UC Berkeley
Department of Electrical Engineering and Computer Science
Department of Statistics

EECS 281A / STAT 241A Statistical Learning Theory

Solution to Problem Set 5
Fall 2012

Issued: Tues. Oct. 16, 2012 **Due:** Tues. Oct. 30, 2012

Reading: Chapters 9, 10, 11, 12

Problem 5.1
EM algorithm for hidden Markov models

(a) Implement the EM algorithm for HMMs with Gaussian emission probabilities $p(y_t \mid x_t)$, where $y_t \in \mathbb{R}^2$ and $x_t \in \{0, 1, \ldots, m-1\}$. Restrict the covariance matrices to be isotropic (i.e., $\Sigma = \sigma^2 I$).

Solution: Following equations (12.48)–(12.50) in the book with emission probability

$$p(y_t \mid x_t = i) = \frac{1}{2\pi\sigma_i^2} \exp\left(-\frac{(y_t - \mu_i)^\top (y_t - \mu_i)}{2\sigma_i^2}\right),$$

we can write the complete log likelihood of the data as

$$\begin{aligned}
\ell(\theta; x, y) &= \log p(x_0) + \sum_{t=0}^{T-1} \log p(x_{t+1} \mid x_t) + \sum_{t=0}^{T-1} \log p(y_t \mid x_t) \\
&= \sum_{i=1}^{M} q_0^i \log \pi_i + \sum_{t=0}^{T-1} \sum_{i,j=1}^{M} q_t^i q_{t+1}^j \log a_{ij} \\
&\quad + \sum_{t=0}^{T} \sum_{i=1}^{M} q_t^i \left(-\log 2\pi - \log \sigma_i^2 - \frac{1}{2\sigma_i^2}(y_t - \mu_i)^\top (y_t - \mu_i)\right).
\end{aligned}$$

Note that there is no $1/2$ in front of the $\log 2\pi$ and $\log \sigma_i^2$ since the Gaussians are 2-dimensional. This affected some people's M-step for updating the σ^2's, and caused their σ^2 updates to go to infinity.

We see that for the E-step we need to calculate $\gamma_t^i = p(x_t = i \mid y, \theta^{(p)})$ and $\xi_{t,t+1}^{i,j} = p(x_t = i, x_{t+1} = j \mid y, \theta^{(p)})$. We compute these using the forward-backward algorithm, as described in Chapter 12.

The M-step for updating A is the same as in the book. The M-step for updating the mean and covariance parameters is identical to the

M-step for a Gaussian (non HMM) mixture model. The M-steps are thus:

$$\begin{aligned}
\widehat{\pi}_i^{(p+1)} &= \gamma_0^i \\
\widehat{a}_{ij}^{(p+1)} &= \frac{\sum_{t=0}^{T-1} \xi_{t,t+1}^{i,j}}{\sum_{t=0}^{T-1} \gamma_t^i} \\
\widehat{\mu}_i^{(p+1)} &= \frac{\sum_{t=0}^{T} \gamma_t^i y_t}{\sum_{t=0}^{T} \gamma_t^i} \\
\widehat{\sigma_i^2}^{(p+1)} &= \frac{\sum_{t=0}^{T} \gamma_t^i (y_t - \widehat{\mu}_i^{(p+1)})^\top (y_t - \widehat{\mu}_i^{(p+1)})}{2 \sum_{t=0}^{T} \gamma_t^i}.
\end{aligned}$$

All code for this problem set is attached at the end of this document.

(b) Fit an HMM with $m = 4$ states to the two-dimensional data in *hmm-gauss.dat* and evaluate the log likelihood on the training and test data in *hmm-test.dat*. Plot the data together with the means of the component densities.

Solution: We run the EM algorithm for 5 trials (with random initializations) and choose the highest log likelihood of the training data. We then compute the log likelihood of the test data using the resulting parameters. The plots of the training and test data are shown in Figure 1, along with the component means and the circles indicating one standard deviation away from the mean. We also do the same procedure for the Gaussian mixture model in part (c) below; the plots are also shown in Figure 1.

The resulting estimated parameters for HMM are:

$$\pi = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \qquad \sigma^2 = \begin{pmatrix} 0.9902 \\ 0.9966 \\ 0.8624 \\ 1.0268 \end{pmatrix}, \qquad \mu = \begin{pmatrix} 1.9896 & \text{-1.9655} \\ \text{-2.0398} & 1.9596 \\ 1.9667 & 2.0889 \\ \text{-1.9979} & \text{-2.1501} \end{pmatrix},$$

$$A = \begin{pmatrix} 0.7266 & 0.0742 & 0.0543 & 0.1448 \\ 0.1332 & 0.6289 & 0.1161 & 0.1217 \\ 0.1211 & 0.0833 & 0.6524 & 0.1432 \\ 0.0870 & 0.0956 & 0.0994 & 0.7180 \end{pmatrix}.$$

