CPSC 540 Assignment 3 (due February 27)

Density Estimation and Project Proposal

1 Discrete and Gaussian Variables

1.1 MLE for General Discrete Distribution

Consider a density estimation task, where we have two variables ($d = 2$) that can each take one of k discrete values. For example, we could have

$$X = \begin{bmatrix} 1 & 3 \\ 4 & 2 \\ \text{k} & 3 \\ 1 & \text{k-1} \end{bmatrix}.$$

The likelihood for example x^i under a general discrete distribution would be

$$p(x_1^i, x_2^i|\Theta) = \theta_{x_1^i, x_2^i},$$

where θ_{c_1,c_2} gives the probability of x_1 being in state c_1 and x_2 being in state c_2, for all the k^2 combinations of the two variables. In order for this to define a valid probability, we need all elements θ_{c_1,c_2} to be non-negative and they must sum to one, $\sum_{c_1=1}^{k}\sum_{c_2=1}^{k}\theta_{c_1,c_2} = 1$.

1. Given n training examples, derive the MLE for the k^2 elements of Θ.
2. Because of the sum-to-1 constraint, there are only (k^2-1) degrees of freedom in the discrete distribution, and not k^2. Derive the MLE for this distribution assuming that

$$\theta_{k,k} = 1 - \sum_{c_1=1}^{k}\sum_{c_2=1}^{k}\mathcal{I}[c_1 \neq k, c_2 \neq k]\theta_{c_1,c_2},$$

   so that the distribution only has $(k^2 - 1)$ parameters.
3. If we had separate parameter θ_{c_1} and θ_{c_2} for each variables, a reasonable choice of a prior would be a product of Dirichlet distributions,

$$p(\theta_{c_1}, \theta_{c_2}) \propto \theta_{c_1}^{\alpha_{c_1}-1}\theta_{c_2}^{\alpha_{c_2}-1}.$$

   For the general discrete distribution, a prior encoding the same assumptions would be

$$p(\theta_{c_1,c_2}) \propto \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2}.$$

   Derive the MAP estimate under this prior (assuming we use k^2 variabls to parameterize Θ)

Hint: it is convenient to write the likelihood for an example i in the form

$$p(x^i|\Theta) = \prod_{c\in[k]^2}\theta_c^{\mathcal{I}[x^i=c]},$$

where c is a vector containing (c_1, c_2), $[x^i = c]$ evaluates to 1 if all elements are equal, and $[k]^2$ is all ordered pairs (c_1, c_2). You can use the Lagrangian to enforce the sum-to-1 constraint on the log-likelihood, and you may find it convenient to define $N_c = \sum_{i=1}^n \mathcal{I}[x^i = c]$.

Answers:

1. The log-likelihood is given by

$$-\log p(X|\Theta) = -\sum_{i=1}^{n} \sum_{c \in [k]^2} \mathcal{I}[x^i = c] \log \theta_c + \text{const.}$$

   By re-arranging the sums and ignoring the constant term the Lagrangian is given by

$$L(\Theta, z) = -\sum_{c \in [k]^2} \log \theta_c \sum_{i=1}^{n} \mathcal{I}[x^i = c] + z(\sum_{c \in [k]^2} \theta_c - 1).$$

   We have

$$\nabla_{\theta_c} L(\Theta, z) = -\frac{1}{\theta_c} \sum_{i=1}^{n} \mathcal{I}[x^i = c] + z,$$

   and setting this equal to 0 we get

$$\theta_c = \frac{\sum_{i=1}^n \mathcal{I}[x^i = c]}{z}.$$

   From the sum-to-one constraint we have

$$1 = \sum_{c \in [k]^2} \theta_c = \sum_{c \in [k]^2} \sum_{i=1}^{n} \frac{\mathcal{I}[x^i = c]}{z},$$

   or that

$$z = \sum_{i=1}^{n} \sum_{c \in [k]^2} \mathcal{I}[x^i = c] = \sum_{i=1}^{n} 1 = n.$$

   This gives

$$\theta_c = \frac{N_c}{n},$$

   where N_c is the number of times we have $x^i = (c_1, c_2)$.

