CSC411 HW6

Casey Juanxi Li - 998816973

Nov 20 2018

1 Learning the parameters. (The "M" part of E-M)

1. M-step update rules for Θ and π:

$$\begin{aligned}
\pi_k &\leftarrow ? \\
\log(\text{posterior prob, wrt } \pi) &= \sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} \left[\log \Pr\left(z^{(i)}=k\right) + \log p\left(\mathbf{x}^{(i)}|z^{(i)}=k\right)\right] + \log p(\boldsymbol{\pi}) + \log p(\Theta) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} \left[\log \Pr\left(z^{(i)}=k\right)\right] + \log p(\boldsymbol{\pi}) \qquad \text{(remove parts that don't depend on } \pi) \\
&= \left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} log(\pi_k)\right) + log(\prod_{k=1}^{K} \pi_k^{a_k-1}) \\
&= \left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} log(\pi_k)\right) + \sum_{k=1}^{K} log(\pi_k^{a_k-1})
\end{aligned}$$

Now we:

- Add the Lagragian constraint that all π must sum to one.
- To make math easier, we'll take the derivative wrt a specific π_k, knowing that all other π_k can be found in the same way. This allows us to get rid of the $\sum_k$.
- Lastly, we remember that for the purpose of this step, we get to treat $r_k^{(i)}$ as a constant, since it gets updated later in the E step.

So now we have:

$$\begin{aligned}
&\left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} log(\pi_k)\right) + \sum_{k=1}^{K} log(\pi_k^{a_k-1}) - \lambda(\pi_1+\pi_2+...+\pi_k-1) \\
\frac{d}{d\pi_k} &= \left(\sum_{i=1}^{N} r_k^{(i)} \frac{1}{(\pi_k)}\right) + \frac{a_k-1}{\pi_k} - \lambda \\
0 &= \frac{(\sum_{i=1}^{N} r_k^{(i)}) + a_k + 1}{\pi_k} - \lambda \\
\pi_k &= \frac{(\sum_{i=1}^{N} r_k^{(i)}) + a_k + 1}{\lambda}
\end{aligned}$$

Sub into Lagrangian constraint:

$$\begin{aligned}
\sum_k \frac{(\sum_{i=1}^{N} r_k^{(i)}) + a_k + 1}{\lambda} &= 1 \\
\sum_{k=1}^{K}(\sum_{i=1}^{N} r_k^{(i)}) + a_k + 1 &= \lambda
\end{aligned}$$

$$\boxed{\pi_k = \frac{(\sum_{i=1}^{N} r_k^{(i)}) + a_k + 1}{\sum_{k=1}^{K}\left[(\sum_{i=1}^{N} r_k^{(i)}) + a_k + 1\right]}}$$

Now θ:

$$\theta_{k,j} \leftarrow ?$$

$$\begin{aligned}
\log(\text{posterior prob, wrt } \theta) &= \sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} \left[\log \Pr\left(z^{(i)}=k\right) + \log p\left(\mathbf{x}^{(i)}|z^{(i)}=k\right)\right] + \log p(\boldsymbol{\pi}) + \log p(\Theta) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} \left[\log p\left(\mathbf{x}^{(i)}|z^{(i)}=k\right)\right] + \log p(\Theta) \qquad \text{(remove parts that don't depend on } \theta) \\
&= \left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} log(\prod_{j=1}^{D} \theta_{k,j}^{x_j^{(i)}} (1-\theta_{k,j})^{1-x_j^{(i)}})\right) + log(\prod_{k=1}^{K} \theta_{k,j}^{a-1}(1-\theta_{k,j})^{b-1}) \\
&= \left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} \sum_{j=1}^{D} log(\theta_{k,j}^{x_j^{(i)}} (1-\theta_{k,j})^{1-x_j^{(i)}})\right) + \sum_{k=1}^{K} log(\theta_{k,j}^{a-1}(1-\theta_{k,j})^{b-1})
\end{aligned}$$

- We don't have to use the Lagrangian here, but let's conveniently define $\rho = (1-\theta)$ with the Lagrangian constraint that $\rho + \theta = 1$
- To make math easier, we'll take the derivative wrt a specific $\theta_{k,j}$, knowing that all other θ can be found in the same way. This allows us to get rid of $\sum_k$ and $\sum_j$.
- And we still get to treat $r_k^{(i)}$ as a constant.