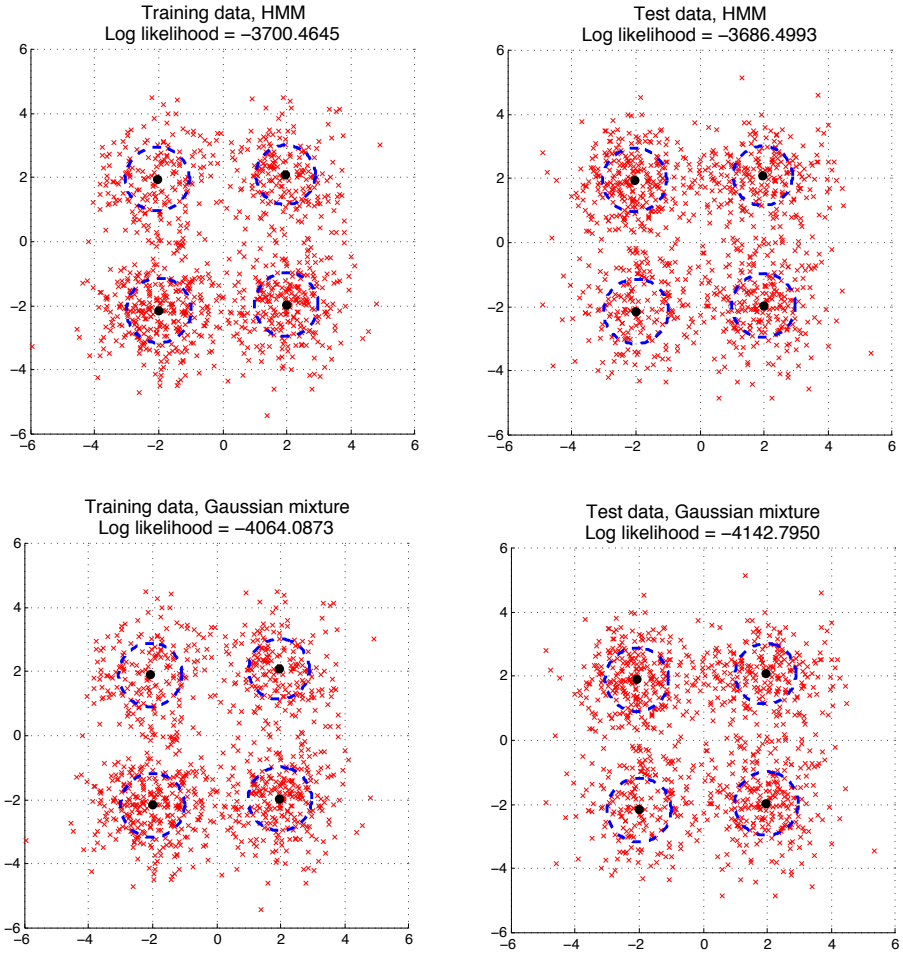


Figure 1: Results of the HMM and Gaussian mixture models.

(c) Fit a Gaussian mixture model with 4 states to the same data (again with isotropic covariance matrices $\sigma^2 I$). Compare the performance with that of the HMM.

Solution: The complete log likelihood for the Gaussian mixture model is

$$\ell(\theta; x, y) = \sum_{t=0}^{T}\sum_{i=1}^{M} x_t^i \log \pi_i - \sum_{t=0}^{T}\sum_{i=1}^{M} x_t^i \frac{(y_t - \mu_i)^\top (y_t - \mu_i)}{2\sigma_i^2} - \sum_{t=0}^{T}\sum_{i=1}^{M} x_t^i \log \sigma_i^2 - T\log(2\pi),$$

so the expected complete log likelihood is

$$\mathbb{E}[\ell(\theta; x, y)] = \sum_{t=0}^{T}\sum_{i=1}^{M} \gamma_t^i \log \pi_i - \sum_{t=0}^{T}\sum_{i=1}^{M} \gamma_t^i \frac{(y_t - \mu_i)^\top (y_t - \mu_i)}{2\sigma_i^2} - \sum_{t=0}^{T}\sum_{i=1}^{M} \gamma_t^i \log \sigma_i^2 - T\log(2\pi).$$

E step. The posterior probabilities are

$$\gamma_t^i = p(x_t = i \mid y_t) = \frac{\pi_i \, \frac{1}{2\pi\sigma_i^2} \exp\left(-\frac{(y_t-\mu_i)^\top (y_t-\mu_i)}{2\sigma_i^2}\right)}{\sum_{j=1}^{M} \pi_j \, \frac{1}{2\pi\sigma_j^2} \exp\left(-\frac{(y_t-\mu_j)^\top (y_t-\mu_j)}{2\sigma_j^2}\right)}.$$

M step. By the same argument as in part **(a)**, we have the update rules

$$\begin{aligned}
\widehat{\pi}^{(p+1)} &= \frac{1}{T}\sum_{t=0}^{T} \gamma_t \\
\widehat{\mu}_i^{(p+1)} &= \frac{\sum_{t=0}^{T} \gamma_t^i y_t}{\sum_{t=0}^{T} \gamma_t^i} \\
\widehat{\sigma_i^2}^{(p+1)} &= \frac{\sum_{t=0}^{T} \gamma_t^i (y_t - \widehat{\mu}_i^{(p+1)})^\top (y_t - \widehat{\mu}_i^{(p+1)})}{2\sum_{t=0}^{T} \gamma_t^i}.
\end{aligned}$$

The resulting log likelihoods are:

	training	test
HMM	−3700.46	−3686.50
Gaussian mixture	−4064.09	−4142.80

We see that the HMM yields higher log likelihood, which is to be expected since Gaussian mixture model is a special case of the HMM (when the transition probability $p(x_{t+1} \mid x_t)$ is independent of x_t). That is, HMM is a more expressive model, so it should be able to fit the data better.

Problem 5.2

EM and missing values: Suppose you have a random sample of twins and are interested in studying *identical* twins. However, you observe only:

- $m \equiv$ the total number of male twins (both identical and fraternal)
- $f \equiv$ the total number of female twins, and
- $b \equiv$ the number of twins of opposite gender.

Let θ be the probability that a pair of twins are identical. Assume that, given identical twins, the probability the twins are male is p. Given fraternal twins, assume the number of males is Binomial$(2, q)$.

Give an algorithm for calculating the MLEs for θ, p, and q. (*Hint*: If you knew exactly how many identical male and female twins there are, then the MLEs would be easy to calculate.)

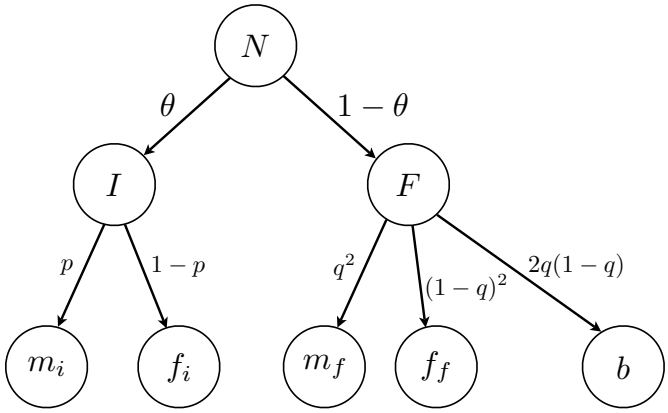
Solution: We introduce the following latent variables:

$$\begin{aligned} m_i &= \#\text{ identical male twins} \\ f_i &= \#\text{ identical female twins} \\ m_f &= \#\text{ fraternal male twins} \\ f_f &= \#\text{ fraternal female twins.} \end{aligned}$$

For convenience, we write

$$\begin{aligned} I &= m_i + f_i = \#\text{ identical twins} \\ F &= m_f + f_f = \#\text{ fraternal twins} \\ N &= m + f + b = \#\text{ twins in total.} \end{aligned}$$

The generative process is described by the following diagram:



Note that this is not a graphical model describing the distribution. The graphical model has "cliques" $\{N, I, F\}$, $\{I, m_i, f_i\}$, and $\{F, m_f, f_f, b\}$.

