10-708 Probabilistic Graphical Models

Homework 2

Due Mar 22, 11:59 PM

Rules:

1. Homework is due on the due date at 11:59 PM. The homework should be submitted via Gradescope. Solution to each problem should start on a *new page* and marked appropriately on Gradescope. For policy on late submission, please see course website.
2. We recommend that you typeset your homework using appropriate software such as LaTeX. If you are writing, please make sure your homework is cleanly written up and legible. The TAs will not invest undue effort to decrypt bad handwriting.
3. **Code submission:** for programming questions, you must submit the complete source code of your implementation also via Gradescope. Remember to include a small README file and a script that would help us execute your code.
4. **Collaboration:** You are allowed to discuss the homework, but you should write up your own solution and code. Please indicate anyone you collaborated with in your submission.

---

1 KL-divergence [30 pts] (Yuanning)

1.1 Divergence measures and maximum likelihood estimation (3 + 4 + 4 + 5 = 16 pts)

In Lecture 8, we introduced *Iterative Proportional Fitting* (IPF) algorithm as a way to learn the parameters of a undirected graphical model, and we claim that IPF is also a coordinate ascent algorithm (see slide 41). In this question, we are going to prove that **IPF algorithm is essentially a coordinate ascent algorithm that maximizes the likelihood $p(x;\theta)$ over parameter θ**. Here we consider use KL-divergence as a tool to prove our main result. In Part (a-c) we prove a set of useful lemmas with respect to KL-divergence and MLE. Finally in Part (d), we prove the main result using lemmas from (a-c).

(a) Let $p(x), q(x)$ be two positive probability densities on $\mathbb{R}$. Prove Gibbs inequality

$$D_{KL}\big(p(x)\|q(x)\big) = \int p(x)\log\Big(\frac{p(x)}{q(x)}\Big)dx \geq 0$$

and the equality holds if $p(x) = q(x)$.

(b) Assume we have N *i.i.d.* samples $x^{(1)}, ..., x^{(N)} \in \mathbb{R}^d$ from distribution $p(x;\theta^*)$, where θ^* is the true parameter that we want to estimate. The maximum likelihood estimator is

$$\widehat{\theta}_{MLE} \equiv \operatorname*{argmax}_{\theta} \frac{1}{N}\sum_{i=1}^{N}\log p(x^{(i)};\theta)$$

Show that, for large N, maximum likelihood estimation also (asymptotically) minimizes the KL-divergence $D_{KL}(p(x;\theta^*)\|p(x;\theta))$, i.e.

$$\widehat{\theta}_{MLE} = \operatorname*{argmin}_{\theta} D_{KL}(p(x;\theta^*)\|p(x;\theta)), \quad N \to \infty$$

Hint: consider the law of large numbers.

(c) Let p, q be two positive distributions, and x_A, x_B be two (non-overlapping) sets of variables. Prove the following equation

$$D_{KL}(p(x_A, x_B)\|q(x_A, x_B)) = D_{KL}(p(x_A)\|q(x_A)) + \sum_{x_A} p(x_A) D_{KL}(p(x_B|x_A)\|q(x_B|x_A))$$

(d) Consider a decomposable undirected graphical model $p(x) = \frac{1}{Z}\prod_i \phi_{C_i}(x_{C_i})$, where $C_1, ..., C_k$ are k clusters of the graph, and $U = \cup_{i=1}^{k} C_i$ is the full set of all nodes. Given a set of N *i.i.d.* samples $x^{(1)}, ..., x^{(N)}$, the IPF algorithm estimates the potential function $\phi_C(x_C)$ for each cluster using the following iteration

$$\phi_C^{(t+1)}(x_C) = \phi_C^{(t)}(x_C)\frac{\epsilon(x_C)}{p^{(t)}(x_C)}$$

where $\epsilon(x_C) = \frac{1}{N}\sum_{i=1}^{n}\mathbb{I}[x_C = x_C^{(i)}]$ is the empirical marginal distribution of cluster C, and $p^{(t)}(x_C) = \sum_{x_{U\setminus C}} p^{(t)}(x) = \sum_{x_{U\setminus C}} \frac{1}{Z^{(t)}}\prod_{i=1}^{k}\phi_{C_i}^{(t)}(x_{C_i})$ is the estimated marginal distribution of cluster C at the current step t.