$$\left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_k^{(i)} \sum_{j=1}^{D} log(\theta_{k,j}^{x_j^{(i)}} (\rho)^{1-x_j^{(i)}})\right) + \sum_{k=1}^{K} log(\theta_{k,j}^{a-1}(\rho)^{b-1}) - \lambda(\theta_{k,j} + \rho_{k,j} - 1)$$

$$\begin{aligned}
\frac{d}{d\theta_{k,j}} &= \left(\sum_{i=1}^{N} \frac{r_k^{(i)} \times x_j^{(i)}}{\theta_{k,j}}\right) + \frac{a-1}{\theta_{k,j}} - \lambda \\
\frac{d}{d\rho_{k,j}} &= \left(\sum_{i=1}^{N} \frac{r_k^{(i)} \times (1-x_j^{(i)})}{\rho_{k,j}}\right) + \frac{b-1}{\rho_{k,j}} - \lambda
\end{aligned}$$

Setting both derivatives to 0 we get:

$$\begin{aligned}
\left(\sum_{i=1}^{N} \frac{r_k^{(i)} \times x_j^{(i)}}{\theta_{k,j}}\right) + \frac{a-1}{\theta_{k,j}} &= \lambda \\
\left(\sum_{i=1}^{N} \frac{r_k^{(i)} \times (1-x_j^{(i)})}{\rho_{k,j}}\right) + \frac{b-1}{\rho_{k,j}} &= \lambda \\
\frac{\left(\sum_{i=1}^{N} r_k^{(i)} \times x_j^{(i)}\right) + a - 1}{\lambda} &= \theta_{k,j} \\
\frac{\left(\sum_{i=1}^{N} r_k^{(i)} \times (1-x_j^{(i)})\right) + b - 1}{\lambda} &= \rho_{k,j}
\end{aligned}$$

Then sub into the Lagrangian constraint:

$$\frac{\left(\sum_{i=1}^{N} r_k^{(i)} \times x_j^{(i)}\right) + a - 1}{\lambda} + \frac{\left(\sum_{i=1}^{N} r_k^{(i)} \times (1-x_j^{(i)})\right) + b - 1}{\lambda} = 1$$

$$\lambda = \left(\sum_{i=1}^{N} r_k^{(i)} \times x_j^{(i)}\right) + a - 1 + \left(\sum_{i=1}^{N} r_k^{(i)} \times (1-x_j^{(i)})\right) + b - 1$$

$$\boxed{\theta_{k,j} = \frac{\left(\sum_{i=1}^{N} r_k^{(i)} \times x_j^{(i)}\right) + a - 1}{\left(\sum_{i=1}^{N} r_k^{(i)} \times 1\right) + a - 1 + b - 1}}$$

2. Implement `Model.update_pi` and `Model.update_theta`.

   Output of `mixture.print_part_1_values()`:

```
pi[0] 0.08499999999999992
pi[1] 0.1299999999999999
theta[0, 239] 0.4229404029304033
theta[3, 298] 0.2962545995347294
```

2 Posterior Inference

1. Derive the rule for computing the posterior probability distribution $p(z|x)$.

   I'll write this for a specific k, conditioned on a specific $x^{(i)}$. We'd repeat a similar calculation for all N images and for all k components in order to get the N x k matrix R.

$$
\begin{aligned}
R_{i,k} = p(z = k|x^{(i)}) &= \frac{p(x|z = k) \cdot p(z = k)}{\sum_{k'} [p(x|z = k') \cdot p(z = k')]} \\
&= \frac{(\prod_{j=1}^{D} \theta_{k,j}^{x_j^{(i)}} (1 - \theta_{k,j})^{1-x_j^{(i)}}) \times \pi_k}{\text{(sum of numerator, evaluated over all k)}}
\end{aligned}
$$

   To account for how the red part became a dot product in `Model.log_likelihood()`, note that:

$$
\begin{aligned}
log(\prod_{D} \theta^x (1 - \theta)^{(1-x)}) &= \sum_{D} log(\theta^x (1 - \theta)^{(1-x)}) \\
&= \sum_{D} log(\theta^x) + log((1 - \theta)^{(1-x)}) \\
&= \sum_{D} \big(x \cdot log(\theta) + (1 - x) \cdot log((1 - \theta))\big)
\end{aligned}
$$

   Lastly, the M matrix indicates whether a pixel is observed or not. When $M_j = 0$ we want pixel j to have no impact on the probability (@872). When M is 0, we want both θ and $(1 - \theta)$ to be put to power of 0, in order to equal 1.

