CPSC 540 Assignment 3 (due February 21 at midnight)

The assignment instructions are the same as for the previous assignment.

1. Name(s): Cody Griffith, Ziming Yin, Tim Jaschek
2. Student ID(s): Cody: 88416169 , Ziming: 88489166 , Tim: 91220160.

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

$$\begin{bmatrix} 1 & 3 \\ 4 & 2 \\ k & 3 \\ 1 & k-1 \end{bmatrix}$$

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.

   Answer: We assume our underlying random variables to be independent. Therefore the likelihood and the negative log-likelihood are given by

$$p(X|\Theta) = \prod_{i=1}^{n}\prod_{c\in[k^2]}\theta_c^{\mathcal{I}(x^i=c)}$$

$$L(\Theta,\lambda) = -\log p(X|\Theta) + \lambda(\sum_{c\in[k^2]}\theta_c - 1)$$

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

where we used a Lagrangian multiplier to enforce $\sum \theta_c = 1$. As we wan to optimize the negative log-likelihood let us take the partial derivatives and force them to be zero.

$$\nabla L(\Theta, \lambda) = -\frac{1}{\theta_c} \sum_{i=1}^{n} \mathcal{I}[x^i = c] + \lambda = 0$$
$$\theta_c = \frac{\sum_{i=1}^{n} \mathcal{I}(x^i = c)}{\lambda}.$$

Using that $\sum_{c\in[k^2]} \theta_c = 1$ this yields

$$1 = \frac{\sum_{c\in[k^2]} \sum_{i=1}^{n} \mathcal{I}(x^i = c)}{\lambda}$$
$$\lambda = \sum_{c\in[k^2]} \sum_{i=1}^{n} \mathcal{I}(x^i = c) = n$$

Hence,

$$\theta_c = \frac{\sum_{i=1}^{n} \mathcal{I}(x^i = c)}{n} = \frac{N_c}{n}.$$

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.

Answer:

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

is the same as

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

Thus we can still use same Lagrangian as in 1, and end with

$$\theta_c = \frac{\sum_{i=1}^{n} \mathcal{I}(x^i = c)}{n} = \frac{N_c}{n}$$

But rewrite it into $k^2 - 1$ parameters:

$$\theta_c[c \neq k^2] = \frac{\sum_{i=1}^n \mathcal{I}(x^i = c)}{n} = \frac{N_c}{n}$$

$$\theta_{k^2} = \frac{\sum_{i=1}^n \mathcal{I}(x^i = k^2)}{n} = \frac{n - \sum_{c=1}^{k^2-1} N_c}{n}$$

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 variables 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 $N_c = \sum_{i=1}^n \mathcal{I}[x^i = c]$.

Answer: By Bayes formula

$$p(\Theta|X) = \prod_{i=1}^n \frac{p(x^i|\Theta)p(\Theta)}{p(x^i)}.$$

As the denominator does not depend on Θ it can be omitted in our MAP estimation.

$$\arg\max_{\Theta} p(\Theta|X) = \arg\max_{\Theta} \prod_{i=1}^n \prod_{c \in [k^2]} \theta_c^{\mathcal{I}(x^i=c)+(\alpha c_1+\alpha c_2-2)}$$

$$L(\Theta, \lambda) = -\log p(X|\Theta) + \lambda(\sum_{c \in [k^2]} \theta_c - 1) + \text{const.}$$

$$= -\sum_{i=1}^n \sum_{c \in [k^2]} (\mathcal{I}(x^i = c) + (\alpha c_1 + \alpha c_2 - 2)) \log \theta_c$$

$$+ \lambda(\sum_{c \in [k^2]} \theta_c - 1),$$

where we used once again a Lagrangian multiplier to enforce $\sum \theta_c = 1$. Optimizing this expression with basic calculus yieds