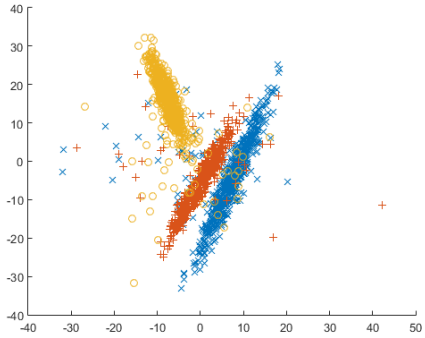
2. The Lagrangian under this constraint is the same, and all the other steps are the same so we get the same MLE.

3. For the MAP estimate we'll replace $\sum_{i=1}^n \mathcal{I}[x^i = c]$ in the Lagrangian by $[\sum_{i=1}^n \mathcal{I}[x^i = c]] + \alpha_{c_1} + \alpha_{c_2} - 2$. Following the steps above we obtain

$$\theta_c = \frac{N_c + \alpha_{c_1} + \alpha_{c_2} - 2}{z} = \frac{N_c + \alpha_{c_1} + \alpha_{c_2} - 2}{n + \sum_{c \in [k]^2} [\alpha_{c_1} + \alpha_{c_2} - 2]}$$

1.2 Generative Classifiers with Gaussian Assumption

Consider the 3-class classification dataset in this image:



In this dataset, we have 2 features and each colour represents one of the classes. Note that the classes are highly-structured: the colours each roughly follow a Gausian distribution plus some noisy samples.

Since we have an idea of what the features look like for each class, we might consider classifying inputs x using a *generative classifier*. In particular, we are going to use Bayes rule to write

$$p(y = c|x, \Theta) = \frac{p(x|y = c, \Theta) \cdot p(y = c|\Theta)}{p(x|\Theta)},$$

where Θ represents the parameters of our model. To classify a new example $\hat{x}$, generative classifiers would use

$$\hat{y} = \underset{y \in \{1,2,\dots,k\}}{\arg\max}\ p(\hat{x}|y = c, \Theta)p(y = c|\Theta),$$

where in our case the total number of classes k is 3 (The denominator $p(\hat{x}|\Theta)$ is irrelevant to the classification since it is the same for all y.) Modeling $p(y = c|\Theta)$ is eays: we can just use a k-state categorical distribution,

$$p(y = c|\Theta) = \theta_c,$$

where θ_c is a single parameter for class c. The maximum likelihood estimate of θ_c is given by n_c/n, the number of times we have $y^i = c$ (which we've called n_c) divided by the total number of data points n.

Modeling $p(x|y = c, \Theta)$ is the hard part: we need to know the *probability of seeing the feature vector x given that we are in class c*. This corresponds to solving a density estimation problem for each of the k possible classes. To make the density estimation problem tractable, we'll assume that the distribution of x given that $y = c$ is given by a $\mathcal{N}(\mu_c, \Sigma_c)$ Gaussian distribution for a class-specific μ_c and Σ_c,

$$p(x|y = c, \Theta) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_c|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu_c)^T\Sigma_c^{-1}(x - \mu_c)\right).$$

Since we are distinguishing between the probability under k different Gaussians to make our classification, this is called *Gaussian discriminant analysis* (GDA). In the special case where we have a constant $\Sigma_c = \Sigma$ across all classes it is known as *linear discriminant analysis* (LDA) since it leads to a linear classifier between any two classes (while the region of space assigned to each class forms a convex polyhedron as in k-means clustering). Another common restriction on the Σ_c is that they are diagonal matrices, since this only requires $O(d)$ parameters instead of $O(d^2)$ (corresponding to assuming that the features are independent univariate Gaussians given the class label). Given a dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^n$, where $x^i \in \mathbb{R}^d$ and $y^i \in \{1, \dots, k\}$, the maximum likelihood estimate (MLE) for the μ_c and Σ_c in the GDA model is the solution to

$$\underset{\mu_1,\mu_2,\dots,\mu_k,\Sigma_1,\Sigma_2,\dots,\Sigma_k}{\arg\max} \prod_{i=1}^{n} p(x^i|y^i, \mu_{y^i}, \Sigma_{y^i}).$$

This means that the negative log-likelihood will be equal to

$$-\log p(X|y,\Theta) = -\sum_{i=1}^{n} \log p(x^i|y^i|\mu_{y^i},\Sigma_{y^i})$$
$$= \sum_{i=1}^{n} \frac{1}{2}(x^i-\mu_{y^i})^T\Sigma_{y^i}^{-1}(x^i-\mu_{y^i}) + \frac{1}{2}\sum_{i=1}^{n}\log|\Sigma_{y^i}| + \text{const.}$$