The complete data likelihood is

$$\begin{aligned} L(\theta, p, q \mid \mathcal{D}) &= \mathbb{P}(I, F \mid N)\, \mathbb{P}(m_i, f_i \mid I, N)\, \mathbb{P}(m_f, f_f, b \mid F, N) \\ &= \left\{ \binom{N}{m_i + f_i} \theta^{m_i + f_i} (1-\theta)^{m_f + f_f + b} \right\} \left\{ \binom{I}{m_i} p^{m_i} (1-p)^{f_i} \right\} \\ &\quad \times \left\{ \binom{F}{m_f, f_f, b} q^{2m_f} (1-q)^{2f_f} (2q(1-q))^b \right\} \\ &= C\, \theta^{m_i + f_i}\, (1-\theta)^{m_f + f_f + b}\, p^{m_i}\, (1-p)^{f_i}\, q^{2m_f + b}\, (1-q)^{2f_f + b}, \end{aligned}$$

where $C \equiv C(m_i, f_i, m_f, f_f, b)$ does not depend on the parameters θ, p, q. Taking the log, we see that the log likelihood is linear in m_i, f_i, m_f, f_f, b, so they are an appropriate set of sufficient statistics. Note that we have already written the likelihood in a form where the parameters are "decoupled" to make maximization easy.

E-step: For the E-step, we need to calculate the expected value of each of these sufficient statistics. For notational convenience, assume all expectations are taken with respect to the current estimate of the parameters.

Reading from the diagram or by algebraic manipulation, we see that

$$\begin{aligned} m_i &\sim \text{Multinomial}(m+f+b,\ \theta p) \\ f_i &\sim \text{Multinomial}(m+f+b,\ \theta(1-p)) \\ m_f &\sim \text{Multinomial}(m+f+b,\ (1-\theta)q^2) \\ f_f &\sim \text{Multinomial}(m+f+b,\ (1-\theta)(1-q)^2) \\ b &\sim \text{Multinomial}(m+f+b,\ 2(1-\theta)q(1-q)). \end{aligned}$$

This implies for any particular set of twins,

$$\mathbb{P}(\text{identical} \mid \text{male}) = \frac{\mathbb{P}(\text{male AND identical})}{\mathbb{P}(\text{male})} = \frac{p\theta}{p\theta + q^2(1-\theta)}.$$

We can calculate similar results given the twins are female or mixed male/female, so we obtain

$$\begin{aligned} \mathbb{E}[m_i \mid m] &= \frac{mp\theta}{p\theta + q^2(1-\theta)} \\ \mathbb{E}[f_i \mid f] &= \frac{m(1-p)\theta}{(1-p)\theta + (1-q)^2(1-\theta)} \\ \mathbb{E}[m_f \mid m] &= m - \mathbb{E}[m_i \mid m] \\ \mathbb{E}[f_f \mid f] &= f - \mathbb{E}[f_i \mid f]. \end{aligned}$$

It is important to actually calculate these expectations and, in this case, to give them in closed form. Some of the main reasons one uses EM are that (1) it is often easy to implement, and (2) the iterations are fast so EM runs in a reasonable amount of time. Leaving the E-step in terms of the conditional probabilities (the maximizing auxiliary distribution) means one has to sum over a set of values, which makes EM too slow.

Of those that calculated expectations, there were two common mistakes. The first was saying $\mathbb{E}[m_i \mid m, f, b] = (m+f+b)p\theta$. This is incorrect since it only uses information about the total number of twins and does not make use of the additional information provided by the split into m, f, b. The second mistake was saying $\mathbb{E}[m_i \mid m, f, b] = m\theta$. There is some subtlety in the generative process since fraternal male twins require two "coin flips" versus one for the identical male twins, and the coin flips may have different probabilities.

M-step: Returning to the complete data likelihood, we see that it has the same form as the product of three binomials with parameters θ, p, q.

Therefore, maximizing the expected complete data (log-)likelihood is trivial:

$$\theta \leftarrow \frac{\mathbb{E}[m_i \mid m] + \mathbb{E}[f_i \mid f]}{m + f + b}$$
$$p \leftarrow \frac{\mathbb{E}[m_i \mid m]}{\mathbb{E}[m_i \mid m] + \mathbb{E}[f_i \mid f]}$$
$$q \leftarrow \frac{2\mathbb{E}[m_f \mid m] + b}{2\mathbb{E}[m_f \mid m] + 2\mathbb{E}[f_f \mid f] + 2b} = \frac{\text{expected \# fraternal males}}{\text{expected \# fraternal siblings}}.$$

A common mistake was to forget about or incorrectly including b in the update for q.

Problem 5.3

(Social network analysis and IPF:) Frank is studying dependencies in voting patterns of a collection of d US senators. For any given bill, he collects a vector $x \in \{0,1\}^d$, where $x_i = 1$ means that senator i voted yes on that bill. He models the random vector $(X_1, X_2, \ldots, X_d)$ as a pairwise Markov random field

$$\mathbb{P}_\theta(x_1, x_2, \ldots, x_d) \quad \propto \prod_{(s,t)\in E} \exp(\theta_{st}(x_s, x_t)).$$

(a) For a set $d = 4$ senators, the data file `Pairwise.dat` contains a 4×30 matrix, summarizing the data from $n = 30$ bills that were voted on in the senate. Implement and apply the IPF updates to estimate the model parameters for each of the following graphs: (i) the graph with edge set $E = \{(12), (23), (34), (14)\}$; and (ii) the graph with edge set $E = \{(12), (23), (13), (14)\}$; and (iii) the fully connected graph with all $\binom{4}{2}$ edges.

Solution: The code for the implementation of the IPF algorithm is provided at the end of this document. We choose to initialize the potential functions as $\theta_{st}(x_s, x_t) = 0$ for all $(s,t) \in E$ so that the normalization constant can be computed explicitly, $Z = 2^4 = 16$. Note that to compute the marginals $p^{(t)}(x_s, x_t)$ we probably should use algorithms such as junction tree, but considering that we are only working with graphs with 4 nodes, we can simply perform the marginalization directly. We also note that the values of θ_{st} are unnormalized, i.e. we can add any constant to any θ_{st} without changing the probability distribution.