Prove that IPF is coordinate ascent in the log likelihood of the data.

Note: coordinate ascent is an iterative way to find the maximum of a multivariate function $f(x)$ with $x \in \mathbb{R}^d$. Specifically, at each iteration, it minimizes $f(x)$ over a coordinate x_i or coordinate block x_C ($C = \{i_1, ..., i_k\}$ is a subset of all the coordinates) while fixing all the other coordinates x_{-C}, i.e.

$$x_C^{(t+1)} = \operatorname*{argmax}_{x_C} f(x_C, x_{-C}^{(t)})$$

1.2 Divergence measures and Fisher information (4 + 2 + 4 + 4 = 14 pts)

Fisher information is a way of measuring the amount of information that an observed variable x has about parameter θ of a distribution $p(x;\theta)$. Divergence measures, such as KL-divergence, have deep connections to Fisher information from the perspective of information geometry. Here we are going to study some important properties that tie together Fisher information, KL-divergence, and parameter estimation.

(a) For a multivariate distribution $p(x;\theta)$ on $\mathbb{R}^d$ with $\theta \in \mathbb{R}^n$, and assume that regularity conditions apply to $p(x;\theta)$, the *Fisher information matrix* $\mathcal{I}(\theta) \in \mathbb{R}^{n\times n}$ is defined as

$$\mathcal{I}(\theta) \equiv \mathbb{E}_{p(x;\theta)}\big[\nabla_\theta \log p(x;\theta) \nabla \log p(x;\theta)^T\big]$$

Show that

$$\mathcal{I}(\theta) = -\mathbb{E}_{p(x;\theta)}[\nabla^2_\theta \log p(x;\theta)]$$

(b) Find the *Fisher information matrix* $\mathcal{I}(\theta)$ for exponential family $p(x;\theta) = h(x)\exp[\theta^T T(x) - A(\theta)]$.

(c) Let $p(x;\theta)$ be a distribution on $\mathbb{R}$, with $\theta \in \mathbb{R}$, and assume that regularity conditions apply to $p(x;\theta)$. We are often interested in how sensitive the pdf is with respect to parameter θ. Consider distribution $p(x;\theta+\delta)$, where a small perturbation $\delta \in \mathbb{R}$ is added to parameter θ. Show that, for small δ

$$D_{KL}\big(p(x;\theta)\|p(x;\theta+\delta)\big) = -\frac{\delta^2}{2}\mathbb{E}_{p(x;\theta)}\big[\frac{\partial^2}{\partial\theta^2}\log p(x;\theta)\big] + o(\delta^2)$$

Note: here we use little o notation to represent remainder $r = o(\delta^2)$, which means $r/\delta^2 \to 0$, as $\delta \to 0$.

(d) Now consider the exponential family distribution $p(x;\theta) = h(x)\exp[\theta^T T(x) - A(\theta)]$, and small perturbed $p(x;\theta+\delta)$ with $\theta, \delta \in \mathbb{R}^n$. Show that

$$D_{KL}\big(p(x;\theta)||p(x;\theta+\delta)\big) = \frac{1}{2}\delta^T \nabla^2 A(\theta)\delta + o(\|\delta\|^2)$$

2 CRF Learning for OCR [30 pts] (Yifeng)

Note: Please submit all your answer (including derivations, calculated values, figures) to this problem in the pdf file, the executable code and provided dataset in the *.zip file.

2.1 Introduction

In this problem, you will implement your own code to learn the parameters of a conditional random field (CRF) for optical character recognition (OCR).

In the CRF model, there are two kinds of variables: the hidden variables that we want to model and those that are always observed. In the case of OCR, we want to model the character assignments (such as 'a' or 'c'), and we always observe the character images, which are arrays of pixel values. Typically, the unobserved variables are denoted by Y and the observed variables are denoted by X. The CRF seeks to model $P(Y|X)$, the conditional distribution over character assignments given the observed images. We will be working with the OCR model that includes only the singleton and pairwise factors (as we will explain in detail below). The structure of the model is shown below:

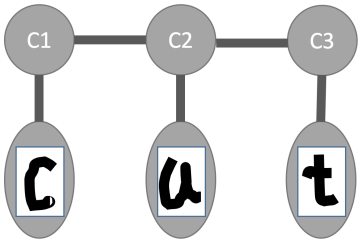

Figure 1: CRF with singleton and pairwise factors.