1. Derive the MLE for the GDA model under the assumption of *common diagonal covariance* matrices, $\Sigma_c = D$ (d parameters). (Each class will have its own mean μ_c.)
2. Derive the MLE for the GDA model under the assumption of *individual scale-identity* matrices, $\Sigma_c = \sigma_c^2 I$ (k parameters).
3. It's painful to derive these from scratch, but you should be able to see a pattern that would allow other common restrictions. Without deriving the result from scratch (hopefully), give the MLE for the case of *individual full* covariance matrices, Σ_c ($O(kd^2)$ parameters).
4. When you run *example_generative* it loads a variant of the dataset in the figure that has 12 features and 10 classes. This data has been split up into a training and test set, and the code fits a k-nearest neighbour classifier to the training set then reports the accuracy on the test data ($\sim 36\%$). The k-nearest neighbour model does poorly here since it doesn't take into account the Gaussian-like structure in feature space for each class label. Write a function *generativeGaussian* that fits a GDA model to this dataset (using individual full covariance matrices). Hand in the function and report the test set accuracy.
5. In this question we would like to replace the Gaussian distribution of the previous problem with the more robust multivariate-t distribution so that it isn't influenced as much by the noisy data. Unlike the previous case, we don't have a closed-form solution for the parameters. However, if you run *example_tdist* it generates random noisy data and fits a multivariate-t model (you will need to add the *minFunc* directory to the Matlab path for the demo to work). By using the *multivariateT* model, write a new function *generativeStudent* that implements a generative model that is based on the multivariate-t distribution instead of the Gaussian distribution. Report the test accuracy with this model.

Hints: you will be able to substantially simplify the notation in parts 1-3 if you use the notation $\sum_{i\in y_c}$ to mean the sum over all values i where $y^i = c$. Similarly, you can use n_c to denote the number of cases where $y_i = c$, so that we have $\sum_{i\in y_c} 1 = n_c$. Note that the determinant of a diagonal matrix is the product of the diagonal entries, and the inverse of a diagonal matrix is a diagonal matrix with the reciprocals of the original matrix along the diagonal. For part three you can use the result from class regarding the MLE of a general multivariate Gaussian. You may find it helpful to use the included *logdet.m* function to compute the log-determinant in more numerically-stable way.

Answer:

1. First let's derive the mean μ_c for class c. If we take the gradient with respect to a particular μ_c and set it to 0 then all terms outside of y_c are zero and we get

$$0 = -\sum_{i\in y_c}\Sigma_c(x^i-\mu_c)$$
$$\Sigma_c^{-1}\mu_c\sum_{i\in y_c}1 = \Sigma_c^{-1}\sum_{i\in y_c}x^i.$$

   Pre-multiplying by Σ_c we get

$$\mu_c = \frac{\sum_{i\in y_c}x^i}{n_c},$$

which (as expected) is the mean of the examples in class c. With Σ_c set to a diagonal D for all classes c, the log-likelihood simplifies to

$$\begin{aligned}
-\log p(X|y,\Theta) + \text{const.} &= \sum_{i=1}^{n} \frac{1}{2}(x^i - \mu_{y^i})^T D^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^{n} \log|D| \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d}(x_j^i - (\mu_{y_i})_j)^2/D_{jj} + \frac{n}{2}\log(\prod_{j=1}^{d} D_{jj}) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d}(x_j^i - (\mu_{y_i})_j)^2/D_{jj} + \frac{n}{2}\sum_{j=1}^{d}\log(D_{jj}) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d}(x_j^i - (\mu_{y_i})_j)^2\Lambda_{jj} - \frac{n}{2}\sum_{j=1}^{d}\log(\Lambda_{jj})
\end{aligned}$$

where we've that the determinant of a diagonal matrix is the product of the diagonal elements, and we've re-parameterized in terms of the inverse Λ. This function is separable in the Λ_{jj} so we can solve it independently for each of the Λ_{jj}. Taking the derivative with respect to Λ_{jj} and equating it with zero we get

$$0 = \sum_{i=1}^{n} \frac{1}{2}(x_j^i - (\mu_{y_i})_j)^2 - \frac{n}{2\Lambda_{jj}}$$

or

$$\frac{1}{\Lambda_{jj}} = \frac{1}{n}\sum_{i=1}^{n}(x_j^i - (\mu_{y_i})_j)^2,$$

which is solved by setting $D_{jj} = \frac{1}{n}\sum_{i=1}^{n}(x_j^i - (\mu_{y_i})_j)^2$, the MLE for the variance along coordinate j around its class-specific mean over all i.