(i) $E = \{(12), (23), (34), (14)\}$.

Algorithm converges after 6 batch iterations.[1] Log likelihood = -79.6370. Estimated parameters:

		(x_s, x_t)			
		(0,0)	(0,1)	(1,0)	(1,1)
$\theta_{st}(x_s, x_t)$	(12)	0.2880	−0.2237	−0.2237	0.0650
	(23)	0.1180	−0.1338	−0.8478	0.4521
	(34)	−0.0114	0.0116	0.0077	−0.0076
	(14)	0.1183	−0.1342	−0.1549	0.1341

(ii) $E = \{(12), (23), (13), (14)\}$.

Algorithm converges after 21 batch iterations. Log likelihood = -78.3564. Estimated parameters:

		(x_s, x_t)			
		(0,0)	(0,1)	(1,0)	(1,1)
$\theta_{st}(x_s, x_t)$	(12)	0.3923	−0.3874	−0.4051	0.2113
	(23)	0.2049	−0.2405	−1.0487	0.5105
	(13)	−0.4326	0.2454	0.3998	−0.3247
	(14)	0.1178	−0.1335	−0.1542	0.1335

(iii) $E = \{(12), (13), (14), (23), (24), (34)\}$.

Algorithm converges after 32 batch iterations. Log likelihood = -76.5029. Estimated parameters:

		(x_s, x_t)			
		(0,0)	(0,1)	(1,0)	(1,1)
$\theta_{st}(x_s, x_t)$	(12)	0.4706	−0.5140	−0.5232	0.3103
	(13)	−0.7243	0.4423	0.2229	−0.2071
	(14)	0.2392	−0.2826	−0.3229	0.2668
	(23)	0.4821	−0.4835	−0.9560	0.3961
	(24)	−0.4571	0.3348	0.3720	−0.5313
	(34)	0.2654	−0.2393	−0.1632	0.1728

[1] One batch iteration is a cycle through all the edges and all the possible configurations of the edges.

(b) Of models (i) and (ii), which model has a higher likelihood?

Solution: Model (ii) has a higher likelihood.

(c) Of all three models (i), (ii) and (iii), which has the highest likelihood? Do you think that it is the "best" model?

Solution: Model (iii) has the highest likelihood. However, this is somewhat vacuous; since it contains all of the edges in both models (i) and (ii), plus additional edges, it is a strictly more expressive model than either of the other two and can capture more complex relationships in the training data. This says nothing about how well it generalizes, as we may have overfit by using such a powerful model without any prior or form of regularization. In practice, we would usually use criteria like the BIC or the AIC to balance the conflicting considerations of maximizing the data likelihood and choosing a model that will generalize well, or employ some expert information to start with a graph that makes practical sense.

Problem 5.4

Model selection for trees: Recall that for a given tree T with edge set $E(T)$, the MLE for the exponential parameters takes the form

$$\widehat{\theta}_s(x_s) = \log \widehat{\mu}_s(x_s) \quad \text{for all } s \in V, \text{ and}$$
$$\widehat{\theta}_{st}(x_s, x_t) = \log \frac{\widehat{\mu}_{st}(x_s, x_t)}{\widehat{\mu}_s(x_s)\widehat{\mu}_t(x_t)} \quad \text{for all } (s,t) \in E(T).$$

Here $\widehat{\mu}$ are the empirical marginals computed from the data (e.g., $\widehat{\mu}_{st}(j,k) = \frac{1}{n}\sum_{i=1}^n \mathbb{I}_{st;jk}(x_{is}, x_{it})$), and we assume that they take strictly positive values.

(a) Define the rescaled log likelihood $\ell(\theta) = \frac{1}{n}\sum_{i=1}^n \log \mathbb{P}_\theta(x_{i\cdot})$. Letting $\widehat{\theta}(T)$ denote the MLE for trees, show that $\ell(\widehat{\theta}(T))$ depends on the empirical marginals only via

$$\text{Singleton entropy:} \quad H(\widehat{\mu}_s) = -\sum_{x_s} \widehat{\mu}_s(x_s) \log \widehat{\mu}_s(x_s) \quad \forall\, s \in V, \text{ and}$$
$$\text{Joint edge entropy:} \quad H(\widehat{\mu}_{st}) = -\sum_{x_s, x_t} \widehat{\mu}_{st}(x_s, x_t) \log \widehat{\mu}_{st}(x_s, x_t) \quad \forall\, (s,t) \in E(T).$$

Solution: Suppose all the random variables X_s take values in a finite set $\mathcal{X}$. Note that for each $1 \le i \le n$ and $s \in V$, we can write

$$\widehat{\theta}_s(x_{is}) = \log \widehat{\mu}_s(x_{is}) = \sum_{j \in \mathcal{X}} \mathbb{I}_{s;j}(x_{is}) \log \widehat{\mu}_s(j),$$

so

$$\frac{1}{n}\sum_{i=1}^{n}\widehat{\theta}_s(x_{is}) = \frac{1}{n}\sum_{j\in\mathcal{X}}\sum_{i=1}^{n}\mathbb{I}_{s;j}(x_{is})\log\widehat{\mu}_s(j) = \sum_{j\in\mathcal{X}}\widehat{\mu}_s(j)\log\widehat{\mu}_s(j) = -H(\widehat{\mu}_s).$$

