CS 475 Machine Learning: Homework 4
EM and Graphical Models

Due: Monday November 18, 2019, 11:59pm

100 Points Total Version 1.4

Make sure to read from start to finish before beginning the assignment.

1 Programming (50 points)

You will implement a Markov Random Field (MRF) for image segmentation. Please see section 19.4.3 of Kevin Murphy's textbook for a description of the model. We will be providing a Python model framework/skeleton as before. Remember: **do not change the names of any of the files or command-line arguments.**

1.1 Image Segmentation using MRFs

The goal of image segmentation is to segment the objects displayed in an image into K different groups. Given a grayscale image containing S pixels, the goal is to assign each pixel into one of these K groups. For example, for an image of an animal standing in a field, a 3 group segmentation may correspond to the animal, the sky and the ground. An example of such a segmentation can be seen in figure 1.

Segmentation models determine segments based on pixel color/values, where a segment contains pixel that all appear similarly. In this sense, segmentation is similar to pixel clustering. However, an image segmentation model also considers the position of pixels. Adjacent pixels are likely to belong to the same cluster.

In this assignment, you will implement and train an MRF that will assign each of the S pixels in an image into one of K discrete states. The MRF will encourages pixels to be assigned to segments with a similar color and many adjacent pixels. We will use the Berkeley Segmentation Dataset and Benchmark: `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`

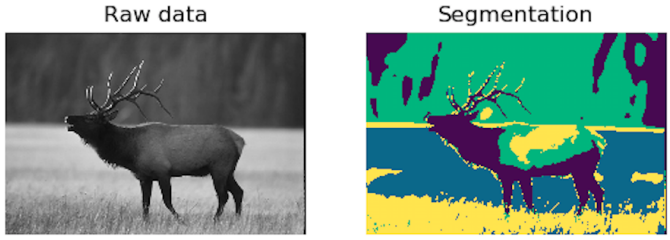


Figure 1: An example of an image segmentation on a grayscale image.

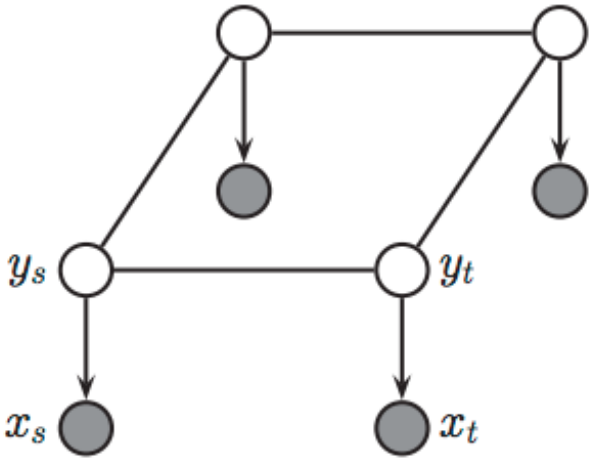


Figure 2: Illustration of Potts model (image credit to Kevin Murphy's book).

1.2 MRF specification

You will implement the Potts model, a type of MRF shown in Figure 2 with the goal of clustering pixels in an image. From this perspective, each image is a dataset containing many pixels that will be clustered. As the distribution of pixel intensities will vary from image to image, the clustering results will look different in different images. Therefore, unlike models where parameters are trained over many images, parameters in the Potts model will be trained separately for each image. In this model, each pixel s in the image corresponds to two random variables in the graphical model: y_s and x_s.

1. $y_s \in 1, 2, ..., K$ is a latent random variable for pixel s that has K possible states, where the value of y_s indicates the segment.
2. $x_s \in 0, ..., 255$ is the observed intensity of pixel s.

For notation, let $y = \{(y_s)\}_{s=1}^{S}$ and $x = \{(x_s)\}_{s=1}^{S}$.

Network structure: The Potts model connects each latent variable y_s to the latent variables corresponding to neighboring (adjacent) pixels. Assuming pixel s corresponds to the pixel in row i and column j of the image, the Potts model connects y_s to four other pixels:

1. The pixel in row $i + 1$ and column j.
2. The pixel in row $i - 1$ and column j.
3. The pixel in row i and column $j + 1$.
4. The pixel in row i and column $j - 1$.