2. If this case we have

$$-\log p(X|y,\Theta) + \text{const.} = \sum_{i=1}^{n} \frac{1}{2\sigma_{y^i}^2}(x^i - \mu_{y^i})^T(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^{n} d\log(\sigma_{y^i}^2)$$

Parameterizing in terms of the squared inverse, we an solve for a particular σ_c using

$$0 = \frac{1}{2}\sum_{i\in y_c}\|x^i - \mu_c\|^2 - \frac{n_c}{\sigma_c^2},$$

implying that $\sigma_c^2 = \frac{1}{n_c d}\sum_{i\in y_c}\|x^i - \mu_c\|^2$, the average variance across variables for examples in the class c.

3. From class we know that the MLE for a Gaussian is $\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x^i - \mu)(x^i - \mu)^T$. Thus if we worked out the general case, we would see that we need to do this for each class:

$$\Sigma_c = \frac{1}{n_c}\sum_{i\in y_c}(x^i - \mu_c)(x^i - \mu_c)^T.$$

4. The Gaussian generative classifiers improves the performance from 0.37 (with KNN) to 0.63. The code should look roughly like this:

```
function [ model ] = generativeGaussian( X, y )

[n,d] = size(X);
k = max(y);

mu = zeros(d,k);
Sigma = zeros(d,d,k);
for c = 1:k
    yc = find(y==c);
    nc(c,1) = length(yc);

    mu(:,c) = (1/nc(c))*sum(X(yc,:))';
    for i = yc'
        % This could be vectorized for speed
        Sigma(:,:,c) = Sigma(:,:,c) + (1/nc(c))*(X(i,:)' - mu(:,c))*(X(i,:)' - mu(:,c))';
    end
end

model.k = k;
model.theta = nc/n;
model.mu = mu;
model.Sigma = Sigma;
model.predict = @predict;
end

function yhat = predict(model, Xhat)
[t,d] = size(Xhat);
k = model.k;
theta = model.theta;
mu = model.mu;
Sigma = model.Sigma;

for i = 1:t
    % This could be vectorized for speed
    % (really just need one Cholesky per class)
    for c = 1:k
        logProb(c) = theta(c) - (1/2)*(Xhat(i,:)'-mu(:,c))'*(Sigma(:,:,c)\(Xhat(i,:)'-mu(:,c))) - (1/2)*logdet(Sigma(:,:,c));
    end
    [~,yhat(i,1)] = max(logProb);

end
end
```

5. This robust generative classifiers improves the performance further up to 0.78.

1.3 Self-Conjugacy for the Mean Parameter

If x is distributed according to a Gaussian with mean μ,

$$x \sim \mathcal{N}(\mu, \sigma^2),$$

and we assume that μ itself is distributed according to a Gaussian

$$\mu \sim \mathcal{N}(\alpha, \gamma^2),$$

then the posterior $\mu|x$ also follows a Gaussian distribution.[1] Derive the form of the (Gaussian) distribution for $p(\mu|x, \alpha, \sigma^2, \gamma^2)$.

Hints: Use Bayes rule and use the $\propto$ sign to get rid of factors that don't depend on μ. You can "complete the square" to make the product look like a Gaussian distribution, e.g. when you have $\exp(ax^2 - bx + \text{const})$ you can factor out an a and add/subtract $(b/2a)^2$ to re-write it as

$$\begin{aligned}\exp\left(ax^2 - bx + const\right) &\propto \exp\left(ax^2 - bx\right) = \exp\left(a(x^2 - (b/a)x)\right) \\ &\propto \exp\left(a(x^2 - (b/a)x + (b/2a)^2)\right) = \exp\left(a(x - (b/2a))^2\right).\end{aligned}$$

Note that multiplying by factors that do not depend on μ within the exponent does not change the distribution. In this question you will want to complete the square to get the distribution on μ, rather than x. You may find it easier to solve thie problem if you parameterize the Gaussians in terms of their 'precision' parameters (e.g., $\lambda = 1/\sigma^2$, $\lambda_0 = 1/\gamma^2$) rather than their variances σ^2 and γ^2.