Similarly, since for $1 \le i \le n$ and $(s,t) \in E$ we have

$$\begin{aligned}\widehat{\theta}_{st}(x_{is},x_{it}) &= \log\frac{\widehat{\mu}_{st}(x_{is},x_{it})}{\widehat{\mu}_s(x_{is})\widehat{\mu}_t(x_{it})} \\ &= \sum_{j,k\in\mathcal{X}}\mathbb{I}_{st;jk}(x_{is},x_{it})\log\widehat{\mu}_{st}(j,k) - \sum_{j\in\mathcal{X}}\mathbb{I}_{s;j}(x_{is})\log\widehat{\mu}_s(j) \\ &\quad - \sum_{k\in\mathcal{X}}\mathbb{I}_{t;k}(x_{it})\log\widehat{\mu}_t(k),\end{aligned}$$

we can also write

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\widehat{\theta}_{st}(x_{is},x_{it}) &= \frac{1}{n}\sum_{j,k\in\mathcal{X}}\sum_{i=1}^{n}\mathbb{I}_{st;jk}(x_{is},x_{it})\log\widehat{\mu}_{st}(j,k) \\ &\qquad - \frac{1}{n}\sum_{j\in\mathcal{X}}\sum_{i=1}^{n}\mathbb{I}_{s;j}(x_{is})\log\widehat{\mu}_s(j) - \frac{1}{n}\sum_{k\in\mathcal{X}}\sum_{i=1}^{n}\mathbb{I}_{t;k}(x_{it})\log\widehat{\mu}_t(k) \\ &= \sum_{j,k\in\mathcal{X}}\widehat{\mu}_{st}(j,k)\log\widehat{\mu}_{st}(j,k) - \sum_{j\in\mathcal{X}}\widehat{\mu}_s(j)\log\widehat{\mu}_s(j) - \sum_{k\in\mathcal{X}}\widehat{\mu}_t(k)\log\widehat{\mu}_t(k) \\ &= -H(\widehat{\mu}_{st}) + H(\widehat{\mu}_s) + H(\widehat{\mu}_t).\end{aligned}$$

Using the relations above, we can write down the rescaled log-likelihood for the MLE as follows:

$$\begin{aligned}\ell(\widehat{\theta}(T)) &= \frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{1}{Z}\prod_{s\in V}\exp(\widehat{\theta}_s(x_{is}))\prod_{(s,t)\in E}\exp(\widehat{\theta}_{st}(x_{is},x_{it})\right) \\ &= -\log Z + \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{s\in V}\widehat{\theta}_s(x_{is}) + \sum_{(s,t)\in E}\widehat{\theta}_{st}(x_{is},x_{it})\right) \\ &= -\log Z - \sum_{s\in V}H(\widehat{\mu}_s) + \sum_{(s,t)\in E}\big(-H(\widehat{\mu}_{st}) + H(\widehat{\mu}_s) + H(\widehat{\mu}_t)\big) \\ &= -\log Z + \sum_{s\in V}(\deg(s)-1)H(\widehat{\mu}_s) - \sum_{(s,t)\in E}H(\widehat{\mu}_{st}).\end{aligned}$$

In the equations above, Z is a normalizing constant (which is actually equal to 1 in this case) and $\deg(s)$ is the degree of vertex s in the tree.

This shows that $\ell(\widehat{\theta}(T))$ only depends on the empirical marginals via the singleton and joint edge entropy. Alternatively, we can also write $\ell(\widehat{\theta}(T))$ as

$$\ell(\widehat{\theta}(T)) = -\log Z - \sum_{s \in V} H(\widehat{\mu}_s) + \sum_{(s,t) \in E} I(\widehat{\mu}_{st}; \widehat{\mu}_s, \widehat{\mu}_t), \qquad (1)$$

where $I(\widehat{\mu}_{st}; \widehat{\mu}_s, \widehat{\mu}_t) = H(\widehat{\mu}_s) + H(\widehat{\mu}_t) - H(\widehat{\mu}_{st})$ is the mutual information.

(b) In the model selection problem for trees, the goal is to choose, from *all trees on d nodes*, the highest likelihood tree i.e., $\widehat{T} \in \arg\max_T \ell(\widehat{\theta}(T))$. Show how this problem can be cast as a maximum weight spanning tree calculation. This is important, because it allows us to select the best tree by simple algorithms. (*Hint:* The mutual information $I(\widehat{\mu}_{st}; \widehat{\mu}_s, \widehat{\mu}_t) = H(\widehat{\mu}_s) + H(\widehat{\mu}_t) - H(\widehat{\mu}_{st})$ should be relevant in your calculations.)

Solution: Using the representation (1), we see that the term $-\log Z - \sum_{s \in V} H(\widehat{\mu}_s)$ is independent of the choice of the tree, so

$$\widehat{T} \in \arg\max_T \ell(\widehat{\theta}(T)) \quad \Longleftrightarrow \quad \widehat{T} \in \arg\max_T \sum_{(s,t) \in E} I(\widehat{\mu}_{st}; \widehat{\mu}_s, \widehat{\mu}_t).$$

Therefore, finding $\widehat{T}$ is equivalent to finding a maximum weight spanning tree in the complete graph with vertices V, in which each edge (s,t) is assigned weight $I(\widehat{\mu}_{st}; \widehat{\mu}_s, \widehat{\mu}_t)$.

(c) Use the technique from (b) to select the best fitting tree for the data in `Pairwise.dat`.

Solution: The script for computing the mutual information $I(\widehat{\mu}_{st}; \widehat{\mu}_s, \widehat{\mu}_t)$ is attached at the end of this document. Figure 2 shows the weighted complete graph with the mutual information as the edge weights. The maximum weight spanning tree, which can be found by inspection (or by running Kruskal's algorithm), is shown with solid edges.

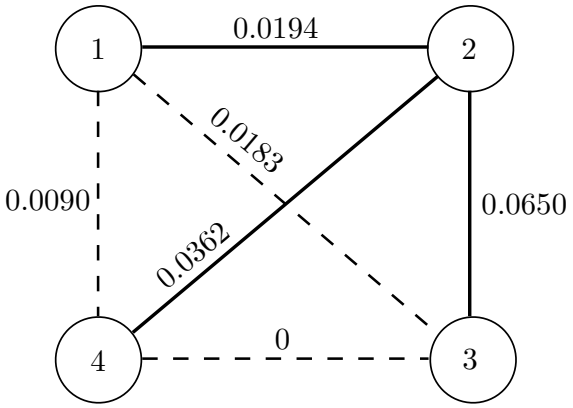


Figure 2: Weighted graph for Problem 5.4(c), with the mutual information $I(\widehat{\mu}_{st}; \widehat{\mu}_s, \widehat{\mu}_t)$ as the edge weights. The maximum weight spanning tree is shown with solid edges.