   This can be achieved by slightly altering the term in the $\prod_D$ to this:

$$
\prod_{D} \theta_{k,j}^{min(x_j^{(i)}, M)} (1 - \theta_{k,j})^{min(1-x_j^{(i)}, M)}
$$

2. Implement `Model.compute_posterior`. Steal this from `Model.log_likelihood()`, but create the `min()` terms from above as matrices and throw those into the dot product instead:

```
one_minus_X = 1. - X

min_X = np.minimum(M, X)
min_one_minus_X = np.minimum(M, one_minus_X)

log_p_x_given_z = np.dot(min_X, np.log(self.params.theta).T) + \
                  np.dot(min_one_minus_X, np.log(1. - self.params.theta).T)
```

3. Implement `Model.posterior_predictive_means`.

   Result of `mixture.print_part_2_values()`:

```
R[0, 2] 0.17511938747691308
R[1, 0] 0.6888326410301712
P[0, 183] 0.6519638229599034
P[2, 628] 0.4733939144603015
```

3 Conceptual Questions.

1. If we instead used a = b = 1 (which corresponds to a uniform distribution), the MAP learning algorithm would have the problem that it might assign zero probability to images in the test set. Why might this happen?

   Using the hint, examine the expression for the posterior probability of a component being true, given a specific image:

$$R_{i,k} = p(z = k|x^{(i)}) = \frac{p(x|z = k) \cdot p(z = k)}{\sum_{k'} [p(x|z = k') \cdot p(z = k')]}$$
$$= \frac{(\prod_{j=1}^{D} \theta_{k,j}^{x_j^{(i)}} (1 - \theta_{k,j})^{1-x_j^{(i)}}) \times \pi_k}{\text{(sum of numerator, evaluated over all k)}}$$

   Focusing just on the numerator, we can see that this quantity would be zero if there is a $x_j^{(i)} = 1$ which gets matched up with a $\theta_{k,j} = 0$. Or conversely, a $(1 - x_j^{(i)}) = 1$ which gets matched up with a $(1 - \theta_{k,j}) = 0$.

   Noting that θ is calculated as:

$$\theta_{k,j} = 0 = \frac{\left(\sum_{i=1}^{N} r_k^{(i)} \times x_j^{(i)}\right) + a - 1}{\left(\sum_{i=1}^{N} r_k^{(i)} \times 1\right) + a - 1 + b - 1}$$

   When is $\theta_{k,j} = 0$? If pixel $x_j^{(i)}$ is never "on" in the train set, we are just left with a - 1 in the numerator. If a = 1, then the numerator is 0.

   Thus, if we encounter a pixel that is never "on" in the training set, but is on in the test image, and we chose Beta(1,1) as a prior, the image is given a probability of zero. The converse is also true for an pixel that is always "on" ($\theta = 1$) in the train set but is "off" in a test image.

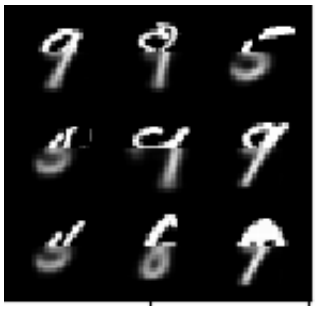
   This sort of makes sense, since Beta(1,1) is an uninformative prior. If we have zero previous information about a certain pixel, and no info from the prior either, it makes sense to say that the pixel is "infinitely unlikely" to be on.

2. The model from Part 2 gets significantly higher average log probabilities on both the training and test sets, compared with the model from Part 1. This is counterintuitive, since the Part 1 model has access to additional information: labels which are part of a true causal explanation of the data (i.e. what digit someone was trying to write). Why do you think the Part 2 model still does better?

   Even within a class of digits, you still have variations in structure: numbers can be fat/skinny, angled, etc.