Answer:

[1] We say that the Gaussian distribution is the 'conjugate prior' for the Gaussian mean parameter (we'll formally discuss conjugate priors later in the course). Another reason the Gaussian distribution is important is that is the only (non-trivial) continuous distribution that has this 'self-conjugacy' property.

From the question description (and keeping in mind that factors in the product that don't depend on μ don't change the distribution),

$$\begin{aligned} p(\mu|x,\alpha,\sigma^2,\gamma^2) &\propto p(x|\mu,\sigma^2)p(\mu|\alpha,\gamma^2) \\ &\propto \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\exp\left(-\frac{(\mu-\alpha)^2}{2\gamma^2}\right) \\ &= \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}-\frac{(\mu-\alpha)^2}{2\gamma^2}\right) \end{aligned}$$

We now re-parameterize in terms of the precisions $\lambda = 1/\sigma^2$ and $\lambda_0 = 1/\gamma^2$,

$$\begin{aligned} p(\mu|x,\alpha,\lambda,\lambda_0) &\propto \exp\left(-\frac{\lambda}{2}(x-\mu)^2-\frac{\lambda_0}{2}(\mu-\alpha)^2\right) \\ &= \exp\left(-\frac{1}{2}\left[\lambda(x^2-2x\mu+\mu^2)+\lambda_0(\mu^2-2\mu\alpha+\alpha^2)\right]\right) \\ &\propto \exp\left(-\frac{1}{2}\left[\lambda(-2x\mu+\mu^2)+\lambda_0(\mu^2-2\mu\alpha)\right]\right) \\ &= \exp\left(-\frac{1}{2}\left[\mu^2(\lambda+\lambda_0)-2\mu(x\lambda+\alpha\lambda_0)\right]\right) \\ &= \exp\left(-\frac{(\lambda+\lambda_0)}{2}\left[\mu^2-2\mu\frac{(x\lambda+\alpha\lambda_0)}{(\lambda+\lambda_0)}\right]\right) \\ &\propto \exp\left(-\frac{(\lambda+\lambda_0)}{2}\left[\mu^2-2\mu\frac{(x+\alpha)}{(\lambda+\lambda_0)}+\frac{(x\lambda+\alpha\lambda_0)^2}{(\lambda+\lambda_0)^2}\right]\right) \\ &= \exp\left(-\frac{(\lambda+\lambda_0)}{2}\left(\mu-\frac{x\lambda+\alpha\lambda_0}{\lambda+\lambda_0}\right)^2\right) \end{aligned}$$

This is proportional to a Gaussian distribution with a mean of $\frac{x\lambda-\alpha\lambda_0}{\lambda+\lambda_0}$ and a precision of $\lambda+\lambda_0$, so we have

$$\mu|x,\alpha,\lambda,\lambda_0 \sim \mathcal{N}\left(\frac{x\lambda+\alpha\lambda_0}{\lambda+\lambda_0},(\lambda+\lambda_0)^{-1}\right).$$

Thus, we add the precision values together to get the final precision, and the final mean is the sum of the data point and the prior mean (weighted by their precisions) divided by the new precision. Equivalently, we could write

$$\mu|x,\alpha,\sigma^2,\gamma^2 \sim \mathcal{N}\left(\frac{\frac{x}{\sigma^2}+\frac{\alpha}{\gamma^2}}{\frac{1}{\sigma^2}+\frac{1}{\gamma^2}},\left(\frac{1}{\sigma^2}+\frac{1}{\gamma^2}\right)^{-1}\right),$$

and $p(\mu|x,\alpha,\sigma^2,\gamma^2)$ is the Gaussian PDF with the above mean and variance.

If you assume that $\sigma^2=\gamma^2=1$, this would simplify to

$$\mu|x,\alpha,\sigma^2,\gamma^2 \sim \mathcal{N}\left(\frac{x+\alpha}{2},\frac{1}{2}\right),$$

2 Mixture Models and Expectation Maximization

2.1 Semi-Supervised Gaussian Discriminant Analysis

Consider fitting a GDA model where some of the y^i values are missing at random. In particular, let's assume we have n labeled examples (x^i, y^i) and then another another t unlabeled examples (x^i). This is a special case of *semi-supervised learning*, and fitting generative models with EM is one of the oldest semi-supervised learning techqniues. When the classes exhibit clear structure in the feature space, it can be very effective even if the number of labeled examples is very small.

1. Derive the EM update for fitting the parameters of a GDA model (with individual full covariance matrices) in the semi-superivsed setting where we have n labeled examples and t unlabeled examples.