Code for Problem 5.1

```
%% Script for running the computations for problem 5.1

load hmm-gauss.dat
load hmm-test.dat

numTrials = 5;  % number of times to run EM


%% Fit a HMM with 4 states

% Initialize log likelihood to be a very small number
logLhoodHMM = -1e15;

% Run the EM algorithm numTrials times, choose the highest log likelihood
for i = 1:numTrials
    [pi0_, A_, mu_, sigma2_, logLhood_] = EM_HMM(hmm_gauss, 4);
    if (logLhood_ > logLhoodHMM)
        logLhoodHMM = logLhood_;
        pi0HMM = pi0_;
        A = A_;
        muHMM = mu_;
        sigma2HMM = sigma2_;
    end
end

```

```
% Compute log likelihood of the test data
[¬, ¬, logLhoodHMM_test] = FwBw(hmm_test, pi0HMM, A, muHMM, sigma2HMM);


%% Fit a Gaussian mixture model with 4 states

% Initialize log likelihood to be a very small number
logLhoodMix = -1e15;

% Run the EM algorithm numTrials times, choose the highest log likelihood
for i = 1:numTrials
    [pi0_, mu_, sigma2_, logLhood_] = EM_GaussianMix(hmm_gauss, 4);
    if (logLhood_ > logLhoodMix)
        logLhoodMix = logLhood_;
        pi0Mix = pi0_;
        muMix = mu_;
        sigma2Mix = sigma2_;
    end
end

% Compute log likelihood of the test data
logLhoodMix_test = computeLogLhoodGaussianMix(hmm_test, pi0Mix, muMix, sigma2Mix);


%% Plot training and test data

% Plot training data for HMM
figure;
scatter(hmm_gauss(:,1), hmm_gauss(:,2), 'xr');
title(sprintf('Training data, HMM\nLog likelihood = %.4f', ...
    logLhoodHMM), 'FontSize', 13);
hold on; grid;
axis([-3.5 3.5 -3.5 3.5]); axis square;
scatter(muHMM(:,1), muHMM(:,2), 50, 'ok', 'filled');
for i = 1:4
    z = (0:255)*2*pi/255;
    x = sqrt(sigma2HMM(i))*cos(z) + muHMM(i,1);
    y = sqrt(sigma2HMM(i))*sin(z) + muHMM(i,2);
    plot(x, y, '--b', 'LineWidth', 2);
end


% Plot test data for HMM
figure;
scatter(hmm_test(:,1), hmm_test(:,2), 'xr');
title(sprintf('Test data, HMM\nLog likelihood = %.4f', ...
    logLhoodHMM_test), 'FontSize', 13);
hold on; grid;
axis([-3.5 3.5 -3.5 3.5]); axis square;
```

```
scatter(muHMM(:,1), muHMM(:,2), 50, 'ok', 'filled');
for i = 1:4
    z = (0:255)*2*pi/255;
    x = sqrt(sigma2HMM(i))*cos(z) + muHMM(i,1);
    y = sqrt(sigma2HMM(i))*sin(z) + muHMM(i,2);
    plot(x, y, '--b', 'LineWidth', 2);
end


% Plot training data for Gaussian mixture model
figure;
scatter(hmm_gauss(:,1), hmm_gauss(:,2), 'xr');
title(sprintf('Training data, Gaussian mixture\nLog likelihood = %.4f', ...
    logLhoodMix), 'FontSize', 13);
hold on; grid;
axis([-3.5 3.5 -3.5 3.5]); axis square;
scatter(muMix(:,1), muMix(:,2), 50, 'ok', 'filled');
for i = 1:4
    z = (0:255)*2*pi/255;
    x = sqrt(sigma2Mix(i))*cos(z) + muMix(i,1);
    y = sqrt(sigma2Mix(i))*sin(z) + muMix(i,2);
    plot(x, y, '--b', 'LineWidth', 2);
end


% Plot test data for Gaussian mixture model
figure;
scatter(hmm_test(:,1), hmm_test(:,2), 'xr');
title(sprintf('Test data, Gaussian mixture\nLog likelihood = %.4f', ...
    logLhoodMix_test), 'FontSize', 13);
hold on; grid;
axis([-3.5 3.5 -3.5 3.5]); axis square;
scatter(muMix(:,1), muMix(:,2), 50, 'ok', 'filled');
for i = 1:4
    z = (0:255)*2*pi/255;
    x = sqrt(sigma2Mix(i))*cos(z) + muMix(i,1);
    y = sqrt(sigma2Mix(i))*sin(z) + muMix(i,2);
    plot(x, y, '--b', 'LineWidth', 2);
end
```

The code for implementing EM for HMM:

```
function [pi0, A, mu, sigma2, logLhood] = EM_HMM(data, M)
% M is the number of states in the latent variable

% data is T-by-K
T = size(data,1); % number of iid observations
K = size(data,2); % dimensionality of the Gaussian
```

```
7
8  % Initialize cluster probabilities to be uniform 1/M
9  pi0 = ones(M,1)/M;
10
11 % Initialize transition probabilities to be uniform
12 A = ones(M,M)/M;
13
14 % Initialize mean values to be random data points -- mu is M-by-K
15 mu = data(randi(T,M,1), :);
16
17 % Initialize sigma^2 to be the average distance between the data points and
18 % one random data point
19 centered = data - data(randi(T,1,1)*ones(T,1), :);
20 sigma2_init = (sum(sum(centered.^2))/(2*T));
21 sigma2 = sigma2_init*ones(M,1);
22
23 % Compute gamma and xi via forward-backward
24 [gamma, xi, logLhood] = FwBw(data, pi0, A, mu, sigma2);
25
26 % Maximum number of iterations
27 maxIter = 1000;
28
29 % Iterate over the E and M steps
30 for iter = 1:maxIter
31     % Update parameters using gamma and xi
32     pi0 = gamma(:,1);
33
34     % Update transition probabilities matrix
35     for i = 1:M
36         for j = 1:M
37             A(i,j) = sum(xi(i,j,:))/sum(gamma(i,1:end-1));
38         end
39     end
40
41     % Update mean parameters
42     for i = 1:M
43         mu(i,:) = sum(data.*gamma(i*ones(1,K),:)')/sum(gamma(i,:));
44     end
45
46     % Update variance parameters - use the new mu values
47     for i = 1:M
48         centered = data - mu(i*ones(T,1), :); % centered data
49         sigma2(i) = sum(sum(centered.^2,2).*gamma(i,:)')/(2*sum(gamma(i,:)));
50     end
51
52     % Then recompute gamma, xi, and log likelihood
53     logOld = logLhood;
54     [gamma, xi, logLhood] = FwBw(data, pi0, A, mu, sigma2);
55
```

```
    % Check for convergence
    if (logLhood - logOld < 1e-14)
        fprintf('Converges after %d batch iterations!\n', iter);
        break;
    elseif (iter == maxIter)
        fprintf('EM did not converge: final Δ = %.8f\n', logLhood - logOld);
    end
end
```