$$\begin{aligned}
\nabla L(\Theta, \lambda) &= -\frac{1}{\theta_c} \sum_{i=1}^{n} (\mathcal{I}[x^i = c] + (\alpha c_1 + \alpha c_2 - 2)) + \lambda \\
\nabla L(\Theta, \lambda) &= 0 \\
\theta_c &= \frac{\sum_{i=1}^{n} \mathcal{I}(x^i = c) + (\alpha c_1 + \alpha c_2 - 2)}{\lambda} \\
1 = \sum_{c \in [k^2]} \theta_c &= \frac{\sum_{c \in [k^2]} \sum_{i=1}^{n} \mathcal{I}(x^i = c) + (\alpha c_1 + \alpha c_2 - 2)}{\lambda} \\
\lambda &= n + \sum_{c \in [k^2]} (\alpha c_1 + \alpha c_2 - 2) \\
\theta_c &= \frac{N_c + (\alpha c_1 + \alpha c_2 - 2)}{n + \sum_{c \in [k^2]} (\alpha c_1 + \alpha c_2 - 2)}.
\end{aligned}$$

1.2 Generative Classifiers with Gaussian Assumption

Consider the 3-class classification dataset in this image:
** Can't put the imagine, no source **
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

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

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

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

where in our case the total number of classes k is 3.[1] Modeling $p(y^i = c|\Theta)$ is easy: we can just use a k-state categorical distribution,

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

[1]The denominator $p(\tilde{x}^i|\Theta)$ is irrelevant to the classification since it is the same for all y.

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 $p(x^i|y^i = c, \Theta)$ is the hard part: we need to know the *probability of seeing the feature vector x^i 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^i given that $y^i = c$ is given by a $\mathcal{N}(\mu_c, \Sigma_c)$ Gaussian distribution for a class-specific μ_c and Σ_c,

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

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 and softmax classification). 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 $y^i \in \{1, \ldots, k\}$, the maximum likelihood estimate (MLE) for the μ_c and Σ_c in the GDA model is the solution to

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

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

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

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.)

   Answer: First calculate MLE for μ_c.

$$\begin{aligned} f &= \sum_{i=1}^n \frac{1}{2}(x^i - \mu_{y^i})^T\Sigma_{y^i}^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^n \log|\Sigma_{y^i}| \\ \nabla f_\mu &= \sum_{i=1}^n \Sigma^{-1}(x^i - \mu_{y^i}) = 0 \end{aligned}$$

$$\sum_{i=1}^{n}(x^i - \mu_{y^i}) = 0$$
$$\sum_{i\in y_c}(x^i - \mu_c) = 0$$
$$\sum_{i\in y_c} x^i - n_c\mu_c = 0$$
$$\mu_c = \frac{\sum_{i\in y_c} x^i}{n_c}$$

Second look at Σ.

$$\begin{aligned}
f &= \sum_{i=1}^{n}\frac{1}{2}(x^i - \mu_{y^i})^T\Sigma_{y^i}^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^{n}\log|\Sigma_{y^i}| \\
&= \sum_{i=1}^{n}\frac{1}{2}(x^i - \mu_{y^i})^T\Sigma^{-1}(x^i - \mu_{y^i}) + \frac{n}{2}\log|\Sigma| \\
&= \sum_{i=1}^{n}\frac{1}{2}(x^i - \mu_{y^i})^T\Theta(x^i - \mu_{y^i}) + \frac{n}{2}\log|\Theta^{-1}| \\
&= \sum_{i=1}^{n}\frac{1}{2}(x^i - \mu_{y^i})^T\Theta(x^i - \mu_{y^i}) - \frac{n}{2}\log|\Theta| \\
&= \sum_{i=1}^{n}\sum_{j=1}^{d}\frac{1}{2}(x^i_j - (\mu_{y^i})_j)^2\Theta_{jj} - \frac{n}{2}\log|\prod_{j=1}^{d}\Theta_{jj}| \\
&= \sum_{i=1}^{n}\sum_{j=1}^{d}\frac{1}{2}(x^i_j - (\mu_{y^i})_j)^2\Theta_{jj} - \frac{n}{2}\sum_{j=1}^{d}\log\Theta_{jj} \\
\nabla f_{\Theta_{jj}} &= \sum_{i=1}^{n}\frac{1}{2}(x^i_j - (\mu_{y^i})_j)^2 - \frac{n}{2}\frac{1}{\Theta_{jj}} = 0
\end{aligned}$$

$$\Sigma_{jj} = \frac{1}{n}\sum_{i=1}^{n}(x^i_j - (u_{y^i})_j))^2$$

2. Derive the MLE for the GDA model under the assumption of *individual scale-identity* matrices, $\Sigma_c = \sigma_c^2 I$ (k parameters).

