$\mathbb{P}_w(y_{1:T}|x_{1:T}) = \prod_t \mathbb{P}_w(y_t|y_{t-1}, x_{1:T}) = \prod_t \frac{e^{w^T \phi(x_{1:T}, y_{t-1}, y_t, t)}}{Z(x_{1:T}, y_{t-1}, t)}$$

$$Z(x_{1:T}, y_{t-1}, t) = \sum_{y_t} e^{w^T \phi(x_{1:T}, y_{t-1}, y_t, t)}$$

Note that this model is discriminative and not generative - we do not model the joint probability $\mathbb{P}(x, y)$ but rather only model the conditional probability $\mathbb{P}(y|x)$.

4.1 MAP Estimation

The MEMM has the same Markov property that the HMM does, and so the MAP estimation problem is also performed with the Viterbi algorithm. Instead of the transition and emission probabilities we had with HMMs, we have to use the MEMM probabilities which combine transition and emission together:

$$\pi_t(i) = \underset{\{y|y_t=i\}}{max} (\mathbb{P}(y_{1:t}|x_{1:T}))$$

$$\pi_t(i) = \underset{j}{max}\big(\pi_{t-1}(j)\mathbb{P}(y_t = i|y_{t-1} = j, x_{1:T})\big) = \underset{j}{max}\big(\pi_{t-1}(j)\frac{e^{w^T \phi(x_{1:T}, y_j, y_i, t)}}{Z(x_{1:T}, y_j, t)}\big)$$

$$\pi_1(i) = \frac{e^{w^T \phi(x_{1:T}, y_0, y_i, 1)}}{Z(x_{1:T}, y_0, 1)}$$

Like in your HMM implementation, you will need to work in log space here as well...

4.2 Learning w

This model is more expressive than HMMs, but this comes at a cost - there isn't a simple analytical solution for maximizing the log likelihood. We will use a gradient based approach - the perceptron algorithm.

The point of the perceptron algorithm is that you don't need to calculate the gradients. We are using our prediction to approximate the gradient directly, without backpropagation, so we can use non-differentiable module in our model. it's drawback is that it restrict the model to be linear.

You will see a nice example of such a model in the lecture, together with the perceptron algorithm itself. meanwhile, lets calculate the gradients of this function so we have a baseline to compare with the perceptron approximation.

4.2.1 Calculating the Gradient

We'll begin by calculating the gradient of the log likelihood:

$$\ell(S,w) = \sum_i \sum_t \left(w^T \phi(x_{1:T}^{(i)}, y_{t-1}^{(i)}, y_t^{(i)}) - log(Z(x_{1:T}^{(i)}, y_{t-1}^{(i)})) \right)$$

Note that what is giving us problems here, in the same way as for GMMs, is the sum inside the log. Since we are going to sample a single sentence for each update, we'll simplify our notation and only look at a single example from now on:

$$\frac{\partial \ell}{\partial w} = \sum_t \left(\phi(x_{1:T}, y_j, y_i) - \frac{\partial}{\partial w} log(Z(x_{1:T}, y_{t-1})) \right)$$

Focusing now on the derivative of the log of the partition function for a single transition:

$$\frac{\partial}{\partial w} log(Z(x_{1:T}, y_{t-1})) = \frac{\partial}{\partial w} log(\sum_{y_t} e^{w^T \phi(x_{1:T}, y_{t-1}, y_t)}) = \frac{\sum_{y_t} \phi(x_{1:T}, y_{t-1}, y_t) e^{w^T \phi(x_{1:T}, y_{t-1}, y_t, t)}}{\sum_{y_t} e^{w^T \phi(x_{1:T}, y_{t-1}, y_t, t)}}$$

This looks ugly, but it has a really nice interpretation if we rearrange things a bit:

$$= \sum_{y_t} \phi(x_{1:T}, y_{t-1}, y_t) \frac{e^{w^T \phi(x_{1:T}, y_{t-1}, y_t, t)}}{Z(x_{1:T}, y_{t-1}, t))} = \sum_{y_t} \phi(x_{1:T}, y_{t-1}, y_t) \mathbb{P}_w(y_t | y_{t-1}, x_{1:T}) = \mathbb{E}_w[\phi(x_{1:T}, y_{t-1}, y_t) | y_{t-1}, x_{1:T}]$$

So we see that the derivative of the partition function is simply its expected input, according to the current model. So we can write the complete gradient as:

$$\frac{\partial \ell}{\partial w} = \sum_t \left(\phi(x_{1:T}, y_{t-1}, y_t) - \mathbb{E}_w[\phi(x_{1:T}, y_{t-1}, y_t) | y_{t-1}, x_{1:T}] \right)$$

So in words, the gradient of the log likelihood of our model when looking at real data is the difference between the feature vectors of the real data and the feature vectors that the model expects to see!

5 PoS Tagging

The HMM and MEMM are useful models for PoS tagging, but there are a couple of practical considerations we need to go over before we can use these models well.

5.1 Handling Rare Words

For HMMs, according to the way our model was learned using MLE, the probability of any PoS to emit a word we've never seen before is 0. This means that all possible PoS taggings will have a probability of 0 if there is a new word in a given sentence... For MEMMs, ϕ isn't well defined for a word that was never seen before. So we need to change our model a little to account for this problem.

We would like to introduce a probability for each PoS to emit a word that was never seen before. There are better ways to do this, but what we will do for simplicity is to define a new possible observation which will correspond to “rare” words. Before learning the model, we will process our training data so that words that appear less than n times will be replaced by the “rare” observation (n depends on the size of the training data, but for us it will be small, between 2 and 5). We will then learn the model as usual, but now when we want to calculate the MAP estimation using Viterbi, we will first process the given observation (sentence) to make sure that words that are not in our dictionary will be replaced by the “rare” observation.

5.2 Implementing the MEMM using Sparse Data Structures

So far, we didn't define what the ϕ mapping is. The basic mapping we will use for the MEMM will be an indication mapping, that maps a transition and emission to a binary vector which indicates which transition and emission we are calculating the probability for. This means that d will be very large, and contain an element for every possible transition and every possible emission. This means that d will be at least $|Y|^2 + |Y||X|$...

This means that the feature vectors will be incredibly sparse, with only two non-zeros and tens of thousands of zeros.

It would be silly to actually map the data to a huge vector and then take the inner product with w, since almost all of the operation will result in zero. To make things more efficient, don't actually calculate the inner product $w^T\phi$ and instead use simple indexing to get the non zero values and calculate the sum of their products with the relevant indices of w...