As you can see, our CRF for OCR is a specific case of the general CRF mentioned in class. Specifically, you will build the model around log-linear features. A feature is just a function $f_i(D_i)$: $Val(D_i) \rightarrow \mathbb{R}$, where D_i is the set of variables in the scope of the i-th feature. Here, all features are binary indicator features (taking on a value of either 0 or 1). Each feature has an associated weight θ_i. Given the features $\{f_i\}_{i=1}^k$ and the weights $\{\theta_i\}_{i=1}^k$, the distribution is defined as:

$$P(Y|X;\theta) = \frac{1}{Z_X(\theta)} \exp\{\sum_{i=1}^{k} \theta_i f_i(D_i)\}. \tag{1}$$

The term $Z_X(\theta)$ is the partition function:

$$Z_X(\theta) = \sum_{Y} \exp\{\sum_{i=1}^{k} \theta_i f_i(D_i)\}. \tag{2}$$

In our CRF, we have three types of features:

- $f_{i,c}^C(Y_i)$, which operates on single characters / hidden states (an indicator for $Y = c$).
- $f_{i,j,c,d}^I(Y_i, X_{ij})$,which operates on a single character / hidden state and an image pixel associated with that state (an indicator for $Y_i = c, X_{ij} = d$). These are collectively used to encode the individual probability that $Y_i = c$ given X_i.
- $f_{i,c,d}^P(Y_i, Y_{i+1})$which operates on a pair of adjacent characters / hidden states (an indicator for $Y_i = c, Y_{i+1} = d$).

2.2 Gradient of Negative Log-likelihood [5pts]

Our goal is to maximize the log-likelihood of the parameters given the data. Thus the cost / loss we minimize is simply the negative log-likelihood, together with a L_2-regularization penalty on the parameter values to prevent overfitting. The function we seek to minimize is:

$$\text{nll}(X, Y, \theta) = \log(Z_X(\theta)) - \sum_{i=1}^{k} \theta_i f_i(Y, X) + \frac{\lambda}{2} \sum_{i=1}^{k} \theta_i^2. \tag{3}$$

Question: Please prove that partial derivatives for this function have an elegant form:

$$\frac{\partial}{\partial \theta_i} \text{nll}(X, Y, \theta) = E_\theta[f_i] - E_D[f_i] + \lambda \theta_i, \tag{4}$$

where

$$E_\theta[f_i] = \sum_{Y'} P(Y'|X) f_i(Y', X), \tag{5}$$

$$E_D[f_i] = f_i(Y, X). \tag{6}$$

We drop the θ from $P(Y'|X)$ for convenience.

2.3 Shared Parameters [2pts]

There is one more detail to take care of: shared parameters across multiple features. Parameter sharing is a form of templating used to reduce the total number of parameters we need to learn. Let $\{f^{(i)}\}$ be the set of features that share parameter θ_i.

Question: Give a brief explanation that we can expend the equations (3, 4) above as:

$$\text{nll}(X, Y, \theta) = \log(Z_X(\theta)) - \sum_{i=1}^{k} \theta_i \Big(\sum_{f_j \in \{f^{(i)}\}} f_j(Y, X) \Big) + \frac{\lambda}{2} \sum_{i=1}^{k} \theta_i^2. \tag{7}$$

$$\frac{\partial}{\partial \theta_i} \text{nll}(X, Y, \theta) = \sum_{f_j \in \{f^{(i)}\}} E_\theta[f_j] - \sum_{f_j \in \{f^{(i)}\}} E_D[f_i] + \lambda \theta_i. \tag{8}$$

Note that in this case, θ_i is not necessarily related to f_i any longer. Instead, θ_i is associated with a set of features $\{f^{(i)}\}$. k is the total number of parameters of θ_i, and not necessarily the total number of features f_j. The parameters that we use in the CRF are:

- θ^C_c, shared by $\{f^C_{i,c}(Y_i)\}_i$.
- $\theta^I_{c,d}$, shared by $\{f^I_{i,j,c,d}(Y_i, X_{ij})\}_{i,j}$.
- $\theta^P_{c,d}$, shared by $\{f^P_{i,c,d}(Y_i, Y_{i+1})\}_i$.