   Answer: First calculate MLE for μ_c.

$$f = \sum_{i=1}^{n}\frac{1}{2}(x^i - \mu_{y^i})^T\Sigma_{y^i}^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^{n}\log|\Sigma_{y^i}|$$
$$\nabla f_{\mu} = \sum_{i=1}^{n}\Sigma_{y^i}^{-1}(x^i - \mu_{y^i}) = 0$$

$$= \frac{1}{\sigma_c^2} \sum_{i=1}^{n} I(x^i - \mu_{y^i}) = 0$$

$$\sum_{i=1}^{n} (x^i - \mu_{y^i}) = 0$$

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

$$\sum_{i \in y_c} x^i - n_c \mu_c = 0$$

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

Second look at Σ.

$$\begin{aligned}
f &= \sum_{i=1}^{n} \frac{1}{2} (x^i - \mu_{y^i})^T \Sigma_{y^i}^{-1} (x^i - \mu_{y^i}) + \frac{1}{2} \sum_{i=1}^{n} \log |\Sigma_{y^i}| \\
&= \sum_{i=1}^{n} \frac{1}{2} (x^i - \mu_{y^i})^T (\sigma_{y^i}^2 I)^{-1} (x^i - \mu_{y^i}) + \frac{1}{2} \sum_{i=1}^{n} \log |\sigma_{y^i}^2 I| \\
&= \sum_{i=1}^{n} \frac{1}{2} (x^i - \mu_{y^i})^T \sigma_{y^i}^{-2} (x^i - \mu_{y^i}) + \frac{1}{2} \sum_{i=1}^{n} \log \sigma_{y^i}^{2d} \\
&= \sum_{i=1}^{n} \frac{1}{2} (x^i - \mu_{y^i})^T \sigma_{y^i}^{-2} (x^i - \mu_{y^i}) + \frac{d}{2} \sum_{i=1}^{n} \log \sigma_{y^i}^2 \\
\alpha_{y^i} &= \sigma_{y^i}^{-1} \\
f &= \sum_{i=1}^{n} \frac{1}{2} (x^i - \mu_{y^i})^T \alpha_{y^i}^2 (x^i - \mu_{y^i}) - \frac{d}{2} \sum_{i=1}^{n} \log \alpha_{y^i}^2 \\
&= \sum_{i=1}^{n} \alpha_{y^i}^2 \sum_{j=1}^{d} \frac{1}{2} (x_j^i - (\mu_{y^i})_j)^2 - \frac{d}{2} \sum_{i=1}^{n} \log \alpha_{y^i}^2 \\
2\nabla f_{\alpha_{y^i}^2} &= \sum_{i=1}^{n} \sum_{j=1}^{d} (x_j^i - (\mu_{y^i})_j)^2 - d \sum_{i=1}^{n} \frac{1}{\alpha_{y^i}^2} \\
&= 0 \\
0 &= \sum_{i=1}^{n} (\sum_{j=1}^{d} (x_j^i - (\mu_{y^i})_j)^2 - d \frac{1}{\alpha_{y^i}^2}) \\
0 &= \sum_{i \in y_c}^{n} (\sum_{j=1}^{d} (x_j^i - (\mu_c)_j)^2 - d \frac{1}{\alpha_c^2})
\end{aligned}$$

$$\sum_{i \in y_c}^{n} \sum_{j=1}^{d} (x_j^i - (\mu_c)_j)^2 = d\frac{n_c}{\alpha_c^2}$$

$$\alpha_c^2 = \frac{dn_c}{\sum_{i \in y_c}^{n} \sum_{j=1}^{d} (x_j^i - (\mu_c)_j)^2}$$

$$\sigma_c^2 = \frac{\sum_{i \in y_c}^{n} \sum_{j=1}^{d} (x_j^i - (\mu_c)_j)^2}{dn_c}$$

$$\Sigma_c = \sigma_c^2 I$$

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).