2. If you run the demo *example_SSL*, it will load a variant of the dataset from the previous question, but where the number of labeled examples is small and a large number of unlabeled examples are available. The demo first fits a KNN model and then a generative Gaussian model (once you are finished Question 1). Because the number of labeled examples it quite small, the performance is worse than in Question 2. Write a function *generativeGaussianSSL* that fits the generative Gaussian model of the previous question using EM to incorporate the unlabeled data. Hand in the function and report the test error when training on the full dataset.

3. Repeat the previous part, but using the "hard"-EM algorithm where we explicitly classify all the unlabeled examples. How does this change the performance and the number of iterations?

Hint: for the first question most of the work has been done for you in the EM notes on the course webpage. You can use the result (**) and the update of θ_c from those notes, but you will need to work out the update of the parameters of the Gaussian distribution $p(x^i|y^i, \Theta)$.

Hint: for the second question, although EM often leads to simple updates, implementing them correctly can often be a pain. One way to help debug your code is to compute the observed-data log-likelihood after every iteration. If this number goes down, then you know your implementation has a problem. You can also test your updates of sets of variables in this way too. For example, if you hold the μ_c and Σ_c fixed and only update the θ_c, then the log-liklihood should not go down. In this way, you can test each of combinationso of updates on their to make sure they are correct.

Answer:

1. The update for the categorical distribution of the y^i is given by

$$\theta_c^{t+1} = \frac{n_c + \sum_{i=1}^t r_c^i}{n+t},$$

but they don't need to explicitly write that because it's in the EM notes. The update for the Gaussian mean parameters is given by

$$\mu_c = \frac{\sum_{i \in y_c} x^i + \sum_{i=1}^t r_c^i \tilde{x}^i}{n_c + \sum_{i=1}^t r_c^i},$$

and for the covariance matrices is given by

$$\Sigma_c = \frac{\sum_{i \in y_c} (x^i - \mu_c)(x^i - \mu_c)^T + \sum_{i=1}^t r_c^i (\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T}{n_c + \sum_{i=1}^t r_c^i}.$$

2. Here is one possible implementation of the method:

```
% EM iterations
for iter = 1:50
    r = zeros(t,k);
    for i = 1:t
        for c = 1:k
            r(i,c) = theta(c)*mvnpdf(Xtilde(i,:),mu(:,c)',Sigma(:,:,c));
        end
        r(i,:) = r(i,:)/sum(r(i,:));
    end


    for c = 1:k
        yc = find(y==c);
        nc(c,1) = length(yc);

        theta(c) = (nc(c) + sum(r(:,c)))/(n+t);
    end



    mu = zeros(d,k);
    z = zeros(k,1);
    for i = 1:n
        c = y(i);
        mu(:,c) = mu(:,c) + X(i,:)';
        z(c) = z(c) + 1;
    end
    for i = 1:t
        for c = 1:k
            mu(:,c) = mu(:,c) + r(i,c)*Xtilde(i,:)';
            z(c) = z(c) + r(i,c);
        end
    end
    for c = 1:k
        mu(:,c) = mu(:,c)/z(c);
    end


    Sigma = zeros(d,d,k);
    z = zeros(k,1);
    for i = 1:n
        c = y(i);
        Sigma(:,:,c) = Sigma(:,:,c) + (X(i,:)' - mu(:,c))*(X(i,:)' - mu(:,c))';
        z(c) = z(c) + 1;
    end
    for i = 1:t
        for c = 1:k
            Sigma(:,:,c) = Sigma(:,:,c) + r(i,c)*(Xtilde(i,:)' - mu(:,c))*(Xtilde(i,:)' - mu(:,c))';
            z(c) = z(c) + r(i,c);
        end
    end
    for c = 1:k
       Sigma(:,:,c) = Sigma(:,:,c)/z(c);
    end
```

The test accuracy with this approximation increases 0.53 (which was worse than if had all the labels) to 0.75 which non-intuitively is actually *better than if we had all the true labels*. The reason that EM does better is probably that the true labels are very noisy so by integrating over their values the Gaussians aren't chasing as much of this noise.

3. You can modify the EM code to do "hard"-EM by just changing the responsibility variables r so that they are one for the most likely class and zero for the others. I found that this gave the same accuracy (0.75) but converged in fewer iterations.