Essentially, this parameter tying scheme ties parameters across different locations together; that is, a character at the start of the word shares the same parameters as a character at the end of the word.

2.4 InstanceNegLogLikelihood Implementation [18pts]

Given this discussion, we can now state your mission: for a given data instance (X, Y) and a parameter setting θ, you must compute the cost function (regularized negative log-likelihood) and the gradient of parameters with respect to that cost. Doing so involves six terms (ignoring the issue of shared parameters in these):

- The log partition function: $\log(Z_X(\theta))$
- The weighted feature counts: $\theta_i f_i(Y, X)$
- The regularization cost: $\frac{\lambda}{2}\sum_{i=1}^{k}\theta_i^2$
- The model expected feature counts: $\sum_{Y'} P(Y'|X) f_i(Y', X)$
- The data feature counts $f_i(Y, X)$
- The regularization gradient term: $\lambda\theta_i$.

In this problem, you are provided with a single instance and related parameters in `q2instance.mat` to help you implement your `InstanceNegLogLikelihood` function, which should return the negative log-likelihood and gradient of the parameters. First, we should clarify what exactly we mean by a data "instance." The input to `InstanceNegLogLikelihood` function includes X and Y which together form a single instance (X, Y). This instance corresponds to one word, that is, a sequence of characters.

Thus, the variable Y is a vector of character values. For example, $Y = (3, 1, 20)$ means that this instance contains the word "cat." The variable X is a matrix where the i-th row contains the pixel values for the ith character in the example. Each row has 32 values, since the images are 8x4 pixels. The feature sharing described above comes into play due to the fact that our data instances are not single images (as they were in the first part of the assignment) but instead sequences of images that we want to consider together.

(a) Number of Parameters [6pts]

Question: Since we have a sequence of Y with length 3, where each position has 26 hidden states; and each image has 32 binary positions. Considering the shared parameters across multiple features, show that θ has length of 2366, i.e., clarify how many specific parameters in three different cases: $\theta_c^C, \theta_{c,d}^I, \theta_{c,d}^P$.

(b) Implementation of InstanceNegLogLikelihood [12pts]

Please implement your `InstanceNegLogLikelihood` function:
`[nll, grad] = InstanceNegLogLikelihood(X, y, theta, params)`.
Question: You are required to provide the following values in your pdf report:

- Negativelog-likelihood: `sampleNLL`$= \log(Z_X(\theta)) - \sum_{i=1}^{k}\theta_i(\sum_{f_j\in\{f^{(i)}\}} f_j(Y, x)) + \frac{\lambda}{2}\sum_{i=1}^{k}\theta_i^2$,
- L_2 norm of parameter gradient: $||$`sampleGrad`$||_2 = ||\sum_{f_j\in\{f^{(i)}\}} E_\theta[f_j] - \sum_{f_j\in\{f^{(i)}\}} E_D[f_i] + \lambda\theta_i||_2$,

where `[sampleNLL, sampleGrad] = InstanceNegLogLikelihood(sampleX, sampleY, sampleModelParams)`, and `sampleX`, `sampleY`, `sampleTheta`, `sampleModelParams` from `q2instance.mat`: Feel free to use any programming language you want and any resources from the Internet as soon as you acknowledge them properly.

2.5 Stochastic Gradient Descent [5pts]

In this question, we want to train the CRF using stochastic gradient descent with vanishing step size.

In `q2dataset.mat`, we provide `trainData` and `testData`. Each is a struct array with X and y fields. In both, each (X, y) pair is the input the `InstanceNegLogLikelihood`: X gives the observations and y gives the labels. You can directly use `sampleModelParams` from `q2instance.mat` as hyperparameters for training, and use `sampleTheta` from `q2instance.mat` for initialization of parameters.

Question: Train the model using stochastic gradient descent, plot the test loss versus training steps evaluated on the first 10 samples in the `testData`. We only require you to train the model for one epoch (i.e., 220 steps in total), since the training is time consuming. We recommend using the learning rate of $\alpha_k = \frac{1}{1+0.05k}$, where k is the number of steps.