   Answer:

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

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

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 (around $\sim 63\%$ test error). 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 *gda* that fits a GDA model to this dataset (using individual full covariance matrices). Hand in the function and report the test set accuracy.

   Answer: Test Error: 0.367

```
include("misc.jl") # Includes mode function and GenericModel typedef

function gda_predict(Xhat,X,y)
  (n,d) = size(X)
  (t,d) = size(Xhat)
  k = maximum(y)

  sigma = zeros(d,d,k)
  mu = zeros(d,k)
  nc = zeros(1,k)
  for i in 1:k
      yc = find(y.==i)
      nc[i]=length(yc)
      mu[:,i] = sum(X[yc,:],1)/nc[i]
      for j in yc
          sigma[:,:,i] = (X[j,:]-mu[:,i])*(X[j,:]-mu[:,i])'+sigma[:,:,i]
      end
      sigma[:,:,i] = sigma[:,:,i]/nc[i]
  end

  yhat = zeros(t)
  for w in 1:t
      argmax = zeros(1,k)
      for t in 1:k
          argmax[t] = nc[t]/n  -0.5*
          ( logdet(sigma[:,:,t]) ) -0.5*
          (  (Xhat[w,:]-mu[:,t])'*  (sigma[:,:,t])^(-1) * (Xhat[w,:]-mu[:,t]))
      end
      yhat[w]= indmax(argmax)
  end

  return yhat
end

function gda(X,y)
# Implementation of k-nearest neighbour classifier
  predict(Xhat) = gda_predict(Xhat,X,y)
  return GenericModel(predict)
end
```

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_student* it generates random noisy data and fits a multivariate-t model. By using the *studentT* model, write a new function *tda* 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.

   Answer: Test Error: 0.198

```
include("misc.jl")
include("findMin.jl")

using SpecialFunctions

function tda_predict(Xhat,X,y)
  (n,d) = size(X)
  (t,d) = size(Xhat)
  k = maximum(y)
  yhat = zeros(t)

  sigma = zeros(d,d,k)
  mu = zeros(d,k)
  dof = 3*ones(1,k)
  nc = zeros(1,k)
  logZ = zeros(1,k)

  for i in 1:k
      yc = find(y.==i)
      nc[i]=length(yc)
      xc = X[yc,:]
     (mu[:,i], dof[i], sigma[:,:,i],logZ[i]) = studentT(xc)
  end

  prob = zeros(t,k)
  for cls in 1:k
      pdf_cls = PDF(Xhat,mu[:,cls],dof[cls],sigma[:,:,cls],logZ[cls])
      prob[:,cls] = nc[cls]/n * pdf_cls
  end

    for w in 1:t
        yhat[w]= indmax(prob[w,:])
    end

  return yhat
end

function tda(X,y)
  predict(Xhat) = tda_predict(Xhat,X,y)
  return GenericModel(predict)
end
```

```
function studentT(X)
    (n,d) = size(X)
    # Initialize parameters
    mu = zeros(d)
    Sigma = eye(d)
    dof = 3*ones(1,1)
    # Optimize by block coordinate optimization
    mu_old = ones(d)
    funObj_mu(mu) = NLL(X,mu,Sigma,dof,1)
    funObj_Sigma(Sigma) = NLL(X,mu,Sigma,dof,2)
    funObj_dof(dof) = NLL(X,mu,Sigma,dof,3)
    while norm(mu-mu_old,Inf) > .1
    #for i in 1:5
        mu_old = mu
        # Update mean
        mu = findMin(funObj_mu,mu,verbose=false)
        # Update covariance
        Sigma[:] = findMin(funObj_Sigma,Sigma[:],verbose=false)
        # Update degrees of freedom
        dof = findMin(funObj_dof,dof,verbose=false)
    end

    # Compute square root of log-determinant
    SigmaInv = Sigma^-1
    A = chol(Sigma)'
    logStd = sum(log.(diag(A)))
    dof = dof[1] # Take scalar out of containiner
    logZ = lgamma((dof+d)/2) - (d/2)*log(pi) - logStd - lgamma(dof/2) - (d/2)*log(d
    return (mu, dof, SigmaInv,logZ)
end


function PDF(Xhat,mu,dof,SigmaInv,logZ)
    (t,d) = size(Xhat)
    PDFs = zeros(t)

    for i in 1:t
        xCentered = Xhat[i,:] - mu
        tmp = 1 + (1/dof)*dot(xCentered,SigmaInv*xCentered)
        nll = ((d+dof)/2)*log(tmp) - logZ
        PDFs[i] = exp(-nll)
    end
```

```
        return PDFs

    end
```