   In the labeled case, we only have 10 classes to work with. The mean of all the 9's might not actually look like any particular 9. This is evident from the following example of a disjoint prediction for the bottom half:

   Contrast this with the EM model where we used more than 10 components - in fact, the default was 100. 100 components gives us many more "buckets" within which we can converge towards θ_k's that describe a particular configuration of a number. We can now have a set of thetas that describe round 9s, another set that describes slanty 9s, etc.

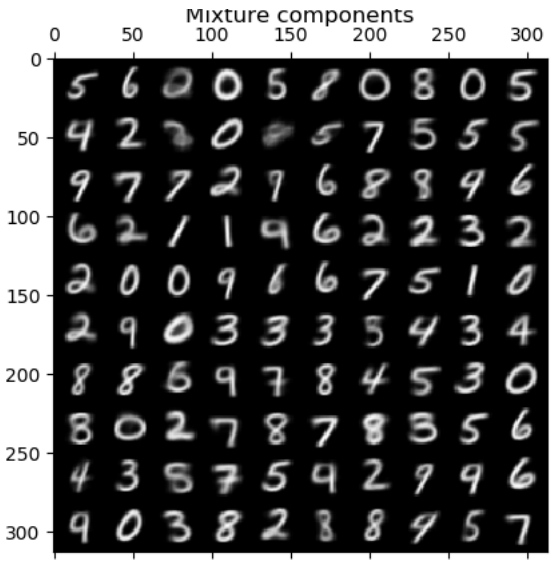


   The additional freedom provided by these 100 components allows us to capture the idiosyncrasies within a class, meaning that variation within each component is reduced.

3. The function `print_log_probs_by_digit_class` computes the average log-probabilities for different digit classes in both the training and test sets. In both cases, images of 1's are assigned far higher log-probability than images of 8's. Does this mean the model thinks 1's are far more common than 8's? I.e., if you sample from its distribution, will it generate far more 1's than 8's? Why or why not?

   No, 1s are not more common. This does indicate, however, that 1's **show less variation.**

   We can verify this with a little bit of additional work, which I've placed under `for_q3(model, X, y)`. Here we use logistic regression to classify the components into one of 10 classes and see what's going on.




```
Training set
Average log-probability of a 0 image: -157.831
Average log-probability of a 1 image: -70.658
Average log-probability of a 2 image: -169.503
Average log-probability of a 3 image: -155.233
Average log-probability of a 4 image: -141.944
Average log-probability of a 5 image: -150.773
Average log-probability of a 6 image: -139.159
Average log-probability of a 7 image: -122.627
Average log-probability of a 8 image: -157.694
Average log-probability of a 9 image: -128.442
...
Extra info for conceptual Q3
Fitting logistic reg. model to classify components...
0 sum of pis: 0.10 | avg sum_theta: 136.216 | comps in class: 11
1 sum of pis: 0.10 | avg sum_theta: 57.093  | comps in class: 3
2 sum of pis: 0.09 | avg sum_theta: 118.390 | comps in class: 12
3 sum of pis: 0.10 | avg sum_theta: 109.757 | comps in class: 11
4 sum of pis: 0.13 | avg sum_theta: 98.842  | comps in class: 12
5 sum of pis: 0.09 | avg sum_theta: 100.507 | comps in class: 13
6 sum of pis: 0.10 | avg sum_theta: 106.134 | comps in class: 10
7 sum of pis: 0.10 | avg sum_theta: 91.946  | comps in class: 11
8 sum of pis: 0.10 | avg sum_theta: 118.716 | comps in class: 12
9 sum of pis: 0.08 | avg sum_theta: 83.665  | comps in class: 5
```

   We can see that in this case, only 3 of the "components" corresponded to a 1-like thing, while we managed to converge to 13 different components that were 5-like.

   However, the π's of all the different classes are pretty much the same.

   If we think of the dot product as a measure of similarity, and then look at the calculation for log-likelihood:

```
log_p_x_given_z = np.dot(X, np.log(self.params.theta).T) + \
                          np.dot(1. - X, np.log(1. - self.params.theta).T)
```

   Handwaving a bit - but if the universe of 1's tends to look similar to the "typical" one that is encoded in θ, it makes sense that its dot products with its corresponding θs will be higher than that for a number which shows more variation.