3 Hierarchical Mixture of Experts [40 pts] (Xiongtao)

In this problem, we are going to work with a two-level hierarchical mixture of experts network (HME). Let $\mathbf{x} \in \mathbb{R}^r$ is the input, $\mathbf{y} \in \mathbb{R}^s$ is the output. If $\mathbf{y}$ is defined in a continuous space, it is a regression problem, and if $\mathbf{y}$ is defined in categories, it is a classification problem. The dataset consists of T paired observations $\mathcal{X} = \{(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})\}$, where $t = 1, ..., T$. Here we are going to use HME to work on regression problems.

The basic structure of an HME is shown in Figure 2. An HME is a tree in which non-terminals are gating networks and leaf nodes are expert networks. Both gating networks and expert networks use $\mathbf{x}$ as input. Usually, expert networks and gating networks are modeled as generalized linear model (GLM). Expert networks consist of some simple models, i.e. linear regression (for regression), Bernoulli probability model (for classification) and so on. Each of them produces a variable $\boldsymbol{\mu}_{ij}$ for each input vector. Gating networks consist of distributions that generate a partition of unity at each point in the input space, for example, they could be softmax operations on $\mathbf{x}$. Intuitively, expert networks define the relationship between observation and response, while gating networks act as prior/weights for expert networks, determining which expert networks are going to make a decision about $\mathbf{x}$. Because gating networks also depend on data, there would various combinations of expert networks used for decisions for different $(\mathbf{x}, \mathbf{y})$. Therefore, HME can model complex data by the combination of some simple models.

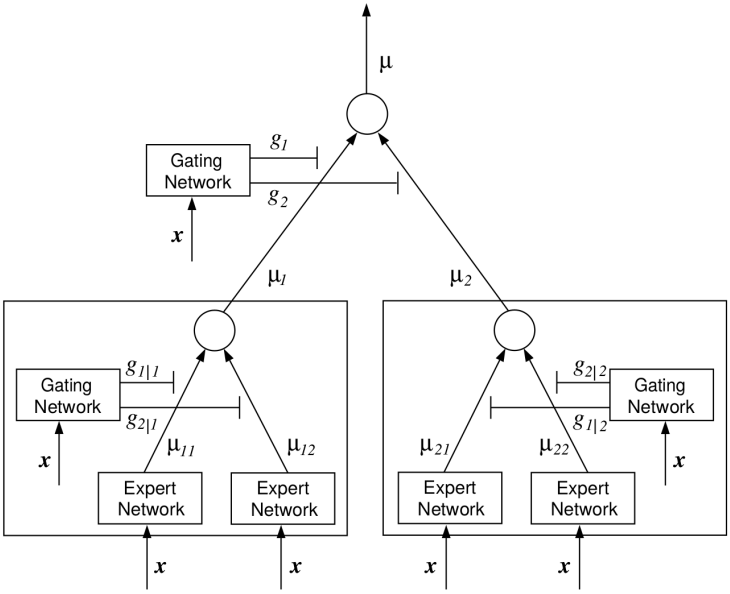

Figure 2: A two-level hierarchical mixture of experts.

In the HME, there are $MN > 0$ expert networks in the bottom layer, we use $ENet_{ij}$ to represent expert networks with index i, j, $i = 1 : M$, $j = 1 : N$. All $ENet_{ij}$, $j = 1 : N$ are children of $i - th$ node in the middle layer of the tree. Here, the expert networks are defined as linear regression models, with

$$\boldsymbol{\mu}_{ij} \sim \mathcal{N}(\mathbf{U}_{ij}\mathbf{x}, \sigma_i^2\mathbf{I})$$

where σ_i is shared among $i - th$ block of expert networks $ENet_{ij} \quad j = 1, ..., N$, $\mathbf{U}_{ij}$ is the weight matrix, and $\mathbf{I}$ is the identity matrix.

For the gating network in the top level, it is a "softmax" function of $\mathbf{v}_i\mathbf{x}$ as

$$g_i = \frac{e^{\mathbf{v}_i^T\mathbf{x}}}{\sum_{i'} e^{\mathbf{v}_{i'}^T\mathbf{x}}}$$

where $\mathbf{v}_i$ is a weight vector, $i = 1 : M$. Clearly, g_i is a GLM for $\mathbf{x}$.