2.2 Mixture of Bernoullis

The function *example_Bernoulli* loads a binarized version of the MNIST dataset and fits a density model that uses an independent Bernoulli to model each feature. If the reports the average NLL on the test data and shows 4 samples generated from the model. Unfortunately, the test NLL is infinity and the samples look terrible.

1. To address the problem that the average NLL is infinity, modify the *densityBernoulli* function to implement Laplace smoothing based on an extra argument α. Hand in the code and report the average

NLL with $\alpha = 1$.

2. Write a new function implementing the mixture of Bernoullis model with Laplace smoothing of the θ values (note that Laplace smoothing only change the M-step). Hand in the code and report the average NLL with $\alpha = 1$ and $k = 10$ for a particular run of the algorithm, as well as 4 samples from the model and 4 of the cluster images.

Answers:

1. The code could just needs a single-line change:
theta = (alpha + sum(X))/(2*alpha + n); The average NLL is 205.8471.

2. The main loop of the code could look like this:

```
for iter = 1:maxIter

    % Compute responsibilities
    for c = 1:k
        r(:,c) = pi(c)*prod(repmat(theta(c,:),[n 1]).^X.*repmat(1-theta(c,:),[n 1]).^(1-X),2);
    end
    for i = 1:n
        r(i,:) = r(i,:)/sum(r(i,:));
    end

    % Update parameters
    Nc = sum(r);
    for c = 1:k
        theta(c,:) = (alpha + sum(diag(sparse(r(:,c)))*X))/(2*alpha + Nc(c));
    end
    pi = Nc/n;

    % Evaluate marginal NLL
    model.pi = pi;
    model.theta = theta;
    nll = sum(predict(model,X))/n

    if abs(nll-nll_old) < .01
        break;
    end
    nll_old = nll;
end
```

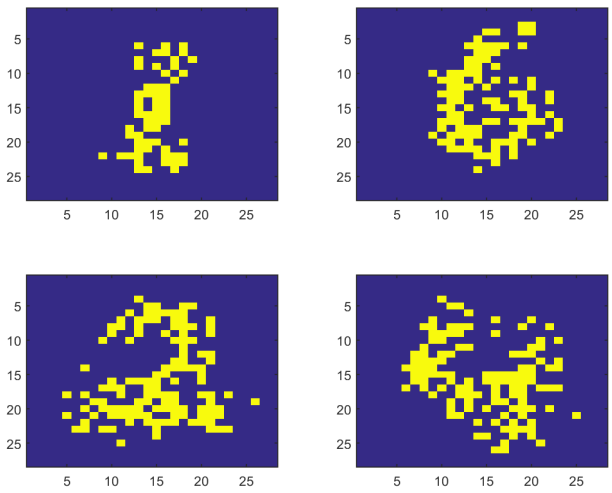
The predict function could look like this:

```
function nlls = predict(model, Xhat)
[t,d] = size(Xhat);
pi = model.pi;
theta = model.theta;
k = length(pi);

lik = zeros(t,1);
for c = 1:k
    lik = lik + pi(c)*prod(repmat(theta(c,:),[t 1]).^Xhat.*repmat(1-theta(c,:),[t 1]).^(1-Xhat),2);
end
nlls = -log(lik);
end
```

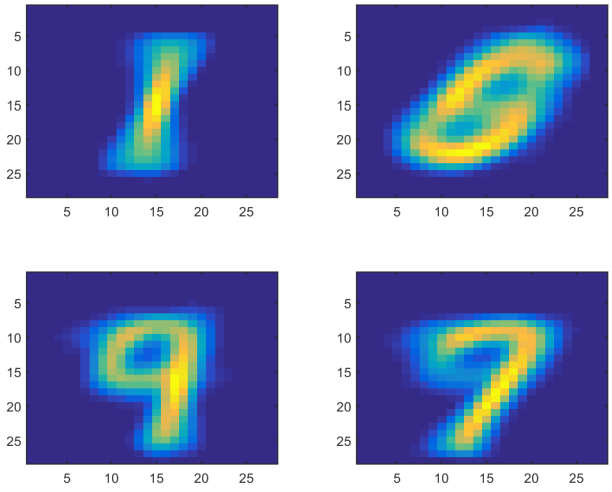
On some systems this leads to underflow, and you could instead compute the responsibilities using

```
% Compute responsibilities
for c = 1:k
    logR(:,c) = log(pi(c)) + sum(X.*repmat(log(theta(c,:)),[n 1]) + (1-X).*repmat(log(1-theta(c,:)),[n 1]),2);
end

for i = 1:n
    logR(i,:) = logR(i,:) - mylogsumexp(logR(i,:));
end
r = exp(logR);
```

The exact NLL will depend on the initialization and the number of iterations, but a number like 165 seems reasonable. Below are examples of what the samples might look like:



These look more like images but the independent assumption is still problematic. Here is what the cluster means might look like:



Depending on how you initialize the thetas, you may also get empty clusters.