A pixel on the boarder of the image will have fewer connections, e.g. a pixel in the corner will only have two adjacent pixels. Let $\mathcal{N}(s)$ correspond to the neighbors of s.

Data likelihood: The joint likelihood of the full Potts model is:

$$p(y, x \mid \theta, J) = p(y \mid J) \prod_{s=1}^{S} p(x_s \mid y_s, \theta) \tag{1}$$

where $p(y \mid J)$ is the prior distribution on the latent variables, y. This prior distribution is parameterized by J, a model hyper-parameter that controls the strength of the connection between two adjacent pixels. $p(x_s \mid y_s, \theta)$ is the emission probability for pixel s, i.e. the likelihood of a pixel, s, observing the value x_s given that it belongs state y_s. θ parameterizes emission probability distributions. Letting $s \sim t$ represent the set of pairs of pixels that share an edge in the MRF. We can expand equation 1 as follows:

$$p(y, x \mid \theta, J) = \left(\frac{1}{Z(J)} \prod_{s \sim t} \psi(y_s, y_t \mid J) \right) \prod_{s=1}^{S} p(x_s \mid y_s, \theta) \tag{2}$$

$$Z(J) = \sum_{y} \prod_{s \sim t} \psi(y_s, y_t \mid J) \tag{3}$$

where $\psi(y_s, y_t \mid J)$ is the edge potential, according to the MRF, between the segmentation for pixel s and pixel t, and $Z(J)$ is the partition function.

Edge potentials: You will use the following potential function to model the interaction between two neighboring pixels:

$$\psi(y_s, y_t \mid J) = \begin{cases} e^J & \text{if } y_s = y_t \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

Emission probabilities: You will use a Gaussian likelihood to model the emission probability $p(x_s|y_s, \theta)$. Specifically when $y_s == k$:

$$p(x_s|y_s = k, \theta) = \mathcal{N}(x_s|\mu_k, \sigma_k) \tag{5}$$

where μ_k and σ_k are the mean and standard deviation parameters, respectively, defining the univariate Gaussian distribution corresponding to state k. The parameter θ corresponds to $\{(\mu_k, \sigma_k)\}_{k=1}^{K}$.

1.3 MRF Parameter Learning

Learning the parameters of the Potts model corresponds to:

1. Computing the expected value of the posterior distribution on the latent variables: $p(y|x, \theta, J)$
2. Optimizing the parameters θ defining the emission probabilities $p(x_s|y_s, \theta)$.

Since this is likelihood maximization with latent variables we will use EM.

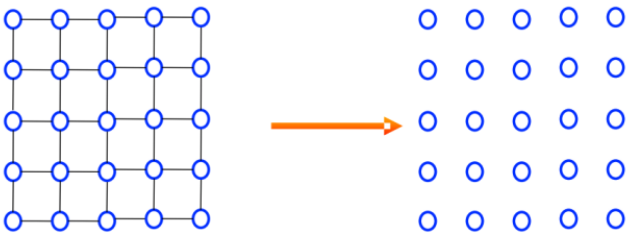


Figure 3: Illustration of Mean Field Variational Approximation (image credit to Kevin Murphy's book).

E-step: Variational Inference: In each iteration of EM, the E-step computes estimates of $p(y|x, \theta, J)$, where θ is fixed at its optimal value according to the most recent M-step. As $\{(x_s)\}_{s=1}^{S}$ is observed, $p(y|x, \theta, J) \propto p(y, x|\theta, J)$. Therefore, the E-step involves computing the distribution specified in equation 2. Unfortunately however, the partition function $Z(J)$ becomes intractable to compute when there are more than a few pixels. Notice that the graph is not tree-structured, so we cannot rely on the exact inference algorithms we learned in class.

Instead, we approximate the posterior $p(y|x, \theta, J)$ using Mean Field Variational Inference (See `https://www.cs.cmu.edu/~epxing/Class/10708-17/notes-17/10708-scribe-lecture13.pdf` for an overview of Variational Inference).