For gating network in the lower level, we also define them as a GLM of $\mathbf{x}$, as

$$g_{j|i}(\mathbf{x}) = \frac{e^{\mathbf{v}_{ij}^T \mathbf{x}}}{\sum_k e^{\mathbf{v}_{ik}^T \mathbf{x}}}$$

where $j = 1 : N$.

As we mentioned before, gating networks could be viewed as weight of each node. Therefore, the output vector at each non-terminal node of the tree is the weighted output of the experts below that node, i.e. for $i - th$ node in the middle layer, the output is

$$\boldsymbol{\mu}_i = \sum_j g_{j|i} \boldsymbol{\mu}_{ij}$$

and the output at the top level is:

$$\boldsymbol{\mu} = \sum_i g_i \boldsymbol{\mu}_i$$

Finally, we define the response variable $\mathbf{y}$ as a (conditional) linear regression on the output of each expert network $\boldsymbol{\mu}_{ij}$ as

$$P(\mathbf{y}|\boldsymbol{\mu}_{ij}, \sigma) = \mathcal{N}(\boldsymbol{\mu}_{ij}, \sigma^2 I)$$

Where σ is the standard deviation, which is shared for the whole network.

From another point of view, we treat the gating networks as prior for $\mathbf{y}$ and the expert networks as conditional distribution with data, then we can get the distribution as

$$P(\mathbf{y}, \boldsymbol{\mu}|\mathbf{x}, \boldsymbol{\theta}) = \sum_i g_i(\mathbf{x}, \mathbf{v}_i) \sum_j g_{j|i}(\mathbf{x}, \mathbf{v}_{ij}) P(\mathbf{y}|\boldsymbol{\mu}_{ij}, \sigma) P(\boldsymbol{\mu}_{ij}|\mathbf{x}, \sigma_i)$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}_{11}, ..., \boldsymbol{\mu}_{MN}\}$, and $\boldsymbol{\theta}$ represents all parameters in the network, i.e. for both expert networks and gating networks.

We can also define a posterior probability associated with each node, that reflects the extent to which the node is activated. For the top level nodes, using the Bayes Rule, we define the posterior probabilities as

$$h_i = \frac{g_i \sum_j g_{j|i} P_{ij}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_E)}{\sum_i g_i \sum_j g_{j|i} P_{ij}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_E)}$$

where

$$P_{ij}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_E) := \sum_{\boldsymbol{\mu}_{ij}} P(\mathbf{y}|\mathbf{x}, \boldsymbol{\mu}_{ij}, \sigma) P(\boldsymbol{\mu}_{ij}|\mathbf{x}, \sigma_i)$$

which is the factorized distribution associated with each expert network, and $\boldsymbol{\theta}_E$ represents all parameters associated with expert networks, i.e. $\boldsymbol{\theta}_E = \{\sigma_i,\ i = 1 : M\} \cup \{\sigma\} \cup \{\mathbf{U}_{ij},\quad i = 1 : M,\quad j = 1 : N\}$.

3.1. Warm up [2 + 1 + 1 = 4 pts]

1. a problem in the model is that there are latent variables $\boldsymbol{\mu}$ in the full distribution, which complicate the model. Therefore, we would like to marginalize out $\boldsymbol{\mu}$ first. Please derive $p(\mathbf{y}|\mathbf{x}, \theta)$. You can keep g_i and $g_{j|i}$ in the distribution, and just work with other parts of the distribution. In the later part of the question, we will just use the marginal distribution, therefore, please be careful and make sure your derivation is correct.

   Hint: Linear Gaussian system would be helpful here.

2. Write out the posterior distributions at the nodes in the bottom level of the network, that is, $h_{j|i}$.

3. HME is considered as a nonlinear model for $\mathbf{x}$ and $\mathbf{y}$, reasoning why this network is a nonlinear model.

3.2. EM algorithm for HME [19 pts]

To derive the EM algorithm, we first define binary latent variables z_i and $z_{j|i}$ that associate with the nodes in the network. $z_{j|i} = 1$ means expert network $ENet_{ij}$ is activated, $z_i = 1$ means node i in top layer is activated. For the bottom layer, we only allow one expert network for the children of $i-th$ node in the top layer being activated, that is $\sum_j z_{j|i} = 1$, and we also just allow one node in the top level activated, that is, $\sum_i z_i = 1$. We define $z_{ij} = z_i z_{j|i}$, which could be interpreted as the activation of the specific expert network.