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. At test time for GDA, you may find it more numerically reasonable to compare log probabilities rather than probabilities of different classes, and you may find it helpful to use the *logdet* function to compute the log-determinant in a more numerically-stable way than taking the log of the determinant. (Also, don't forget to center at training and test time.)

For the last question, you can define an empty array that can be filled with k *DensityModel* objects using:

```
  subModel = Array{DensityModel}(k)
```

1.3 Self-Conjugacy for the Mean Parameter

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

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

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

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

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

Hints: Use Bayes rule and use the $\propto$ sign to get rid of factors that don't depend on μ. You can then "complete the square" to make the product look like a Gaussian distribution. In particular, 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

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

[2] 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.

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^i. 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.

A direct application of Bayes' rule gives, (note we define $\lambda = 1/\sigma^2$ and $\lambda_0 = 1/\gamma^2$),

$$\begin{aligned} p(\mu|x^i, \alpha, \sigma^2, \gamma^2) =& \frac{p(x^i|\mu, \sigma^2)p(\mu|\alpha, \gamma^2)}{p(x^i|\alpha, \sigma^2, \gamma^2)}, \\ \propto& p(x^i|\mu, \sigma^2)p(\mu|\alpha, \gamma^2), \\ \propto& e^{-\lambda(x^i-\mu)^2/2} e^{-\lambda_0(\mu-\alpha)^2/2}. \end{aligned}$$

Here we were able to ignore any terms that didn't contain μ directly by calling them constants of proportionality. We now complete the square of the exponential,

$$\begin{aligned} -\frac{\lambda}{2}(x^i-\mu)^{-}\frac{\lambda_0}{2}(\mu-\alpha)^2 =& -\frac{1}{2}(\lambda+\lambda_0)\left[\mu^2 - 2\frac{\lambda x^i + \lambda_0\alpha}{\lambda+\lambda_0}\mu + C_1\right], \\ =& -\frac{\lambda+\lambda_0}{2}\left[\mu - \frac{\lambda x^i + \lambda_0\alpha}{\lambda+\lambda_0}\right]^2 + C_2 \end{aligned}$$

Where we find some constants that are independent of μ. Thus,

$$\begin{aligned} p(\mu|x^i, \alpha, \sigma^2, \gamma^2) \propto& \exp\left(-\frac{\lambda+\lambda_0}{2}\left[\mu - \frac{\lambda x^i + \lambda_0\alpha}{\lambda+\lambda_0}\right]^2\right), \\ \sim& \mathcal{N}\left(\frac{\lambda x^i + \lambda_0\alpha}{\lambda+\lambda_0}, (\lambda+\lambda_0)^{-1}\right). \end{aligned}$$

Thus, the posterior distribution of μ is also normally distributed with this mean and this standard deviation, thus it is self-conjugate.

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 a set of n labeled examples (x^i, y^i) and another set of 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.

The GDA is a mixture model which means that the probability of x^i is given by

$$p(x^i) = \sum_{c=1}^{k} p(x^i, y^i = c) = \sum_{c=1}^{k} p(y^i = c)p(x^i|y^i = c) = \sum_{c=1}^{k} \pi_c p(x^i|z^i = c),$$

where $p(x^i|y^i = c)$ follow a multivariate Gaussian distribution, i.e.