Mean Field Variational Inference approximates the complex posterior distribution $p(y|x, \theta, J)$ with a fully factorized distribution $q(y)$:

$$p(y|x, \theta, J) \approx q(y) = \prod_{s=1}^{S} q(y_s, p_s) \tag{6}$$

where $p_s \in \mathbb{R}^K$ parameterizes the variational distribution q_s to model the (categorical) posterior probability that y_s is equal to each of the K states. Specifically:

$$q(y_s = k, p_s) = p_s^{(k)} \tag{7}$$

$$\sum_{k=1}^{K} p_s^{(k)} = 1 \tag{8}$$

A schematic of the factorization of the MRF posterior according to Mean Field can be found in figure 3. This is a gross simplification of the true distribution, but the simplification makes inference much easier. We fit this simplified distribution to match, as closely as possible, the true distribution.

Variational Inference Pseudocode We provide the following pseudocode to optimize the Variational parameters (p_s) to make the Variational distributions (q_s) as similar as possible to the true posterior $p(y|x, \theta, J)$ in terms of KL-Divergence, a metric to measure the difference between two probability distributions.

1. Initialize the Variational parameters $\{(p_s) \in \mathbb{R}^K\}_{s=1}^{S}$ at random while ensuring $\sum_{k=1}^{K} p_s^{(k)} = 1$. Note: This step is already implemented in the Python skeleton (see function *initialize_variational_parameters()* in *models.py*) to ensure consistency between students.

2. For iteration $m = 1$ to M:
   (a) For pixel $s = 1$ to S:
      (i) For each state k, update $p_s^{(k)}$ while fixing variational approximations of all other pixels to their most recent updated value according to:

$$q(y_s = k, p_s) = \frac{\mathcal{N}(x_s \mid \mu_k, \sigma_k) \exp(\sum_{i \in \mathcal{N}(s)} J\, q(y_i = k, p_i))}{\sum_{k'=1}^{K} \mathcal{N}(x_s \mid \mu_{k'}, \sigma_{k'}) \exp(\sum_{i \in \mathcal{N}(s)} J\, q(y_i = k', p_i))} \quad (9)$$

   M is hyperparameter controlling the number of Variational inference iterations to run.

To ensure consistency between students: when looping through pixels please use an outer loop to iterate across rows of the image and an inner loop to iterate across columns of the image.

M-step: Maximum Likelihood Estimation: The M-step will update the parameters of the Potts model $\theta = \{(\mu_k, \sigma_k)\}_{k=1}^{K}$ given the most recent E-step's variational approximations to the posterior $(q(y_s, p_s))$.

The updates are as follows. For state $k = 1$ to K:

- $\widehat{\mu_k} = \frac{\sum_{s=1}^{S} q(y_s=k,p_s) x_s}{\sum_{s=1}^{S} q(y_s=k,p_s)}$
- $\widehat{\sigma_k^2} = \frac{\sum_{s=1}^{S} q(y_s=k,p_s)(x_s-\widehat{\mu_k})^2}{\sum_{s=1}^{S} q(y_s=k,p_s)}$

Expectation-Maximization Pseudocode: We now summarize the entire EM procedure.

1. Initialize $\theta^{(0)} = \{(\mu_k, \sigma_k)\}_{k=1}^{K}$ at random. Specifically, randomly assign $\mu_k \; \forall k$ to integers between 10 and 240 (inclusive) and set $\sigma_k = 10 \; \forall k$. This step is already implemented in the Python skeleton (see function *initialize_theta_parameters()* in *models.py*) to ensure consistency between students.
2. For iteration $n = 1$ to N:
   (a) E-Step: Run Variational Inference to generate $q(y_s, p_s)^{(n)} \; \forall s$ while keeping θ fixed at $\theta^{(n-1)}$
   (b) M-Step: Use Maximum Likelihood to learn $\theta^{(n)}$ while keeping $q(y_s, p_s)$ fixed at $q(y_s, p_s)^{(n)}$
3. Run E-step one more time to generate $q(y_s, p_s)^{(N+1)} \forall s$ while keeping θ fixed at $\theta^{(N)}$.
4. For pixel $s = 1$ to S:
   (a) Compute most likely state of pixel s according to $q(y_s, p_s)^{(N+1)}$. States should be encoded $\{0, .., K-1\}$. If there is a tie (ie. two states have equal probability), select the state corresponding to the lower number.