A helper function that computes the γ_t and $\xi_{t,t+1}$ variables, as well as the log likelihood of the data, using the alpha-gamma recursion.

```
function [gamma, xi, logLhood] = FwBw(data, pi0, A, mu, sigma2)
% Computes posterior probabilities using alpha-gamma recursion

M = length(pi0);  % number of states in the latent variables
T = size(data,1); % number of observations

% Compute the alpha variables
% Note: we always normalize so sum of alpha(q_t) over q_t is = 1
% But keep track of the normalizers to compute p(y)
alpha = zeros(M,T);
normalizer = zeros(T,1);

% First initialize alpha(q_0)
centered = data(ones(M,1),:) - mu;
alpha(:,1) = (pi0.*exp(-sum(centered.^2,2)./(2*sigma2))).(2*pi*sigma2);
normalizer(1) = sum(alpha(:,1));
alpha(:,1) = alpha(:,1)/normalizer(1);

% Then move forward in time
for t = 1:T-1
    % Compute p(y_{t+1}|q_{t+1}=i) for i=1..M
    centered = data((t+1)*ones(M,1),:) - mu;
    post = exp(-sum(centered.^2,2)./(2*sigma2))./(2*pi*sigma2);

    % Compute \sum_{q_t} alpha(q_t)*a_{q_t,q_{t+1}=i) for i=1..M
    c = sum(alpha(:,t*ones(1,M)).*A,1)';

    % Store value of alpha
    alpha(:,t+1) = post.*c;
    normalizer(t+1) = sum(alpha(:,t+1));
    alpha(:,t+1) = alpha(:,t+1)/normalizer(t+1);
end

% Compute the gamma variables
gamma = zeros(M,T);
gamma(:,T) = alpha(:,T);
```

```
37
38 % Then move backward in time
39 for t = (T-1):(-1):1
40     % Compute alpha(q_t)*a_{q_t,q_{t+1}} and normalize the sum
41     c = alpha(:,t*ones(1,M)).*A;
42     sum_c = sum(c,1);
43     c = c./sum_c(ones(M,1),:);
44
45     % Store value of gamma
46     gamma(:,t) = sum(c.*gamma(:,(t+1)*ones(1,M))',2);
47 end
48
49 % Compute the marginal log likelihood
50 logLhood = sum(log(normalizer));
51
52 % Compute the xi variables
53 xi = zeros(M,M,T-1);
54 for t = 1:T-1
55     % Compute p(y_{t+1}|q_{t+1}=j) for j=1..M
56     centered = data((t+1)*ones(M,1),:) - mu;
57     post = exp(-sum(centered.^2,2)./(2*sigma2))./(2*pi*sigma2);
58
59     for i = 1:M
60         xi(i,:,t) = (alpha(i,t)*(post.*gamma(:,t+1).*A(i,:)')./alpha(:,t+1))';
61     end
62
63     % Need to normalize the xi since we rescaled alpha
64     xi(:,:,t) = xi(:,:,t)/sum(sum(xi(:,:,t)));
65 end
```

The code for implementing EM for Gaussian mixture model:

```
1  function [pi0, mu, sigma2, logLhood] = EM_GaussianMix(data, M)
2  % M is the number of states in the latent variable
3
4  % data is T-by-K
5  T = size(data,1); % number of iid observations
6  K = size(data,2); % dimensionality of the Gaussian
7
8  % Initialize cluster probabilities to be uniform 1/M
9  pi0 = ones(M,1)/M;
10
11 % Initialize mean values to be random data points -- mu is M-by-K
12 mu = data(randi(T,M,1), :);
13
14 % Initialize sigma^2 to be the average distance between the data points and
15 % one random data point
16 centered = data - data(randi(T,1,1)*ones(T,1), :);
```

```
sigma2_init = (sum(sum(centered.^2))/(2*T));
sigma2 = sigma2_init*ones(M,1);

% Compute log likelihood of the data
logLhood = computeLogLhoodGaussianMix(data, pi0, mu, sigma2);

% Maximum number of iterations
maxIter = 1000;

% Iterate over the E and M steps
for iter = 1:maxIter
    % Compute the posterior probabilities, each row for one q_t
    post = zeros(T,M);
    for t = 1:T
        % Compute y_t-mu_i for i = 1..M
        centered = data(t*ones(M,1),:) - mu;

        % Compute joint probability for y_t:
        % p(y_t,q_t=i) = p(q_t=i)*p(y_t|q_t=1) for i=1..M
        joint_yt = (pi0.*exp(-sum(centered.^2,2)./(2*sigma2)))./(2*pi*sigma2);

        % Normalize and store to post
        post(t,:) = joint_yt'/sum(joint_yt);
    end

    % Then update parameters
    pi0 = sum(post,1)'/T;

    % Update mean parameters
    for i = 1:M
        mu(i,:) = sum(data.*post(:,i*ones(1,K)))/sum(post(:,i));
    end

    % Update variance parameters - use the new mu values
    for i = 1:M
        centered = data - mu(i*ones(T,1), :); % centered data
        sigma2(i) = sum(sum(centered.^2,2).*post(:,i))/(2*sum(post(:,i)));
    end

    % Compute new log likelihood of the data
    logOld = logLhood;
    logLhood = computeLogLhoodGaussianMix(data, pi0, mu, sigma2);

    % Check for convergence
    if (logLhood - logOld < 1e-14)
        fprintf('Converges after %d batch iterations!\n', iter);
        break;
    elseif (iter == maxIter)
        fprintf('EM did not converge: final Δ = %.8f\n', logLhood - logOld);
```

```
    end
end
```