With z_i and $z_{j|i}$, we could rewrite the distribution for a single data point as

$$P(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\theta}) = \prod_i \prod_j (g_i g_{j|i} P_{ij}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}))^{z_{ij}}$$

where $\mathbf{z} = \{z_{ij}\},\ i = 1:M,\ j = 1:N$.

Note, z_{ij} is associated with each data point, then the complete-data distribution among T data points is

$$P(\mathbf{Y}, \mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}) = \prod_t \prod_i \prod_j (g_i g_{j|i} P_{ij}(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \boldsymbol{\theta}))^{z_{ij}^{(t)}}$$

where $\mathbf{Y} = \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(T)}\}$, and $\mathbf{Z} = \{z_{ij}^{(t)}\},\ i = 1:M,\ j = 1:N,\ t = 1:T$.

Then, the EM algorithm could be applied based on the distribution above. Below we provide the pseudo-code for the algorithm.

Input : $\mathcal{X}$, M, N, MaxIter, and other optional parameters
Output: Parameter $\boldsymbol{\theta}$, Log-likelihood in the update, Fitted output $\hat{\mathbf{Y}}$, $\mathbb{E}[z_{ij}^{(t)}|\mathcal{X}]$ in the last iteration

Initialize parameters $\boldsymbol{\theta}$;

`// Compute Log-likelihood`
Compute and record log-likelihood with initial parameters

for $k \leftarrow 1$ **to** *MaxIter* **do**

`// Stage 1: E-Step`
Update $\mathbb{E}[z_i^{(t)}|\mathcal{X}]$, $\mathbb{E}[z_{j|i}^{(t)}|\mathcal{X}]$ and $\mathbb{E}[z_{ij}^{(t)}|\mathcal{X}]$

`// Stage 2: M-Step`
`// Update` $\mathbf{U}_{ij}$
For each expert (i, j), update $\mathbf{U}_{ij}$ by directly solving MLE on expected log-likelihood

`// Update` σ_i
For each i, update σ_i by directly solving MLE on expected log-likelihood

`// Update` σ
Update σ by directly solving MLE

`// Update` $\mathbf{v}_i$ `in top-level gating network`
For each top-level gating network, solve an iteratively reweighted least squares (IRLS) problem to update $\mathbf{v}_i$

`// Update` $\mathbf{v}_{ij}$ `in lower-level gating network`
For each lower-level gating network, solve an IRLS problem to update $\mathbf{v}_{ij}$

`// Stage 3: Report Log-likelihood`
Compute and record log-likelihood

end

`// Compute the prediction of y`
Compute the fitted output $\hat{\mathbf{Y}}$ for all observations $\mathbf{X}$

Algorithm 1: Pseudo-code of EM algorithm for HME model

a. E-step [3 + 1 = 4 pts]

1. Derive E-step, that is, derive $\mathbb{E}[z_i^{(t)}|\mathcal{X}]$, $\mathbb{E}[z_{j|i}^{(t)}|\mathcal{X}]$ and $\mathbb{E}[z_{ij}^{(t)}|\mathcal{X}]$, you don't need to spread out g_i, $g_{j|i}$ and P_{ij}.

2. What is the relationship between $\mathbb{E}[z_i^{(t)}|\mathcal{X}]$, $\mathbb{E}[z_{j|i}^{(t)}|\mathcal{X}]$, $\mathbb{E}[z_{ij}^{(t)}|\mathcal{X}]$, and posterior distributions of nodes, as derived in Part 3.1.3?

b. M-step [3 + 6 + 4 = 13 pts]

1. write down the problems you need to solve to update parameters (the parameters you need to optimize), you don't need to spread out g_i, $g_{j|i}$ and P_{ij}, but please be as specific as possible, that is, throw out constant terms with respect to each parameter as much as possible, for example, parameter $\mathbf{U}_{ij}$ is only related to $\mathbb{E}[z_{ij}^{(t)}|\mathcal{X}]$ and $P_{ij}(\mathbf{y}^{(t)})$, $t = 1 : T$.