N is hyperparameter controlling the number of EM iterations to run.

1.4 Implementation Details

Your code must correctly implement the behavior described by the following command line options, which are passed in as arguments to the constructor of your implementation in models.py.

- `--edge-weight` Model hyper-parameter controlling the MRF prior distribution (J).
- `--num-states` Model hyper-parameter controlling the number of MRF latent states (K).
- `--n-em-iterations` Model hyper-parameter controlling the number of EM iterations (N).
- `--random-seed` Integer specifying random seed to be used to ensure consistency of results between students.
- `--n-vi-iterations` Model hyper-parameter controlling the number of iterations to use during Variational Inference (M).
- `--train-data` A string corresponding to the name and location of the image file.
- `--model-file` A string corresponding to where to store the model parameters
- `--predictions-file` A string corresponding to where to store model predictions. *main.py* will generate a tab-separated matrix to save here, where the matrix is the shape as the input image (from –train-data). Each row of the matrix corresponds to a row of the input image (in pixel space) and each column corresponds to a column of the input image. Each element of this matrix corresponds to the most likely state of the corresponding pixel according to the fitted MRF. States should be encoded $\{0, .., K-1\}$.
- `--visualize-predictions-file` A string corresponding to where to store a plot that jointly shows the raw image as well as image with pixels colored according to your segmentation.

1.5 Examples

We have provided four images for you to train your model and evaluate your clustering: *image_1.jpg*, *image_2.jpg*, *image_3.jpg*, *image_4.jpg*.

As an example of how the code should be run: the following trains an MRF on a training image image_1.jpg:

```
python3 main.py --train-data image_1.jpg --model-file train.model \
     --predictions-file train.predictions --algorithm mrf --edge-weight 1.2 --num-states 3 \
     --visualize-predictions-file segmentation_view.png
```

1.6 Evaluating segmentations

We provide a script called *compute_accuracy.py* which will compute the accuracy of your state predictions relative to correct segmentations that we provide. We provide two correct segmentations for you to compare your results to:

1. `image_1.true_predictions`: A predictions file for *image_1.jpg* generated using the following parameters:

```
--edge-weight 1.2 --num-states 4 --random-seed 1 \
   --n-em-iterations 3 --n-vi-iterations 3
```

2. `image_2.true_predictions`: A predictions file for *image_2.jpg* generated using the following parameters:

```
--edge-weight 1.2 --num-states 4 --random-seed 1 \
   --n-em-iterations 3 --n-vi-iterations 3
```

Note: your state predictions will only be comparible with the correct segmentations if you train your model using the above parameter settings. This script can be run using the following command:

```
python3 compute_accuracy.py image_1.true_predictions image_1.predictions
```

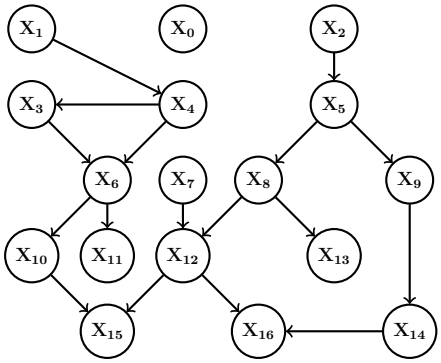
Where *image_1.true_predictions* is the provided correct segmentation and *image_1.predictions* is your state predictions file.

2 Analytical (50 points)

2.1 D-separation in Graphical Models (20 points)

2.1.1 Directed (8 points)

Consider the directed graphical model below. For each question, decide whether the sets **A** and **B** are d-separated given set **C**. Change `\Unchecked` to `\Checked` to provide your answer. Additionally, explain your answer in terms of the rules of d-separation.