A helper function that computes the log likelihood of the data given the parameters. Note that in this case the log likelihood is easy to compute since the observations are i.i.d., so

$$p(y) = \prod_{t=0}^{T} p(y_t) = \prod_{t=0}^{T} \sum_{i=1}^{M} p(y_t, x_t = i).$$

```
function logLhood = computeLogLhoodGaussianMix(data, pi0, mu, sigma2)

% Candidate for the log likelihood log(p(data))
logLhood = 0;

% Compute marginal p(y_t) for each t separately
for t = 1:size(data,1)
    % Compute y_t - mu_i for i = 1..M
    centered = data(t*ones(size(mu,1),1),:) - mu;

    % Compute joint probability for y_t:
    % p(y_t,q_t=i) = p(q_t=i)*p(y_t|q_t=1) for i=1..M
    joint_yt = (pi0.*exp(-sum(centered.^2,2)./(2*sigma2)))./(2*pi*sigma2);

    % Then compute marginal probability p(y_t) and add to total
    logLhood = logLhood + log(sum(joint_yt));
end
```

Code for Problem 5.3

```
function [logLhood Theta] = IPF(data, edges)
% Implement the IPF algorithm in the pairwise Markov random field model
% where the cliques are edges.

% data is M-by-N, assumed binary
% M is the number of nodes in the graph
% N is the number of iid observations
[M N] = size(data);

% edges is a cell, each entry is of the form (s,t) for some 1 ≤ s < t ≤ N
K = length(edges);  % number of edges in the graph
for k = 1:K         % make sure s < t
    edges{k} = [min(edges{k}) max(edges{k})];
end

```

```
% Compute empirical distributions: Emp(i,j,k) is the fraction of the data
% that has (X_s, X_t) = (i-1, j-1), where (s,t) = edges{k}.
Emp = zeros(2,2,K);
for k = 1:K
    % Find the nodes of the edge
    s = edges{k}(1);
    t = edges{k}(2);

    % Iterate over the observations and update the appropriate entry of Emp
    for n = 1:N
        i = data(s,n) + 1;
        j = data(t,n) + 1;
        Emp(i,j,k) = Emp(i,j,k) + 1/N;
    end
end

% Initialize the parameters: Theta(i,j,k) encodes theta_st(X_s=i-1,
% X_t=j-1), where (s,t) = edges{k}.
Theta = zeros(2,2,K);

% In this case the normalizer Z can be computed explicitly
Z = 2^M;

% Compute initial log likelihood
logLhood = N*sum(sum(sum(Emp.*Theta))) - N*log(Z);

% Iterate over all edges and configurations
maxIter = 100; % maximum number of batch iterations
for iter = 1:maxIter
    % Iterate over the edges
    for k = 1:K
        % Find the nodes of the edge
        s = edges{k}(1);
        t = edges{k}(2);

        % Iterate over the configurations of (s,t)
        for i = 1:2
            for j = 1:2
                % Generate possible configurations for the remaining M-2
                % nodes.
                % poss is 2^(M-2)-by-(M-2)
                poss = generatePoss(M-2);

                % Interleave with the configuration of (s,t)
                poss = [poss(:,1:s-1), ...
                        (i-1) * ones(2^(M-2),1), ...
                        poss(:,s:t-2), ...
                        (j-1) * ones(2^(M-2),1), ...
                        poss(:,t-1:end)];
```

```
65
66                  % Compute marginal probability by summing over all possibilities
67                  margPr = 0;
68                  for w = 1:size(poss,1)
69                      logPr = -log(Z);
70                      for e = 1:length(edges)
71                          logPr = logPr + ...
72                                  Theta(poss(w,edges{e}(1))+1, poss(w,edges{e}(2))+1, e);
73                      end
74                      margPr = margPr + exp(logPr);
75                  end
76
77                  % Update the corresponding entry of Theta
78                  Theta(i,j,k) = Theta(i,j,k) + log(Emp(i,j,k)) - log(margPr);
79              end
80          end
81      end
82
83      % Compute log likelihood
84      logOld = logLhood;
85      logLhood = N*sum(sum(sum(Emp.*Theta))) - N*log(Z);
86
87      % Check for convergence
88      if (logLhood - logOld < 1e-14)
89          fprintf('Converges after %d batch iterations!\n', iter);
90          break;
91      elseif (iter == maxIter)
92          fprintf('IPF did not converge: final Δ = %.8f\n', logLhood - logOld);
93      end
94  end
95
96
97  function poss = generatePoss(m)
98  % Generates a (2^m)-by-m array consisting of all possible binary
99  % configurations of m nodes.
100
101 if (m == 1)
102     poss = [0; 1];
103 else
104     possRec = generatePoss(m-1);
105     poss = [zeros(size(possRec,1),1), possRec; ...
106             ones(size(possRec,1),1), possRec];
107 end
```

Code for Problem 5.4

```
1  % This script finds a maximum weight spanning tree from the Pairwise.dat
```

```
% dataset with edge weights given by the mutual information I(\mu_st;
% \mu_s, \mu_t).

load Pairwise.dat
d = size(Pairwise,1);  % number of nodes in the graph
n = size(Pairwise,2);  % number of observations

% Compute empirical vertex marginals \mu_s
mu_s = [(1/n) * sum(Pairwise==0, 2)' ; ...  % X_s = 0
        (1/n) * sum(Pairwise==1, 2)'];      % X_s = 1

% Compute vertex entropy H(\mu_s)
H_s = -sum(mu_s .* log(mu_s))';

% Compute empirical edge marginals \mu_st
mu_st = zeros(2, 2, d*(d-1)/2);
count = 1;  % for ordering the edges from 1 to d*(d-1)/2
for i = 1:d
    for j = i+1:d
        ones_i = real(Pairwise(i,:) == 1);  % when X_i = 1
        ones_j = real(Pairwise(j,:) == 1);  % when X_j = 1
        mu_st(:,:,count) = (1/n) * ...
                           [(1-ones_i)*(1-ones_j)', (1-ones_i)*ones_j'; ...
                                ones_i*(1-ones_j)',     ones_i*ones_j'];
        count = count + 1;
    end
end

% Compute edge entropy H(\mu_st)
H_st = zeros(d*(d-1)/2, 1);
for count = 1:d*(d-1)/2
    mu_edge = mu_st(:,:,count);
    H_st(count) = -sum(sum(mu_edge .* log(mu_edge)));
end

% Compute mutual information I(\mu_st; \mu_s, \mu_t)
I_st = zeros(d*(d-1)/2, 1);
count = 1;  % for ordering the edges from 1 to d*(d-1)/2
for i = 1:d
    for j = i+1:d
        I_st(count) = H_s(i) + H_s(j) - H_st(count);
        fprintf('I(%d,%d) = %.4f\n', i, j, I_st(count));
        count = count + 1;
    end
end
```