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

We have observed variables $O = \{X, y, \tilde{X}\}$ and hidden variables $H = \{\tilde{y}\}$ and unknown model parameters π_c, μ_c and Σ_c. To find the optimal model parameters we would like to perform a maximum likelihood estimate. However, this is not possible, as not all $\tilde{y}$ are given. The expectation maximization algorithm helps out. We start with an initial guess for the model parameters, and then perform a maximum likelihood estimate where we use the expected values for the $\tilde{y}$ with respect to the distribution under the initial guess. This will give us a new, more accurate set of parameters and we continue iteratively. The EM algorithm which has the form:

$$\begin{aligned} \Theta^0 &= \left((\pi_c^0)_{c\in\{1,\ldots,k\}}, (\mu_c^0)_{c\in\{1,\ldots,k\}}, (\Sigma_c^0)_{c\in\{1,\ldots,k\}}\right) && \text{(initial guess)} \\ \Theta^{l+1} &= \arg\max_{\Theta} Q(\Theta|\Theta^l), && l \in \mathbb{N}, \end{aligned}$$

where

$$\begin{aligned} Q(\Theta|\Theta^l) &= \mathbb{E}_{H|O,\Theta^l}[\log p(O, H|\Theta)] = \sum_H p(H|O, \Theta^l) \log p(O, H|\Theta) \\ &= \sum_{\tilde{y}} p(\tilde{y}|X, y, \tilde{X}, \Theta^l) \log p(X, y, \tilde{X}, \tilde{y}|\Theta) \\ &= \sum_{i=1}^{n} \log p(y^i, x^i|\Theta) + \sum_{i=1}^{t} \sum_{\tilde{y}^i\in\{1,2,\ldots,k\}} \underbrace{p(\tilde{y}^i|\tilde{x}^i, \Theta^l)}_{:=r^i_{\tilde{y}^i}} \log p(\tilde{y}^i, \tilde{x}^i|\Theta). \end{aligned}$$

The last inequality is proved in the additional notes to EM on the course webpage. We can go one step further by using that $p(\tilde{x}^i, \tilde{y}^i|\Theta) = p(\tilde{y}^i|(\pi_c)_c)p(\tilde{x}^i|\tilde{y}^i, \Theta)$ (Product rule for joint probabilities). This yields

$$\begin{aligned} Q(\Theta|\Theta^k) &= \sum_{i=1}^{n} \log p(y^i|(\pi_c)_c) + \sum_{i=1}^{n} \log p(x^i|y^i, \Theta) \\ &\quad + \sum_{i=1}^{t}\sum_{\tilde{y}^i=1}^{k} r^i_{\tilde{y}^i} \log p(\tilde{y}^i|(\pi_c)_c) + \sum_{i=1}^{t}\sum_{\tilde{y}^i=1}^{k} r^i_{\tilde{y}^i} \log p(\tilde{x}^i|\tilde{y}^i, \Theta). \end{aligned}$$

Let us first find the optimal weights $(\pi_c)_c$. The likelihood of y is given by

$$p(y|\pi) = \prod_{i=1}^{n}\prod_{c=1}^{k} p(y^i = c|\pi)^{\mathcal{I}[y^i=c]} \prod_{j=1}^{t}\prod_{c'=1}^{k} p(\tilde{y}^j = c'|\pi)^{\mathcal{I}[y^j=c']}.$$

As we do not know the values of $\tilde{y}^i$ we consider the expected negative log-likelihood with a Lagrangian multiplier λ to enforce $\sum_c \pi_c = 1$:

$$\begin{aligned}\mathbb{E}_{\tilde{X},\Theta^l}[NLL](\pi) &= -\sum_{\substack{1\leqslant i\leqslant n\\ 1\leqslant c\leqslant k}} \mathcal{I}[y^i = c]\log(\pi_c) - \sum_{\substack{1\leqslant j\leqslant t\\ 1\leqslant c'\leqslant k}} p(\tilde{y}^j = c'|\tilde{x}^j,\Theta^l)\log(\pi_{c'}) + \lambda\left(\sum_{c=1}^{k}\pi_c - 1\right)\\ &= -\sum_{i=1}^{n}\sum_{c=1}^{k} n_c\log(\pi_c) - \sum_{j=1}^{t}\sum_{\tilde{y}^j\in\{1,\ldots,k\}} r^j_{\tilde{y}^j}\log(\pi_{c'}) + \lambda\left(\sum_{c=1}^{k}\pi_c - 1\right).\end{aligned}$$