(a) $\mathbf{A} = \{x_1\}$, $\mathbf{B} = \{x_9\}$, $\mathbf{C} = \{x_8, x_{16}\}$

☒Yes
☐No
x_{15} is unobserved, so the path is blocked

(b) $\mathbf{A} = \{x_{11}\}$, $\mathbf{B} = \{x_{13}\}$, $\mathbf{C} = \{x_1, x_{15}\}$

☐Yes
☒No
x_{15} is head to head so it unblocks a path from x_{11} to x_{13}

(c) $\mathbf{A} = \{x_4, x_0\}$, $\mathbf{B} = \{x_5, x_{13}\}$, $\mathbf{C} = \{x_{10}, x_{16}\}$

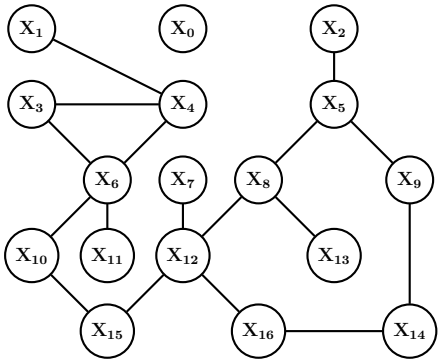
☒Yes
☐No
No paths from x_0 to any nodes, x_{10} is head to tail and blocks all paths

(d) $\mathbf{A} = \{x_3, x_4\}$, $\mathbf{B} = \{x_{13}, x_9\}$, $\mathbf{C} = \{x_7, x_{15}, x_{16}\}$

☐Yes
☒No
x_{15} and x_{16} are head to head and unblock a path from **A** to **B**

2.1.2 Undirected (8 points)

Suppose we take the same graph structure, but make the edges undirected. This change means that the directed version and undirected version will make different conditional-independence assertions. Again, for each of the following questions, decide whether the sets **A** and **B** are d-separated given set **C**. Additionally, explain your answer in terms of the rules of d-separation.



(a) $\mathbf{A} = \{x_1\}$, $\mathbf{B} = \{x_9\}$, $\mathbf{C} = \{x_8, x_{16}\}$

☒Yes
☐No
All paths must go through either x_8 or x_{16}

(b) $\mathbf{A} = \{x_{11}\}$, $\mathbf{B} = \{x_{13}\}$, $\mathbf{C} = \{x_1, x_{15}\}$

☒Yes
☐No
All paths must go through x_{15}

(c) $\mathbf{A} = \{x_4, x_0\}$, $\mathbf{B} = \{x_5, x_{13}\}$, $\mathbf{C} = \{x_{10}, x_{16}\}$

☒Yes
☐No
All paths must go through x_{10}

(d) $\mathbf{A} = \{x_3, x_4\}$, $\mathbf{B} = \{x_{13}, x_9\}$, $\mathbf{C} = \{x_7, x_{15}, x_{16}\}$

☒Yes
☐No
All paths must go through x_{15}

2.1.3 More questions (4 points)

Let $X = (X_1, ..., X_{16})^T$ be a random vector with distribution given by the graphical model. Consider variable X_2.

(a) What is the minimal subset of the variables, $A \subset \mathcal{X} - \{X_2\}$, such that X_2 is independent of the rest of the variables $(\mathcal{X} - (A \cup \{X_2\}))$ given A? Justify your answer.

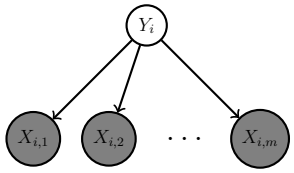
$A = \{X_5\}$
For a Bayesian network, the Markov blanket for a node consists of the co-parents, parents, and children of the node. X_2 has no parents, and no co-parents for its direct child X_5. It is either disconnected from or head-to-tail with any node in A, with the exception of X_5.

(b) Answer the same question for the undirected network.

$A = \{X_5\}$
For a Markov random field, the Markov blanket simply consists of all the neighbors, and X_5 is the only neighbor of X_2.