3 Project Proposal

For the final of this assignment, you must a submit a project proposal for your course project. The proposal should be a maximum of 2 pages (and 1 page or half of a page is ok if you can describe your plan concisely). The proposal should be written for me and the TAs, so you don't need to introduce any ML background but you will need to introduce non-ML topics. The projects must be done is groups of 2-3. If you are doing your assignment in a group that is different from your project group, only 1 group member should include the proposal as part of their submission (we'll do the merge across assignments, and this means that assignments could have multiple proposals). Please state clearly who is involved with each project proposal.

There is quite a bit of flexibility in terms of the type of project you do, as I believe there are many ways that people can make valuable contributions to research. However, note that ultimately the project will have three parts:

1. A very shot paper review summarizing the pros and cons of a particular paper on the topic (due with Assignment 4).
2. A short literature review summarizing at least 10 papers on a particular topic (due with Assignment 5).
3. A final report containing at most 6 pages of text (the actual document can be longer due to figures, tables, references, and proofs) that emphasizes a particular "contribution" (i.e., what doing the project has added to the world).

The reason for this, even though it's strange for some possible projects, is that this is the standard way that results are communicated to the research community.

The three mains ingredients of the project proposal are:

1. What problem you are focusing on.
2. What you plan to do.
3. What will be the "contribution".

Also, note that for the course project that negative results (i.e., we tried something that we thought we would work in a particular setting but it didn't work) are acceptable.

Here are some standard project "templates" that you might want to follow:

- **Application bake-off**: you pick a specific application (from your research, personal interests, or maybe from Kaggle) or a small number of related applications, and try out a bunch of techniques (e.g., random forests vs. logistic regression vs. generative models). In this case, the contribution would be showing that some methods work better than others for this specific application (or your contribution could be that everything works equally well/badly).
- **New application**: you pick an application where ML methods have previously applied, and you test out whether ML methods are effective for the task. In this case, the contribution would be knowing whether ML is suitable for the task.
- **Scaling up**: you pick a specific machine learning technique, and you try to figure out how to make it run faster or on larger datasets (for example, how do we apply kernel methods when n is very large). In this case, the contribution would be the new technique and an evaluation of its performance.
- **Improving performance**: you pick a specific machine learning technique, and try to extend it in some way to improve its performance (for example, how can we efficiently use non-linearity within graphical models). In this case, the contribution would be the new technique and an evaluation of its performance.

- **Generalization to new setting**: you pick a specific machine learning technique, and try to extend it to a new setting (for example, making a graphical-model version of random forests). In this case, the contribution would be the new technique and an evaluation of its performance.
- **Perspective paper**: you pick a specific topic in ML, read a larger number of papers on the topic, then write a report summarizing what has been done on the topic and what are the most promising directions of future work. In this case, the contribution would be your summary of the relationships between the existing works, and your insights about where the field is going.
- **Coding project**: you pick a specific method or set of methods (like independent component analysis), and build an implementation of them. In this case, the contribution would be the implementation itself.
- **Theory**: you pick a theoretical topic (like the variance of cross-validation), read what has been done about it, and try to prove a new result (usually by relaxing existing assumptions or adding new assumptions). The contribution could be a new analysis of an existing method, or why some approaches to analyzing the method will not work.

The above are just suggestions, and many projects will mix several of these templates together, but if you are having trouble getting going then it's best to stick with one of the above templates. Also note that the above includes topics not covered in the course (like random forests), so there is flexibility in the topic, but the topic should be closely-related to ML.

This question is mandatory but will not be formally marked: it's just a sanity check that you have at least one project idea that fits within the scope of 540 course project, and it's an excuse for you to allocate some time to thinking about the project. Also, there is flexibility in the choice of project topics even after the proposal: if you want to explore different topics you can ultimately choose to do a project that is unrelated to the one in your proposal/paper-review/literature-review, although it will likely be easier to do all 4 parts on the same topic.