2. Derive the updating rules for parameter in expert networks based on the problem you write in last question (3.2.b.1). As indicated in the algorithm, we can directly solve the MLE on expected log-likelihood to get the update rules. More specifically, derive updating rules for $\mathbf{U}_{ij}$, σ_i and σ, where $i = 1 : M$, $j = 1 : N$. The result should be as specific as possible, that is, they could be directly used in the implementation.

   Actually, σ is not going to be updated, that is, it will be constant, argue why?

   Hint: think about the gradient after updating σ_i, i=1:M.

3. We need to update $\mathbf{v}_i$ and $\mathbf{v}_{ij}$ using the IRLS algorithm. Based on the problems in previous part, derive the IRLS updating rules for $\mathbf{v}_i$ and $\mathbf{v}_{ij}$ for $i = 1 : M$, $j = 1 : N$. The result should be as specific as possible, that is, they could be directly used in the implementation.

c. Log-likelihood and prediction [2 pts]

1. To monitor the progress and evaluate the performance of the algorithm, we need to report the log-likelihood in each iteration and calculate the prediction in the end. Write out the expressions of log-likelihood and prediction. You can use g_i, $g_{j|i}$ and P_{ij} in the expression, if any of them is in the expressions. But please be as specific as possible, that is, they could be directly used in the implementation.

3.3. Implementation of EM algorithm [17 pts]

In this part, we are going to implement EM algorithm. Please implement the algorithm in the notebook *Question_3.3.ipynb* and answer some following questions. You may follow Algorithm 1 in your implementation. *In your submission, please print out the notebook as a PDF file and put the PDF file as part of the homework, in the mean time, please also submit the raw jupyter notebook file with filename Question_3.3_yourandrewid.ipynb.*

Hint: the problem is designed based on the paper Jordan & Jacobs et al. 1994 [?]. You may get some ideas of how to solve the problem by reading the paper.

4 Logistic Regression [Bonus: 1 + 3 + 1 + 5 = 10 pts] (Yuanning)

(Note: This is a bonus problem that is optional.)

Logistic regression is a specific case of generalized Iinear model to model joint distribution between (y, x), with conditional distribution $y|x \sim \text{Bernoulli}(\mu)$, and logit link function $\beta^T x = \log\left(\frac{\mu}{1-\mu}\right)$. In this problem, we'll use logistic regression to classify a person's age group (under 40 or not) from his movie ratings. Given n *i.i.d.* samples $(y_1, x_1), ..., (y_n, x_n)$, we formulate the problem as a binary classification with output label $y \in \{0, 1\}^n$, corresponding to whether a person's age is under 40, and input features $X \in \mathbb{R}^{n\times(p+1)}$. The first column of X is taken to be 1_n to account for the intercept term. we solve the logistic regression problem using Iteratively Reweighted Least Squares (IRLS).

(a) Find the log likelihood of the samples $\ell(y, X, \beta)$.

(b) Assume a fixed step length $t = 1$, derive the IRLS update for β.

(c) Given X, y and t, write out the steps for performing IRLS to estimate $\hat{\beta}$.

(d) Now, implement IRLS using the movie data set on the website (in `Question4_data.zip`). You can stop IRLS when the change between consecutive objective values is less than 1e-6. Report both the train and test errors of classifying whether a person is under 40. Plot $f^{(k)}$ versus k, where $f^{(k)}$ denotes the objective value at outer iterations k of IRLS.
Note:

- As for the step size, the pure Newton's method uses $t = 1$. However, in practice we often use damped Newton's method $x^+ = x - t(\nabla^2 f(x))^{-1}\nabla f(x)$. At each Newton step, the step size t is typically chosen by using backtrack line search, with parameter $0 < a \leq 0.5$, and $0 < b < 1$. At each iteration, we start with $t = 1$, and while

$$f(x + tv) > f(x) + at\nabla f(x)^T v$$

  we shrink $t \leftarrow bt$, otherwise we perform the Newton update with the current t. Note that $v = -(\nabla^2 f(x))^{-1}\nabla f(x)$ is the Newton direction.
- Initialize your weights with zeros.
- Feel free to use Python/Matlab or any other language for your implementation.
- Please submit all your answer (including derivations, calculated values, figures) to this problem in the pdf file, the executable code and provided dataset in the *.zip file.