2.2 EM and Naive Bayes (30 points)

Background The Naive Bayes graphical model assumes that all features in a m-dimensional $X \in \mathcal{X}^m$ are conditionally independent given the label Y. Let $\mathcal{Y}$ be the domain of the random variable Y. A Naive Bayes classifier can be represented as a graphical model,



When X and Y are fully observed, the Naive Bayes model's likelihood function is

$$p(\{\langle\langle x_{i,1}, \ldots, x_{i,m}\rangle, y_i\rangle\}_{i=1}^n) = \prod_{i=1}^n p(y_i) \prod_{j=1}^m p(x_{i,j} \mid y_i) \tag{10}$$

In terms of parameters, we write the likelihood as

$$= \prod_{i=1}^n \theta_{y_i} \prod_{j=1}^m \theta_{x_{i,j}|y_i}$$

2.2.1 Parameters (4 points)

(a) If our model has m features with $n_X = |\mathcal{X}|$, and $n_Y = |\mathcal{Y}|$ possible values for the label, how many parameters does the Naive Bayes model have? Give a big-$\mathcal{O}$ expression and briefly justify your answer.

$\mathcal{O}(mn_X n_Y)$

$\mathcal{O}(mn_X n_Y)$ for $\theta_{x_{i,j}|y_i}$ since the features are conditionally independent given Y

$\mathcal{O}(n_Y)$ for θ_{y_i}, one each $y_i \in \mathcal{Y}$

(b) Not all settings to the parameter vector will give a valid interpretation as a (prior or conditional) probability distribution. Describe the necessary constraints on our parameter vector θ. Be precise.

$$\theta_{y_i \notin \mathcal{Y}} = 0,\ \theta_{y_i \in \mathcal{Y}} \in [0,1]\ \forall i$$

$$\sum_{y_i \in \mathcal{Y}} \theta_{y_i} = 1,\ \forall i$$

$$\theta_{x_{i,j} \notin \mathcal{X}|y_i} = 0,\ \theta_{x_{i,j} \in \mathcal{X}|y_i} \in [0,1]\ \forall i$$

$$\sum_{j=1}^{M} \sum_{x_{i,j} \in \mathcal{X}} \theta_{x_{i,j}|y_i} = \theta_{y_i}\ \forall i$$

2.2.2 Partially labeled training (26 points)

Now we consider a *semi-supervised* learning scheme with n observations, in which X is observed for all training examples, but Y is only *partially* observed: rather than observing the event $(Y = y_i)$, we observe the event $(Y_i \in \mathcal{Y}_i)$ for some (nonempty) example-specific subset $\mathcal{Y}_i \subseteq \mathcal{Y}$. Thus, our training data is a set of pairs $\mathcal{D} = \{\langle\langle x_{i,1}, \ldots, x_{i,m}\rangle, \mathcal{Y}_i\rangle\}_{i=1}^{n}$.

(a) (6 points) Because some of labels Y_i are missing, the likelihood function in equation 10 does not apply. Write down the likelihood function for our partially labeled dataset. You may use the $p()$ notation instead of θ, if you prefer. [Hint: Check for yourself that when $\mathcal{Y}_i$ always has one element in it that you recover the fully observed case.]

$$\prod_{i=1}^{n} \sum_{y_i \in \mathcal{Y}_i} p(y_i) \prod_{j=1}^{m} p(x_{i,j}|y_i) = \prod_{i=1}^{n} \sum_{y_i \in \mathcal{Y}_i} \theta_{y_i} \prod_{j=1}^{m} \theta_{x_{i,j}|y_i}$$

(b) (7 points) EM algorithm (E-step): The E-step for our model computes an approximate posterior distribution $q(Y_i = y \mid x_i)$ over the missing labels using the current parameter estimate, $\widehat{\theta}$. [Hint: the posterior should put zero probability on the event that $(Y_i \notin \mathcal{Y}_i)$ and it should sum to one over $y \in \mathcal{Y}$.]

For $i \in \{1, \ldots, n\}$ and $y \in \mathcal{Y}$, the posterior is the following distribution

$$q(Y_i = y \mid x_i) = \frac{\widehat{\theta}_y \prod_{j=1}^{m} \widehat{\theta}_{x_{i,j}|y} \text{ bool}(y \in \mathcal{Y}_i)}{\sum_{y' \in \mathcal{Y}_i} \widehat{\theta}_{y'} \prod_{j=1}^{m} \widehat{\theta}_{x_{i,j}|y'}}$$

The bool term just ensures that any y outside of the labeled domain is set to 0

(c) (7 points) EM algorithm (M-step): Write down the M-step objective function using the approximate posterior, q, that you defined in the previous question. There is no need to solve the optimization problem, just write it down clearly.

$$\operatorname*{argmax}_{\theta} \prod_{i=1}^{n} \sum_{y \in Y_i} q(Y_i = y \mid X_i, \mathcal{Y}_i, \widehat{\theta}) \log p(X_i, Y_i = y, \mathcal{Y}_i \mid \theta)$$

or, expanding each of these terms:

$$\operatorname*{argmax}_{\theta} \prod_{i=1}^{n} \sum_{y \in \mathcal{Y}_i} \frac{\widehat{\theta}_y \prod_{j=1}^{m} \widehat{\theta}_{x_{i,j}|y}}{\sum_{y' \in \mathcal{Y}_i} \widehat{\theta}_{y'} \prod_{j=1}^{m} \widehat{\theta}_{x_{i,j}|y'}} \log \left(\sum_{y_i \in \mathcal{Y}_i} \theta_{y_i} \prod_{j=1}^{m} \theta_{x_{i,j}|y_i} \right)$$

Note that I dropped the boolean term from q just because we're able to limit the summation to $y \in \mathcal{Y}_i$

(d) (2 points) Could we optimize the data likelihood (your solution to part (a)) with a gradient-based optimization algorithm instead of EM?

Yes, we could optimize the data likelihood with a gradient-based optimization algorithm since our likelihood function is differentiable. However, this will only give us an approximate solution rather than the optimal solution since our labels are sets $\mathcal{Y}_i$ rather than true labels Y_i

(e) (4 points) What if we wanted to support real-valued features where $\mathcal{X} = \mathbb{R}$? What would be a reasonable way to parameterize the label-conditioned feature distribution $p(x_j \mid y)$? How many parameters does the new model have?

We can assume that $p(x_j \mid y)$ follows a normal distribution (or any other distribution, but for parameter counting we'll use normal. Then the total number of parameters for the model will be $2mn_Y+n_Y$, following a similar logic used in 2.2.1a except here instead of n_X each x_j is parametrized by two variables μ and σ. (To be extra specific, since we do still have the constraint of having everything sum to 1, we can really cut this down to $2(m-1)(n_Y-1)+(n_Y-1)$.)

3 What to Submit

In this assignment you will submit two things.

1. **Submit your code (.py files) to cs475.org as a zip file**. **Your code must be uploaded as code.zip with your code in the root directory**. By 'in the root directory,' we mean that the zip should contain `*.py` at the root (`./*.py`) and not in any sort of substructure (for example `hw1/*.py`). One simple way to achieve this is to zip using the command line, where you include files directly (e.g., `*.py`) rather than specifying a folder (e.g., `hw1`):

   ```
   zip code.zip *.py
   ```

   A common mistake is to use a program that automatically places your code in a subfolder. It is your job to make sure you have zipped your code correctly.

   We will run your code using the exact command lines described earlier, so make sure it works ahead of time, and make sure that it doesn't crash when you run it on the test data. A common mistake is to change the command line flags. If you do this, your code will not run.

   Remember to submit all of the source code, including what we have provided to you. We will include `requirements.txt` and provide the data, but nothing else.

2. **Submit your writeup to gradescope.com**. **Your writeup must be compiled from latex and uploaded as a PDF**. The writeup should contain all of the answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and to use the provided latex template for your answers following the distributed template. You will submit this to the assignment called "Homework 4: EM and Graphical Models: Written".

You will need to create an account on gradescope.com and signup for this class. The course is `https://gradescope.com/courses/21552`. Use entry code `M6ZX2X`. See this video for instructions on how to upload a homework assignment: `https://www.youtube.com/watch?v=KMPoby5g_nE`.

4 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: `https://piazza.com/class/jkqbzabvyr15up`.