Differentiating and forcing the derivative to be zero yields:

$$\begin{aligned}\nabla_{\pi_c}\mathbb{E}_{\tilde{X},\Theta^l}[NLL](\pi) = -\frac{n_c}{\pi_c} - \frac{\sum_{j=1}^{t} r^j_c}{\pi_c} + \lambda = 0\\ \Rightarrow \qquad \pi_c = \frac{n_c + \sum_{j=1}^{t} r^j_c}{\lambda}.\end{aligned}$$

Summing over all $c = 1,\ldots,k$ gives

$$\lambda = \sum_{c=1}^{k} n_c + \sum_{c=1}^{k}\sum_{j=1}^{t} r^j_c = n + t,$$

and hence

$$\pi_c = \frac{n_c + \sum_{j=1}^{t} r^j_c}{n+t}.$$

Let us get back to optimizing Q. As we perform a Gaussian discriminant analysis we can express the probabilities in the formula for $Q(\Theta,\Theta^l)$ in terms of the model parameters

$$\log p(x^i|y^i,(\mu_c)_c,(\Sigma_c)_c) = \frac{1}{2}(x^i-\mu_{y^i})^T\Sigma^{-1}_{y^i}(x^i-\mu_{y^i}) + \frac{1}{2}\log|\Sigma_{y^i}| + \text{const}.$$

Therefore

$$\begin{aligned}Q(\Theta|\Theta^k) = \sum_{i=1}^{n}\log p(y^i|(\pi_c)_c) + \sum_{i=1}^{t}\sum_{\tilde{y}^i=1}^{k} r^i_{\tilde{y}^i}\log p(\tilde{y}^i|(\pi_c)_c)\\ + \frac{1}{2}\sum_{i=1}^{n}\Big((x^i-\mu_{y^i})^T\Sigma^{-1}_{y^i}(x^i-\mu_{y^i}) + \log|\Sigma_{y^i}|\Big)\end{aligned}$$

$$+ \frac{1}{2} \sum_{i=1}^{t} \sum_{\tilde{y}^i=1}^{k} r^i_{\tilde{y}^i} \left((\tilde{x}^i - \mu_{\tilde{y}^i})^T \Sigma^{-1}_{\tilde{y}^i} (\tilde{x}^i - \mu_{\tilde{y}^i}) + \log |\Sigma_{\tilde{y}^i}| \right) + \text{const.}$$

Differentiating with respect to μ_c yields

$$\nabla_{\mu_c} Q(\Theta|\Theta^l) = \sum_{i \in y_c} \Sigma_c^{-1}(x^i - \mu_c) + \sum_{i=1}^{t} r^i_c \Sigma_c^{-1}(\tilde{x}^i - \mu_c)$$

Forcing the derivative to be zero gives us

$$\begin{aligned} \sum_{i \in y_c} \Sigma_c^{-1}(x^i - \mu_c) &= - \sum_{i=1}^{t} r^i_c \Sigma_c^{-1}(\tilde{x}^i - \mu_c) \\ \sum_{i \in y_c} x^i &= n_c \mu_c - \sum_{i=1}^{t} r^i_c \tilde{x}^i - \sum_{i=1}^{t} r^i_c \mu_c \\ \mu_c &= \frac{\sum_{i \in y_c} x^i + \sum_{i=1}^{t} r^i_c \tilde{x}^i}{n_c + \sum_{i=1}^{t} r^i_c}. \end{aligned}$$

The last parameters to determin are $(\Sigma_c)_c$. Again, by taking the derivative

$$\nabla_{\Sigma_c} Q(\Theta|\Theta^l) = \frac{1}{2} \sum_{i \in y_c} \left((x^i - \mu_c)(x^i - \mu_c)^T + \Sigma_c^{-1} \right) + \frac{1}{2} \sum_{i=1}^{t} r^i_c (\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T + r^i_c \Sigma_c^{-1}.$$

Setting this equal to zero leaves us with

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

Hence the expectation maximization update is given by:

$$\begin{cases} (r^i_c)^{l+1} & = p(\tilde{y}^i = c | \tilde{x}^i, \Theta^l) = \frac{p(\tilde{x}^i | \tilde{y}^i = c, \Theta^l) p(\tilde{y}^i = c | \Theta^l)}{\sum_{c'=1}^{k} p(\tilde{x}^i | \tilde{y}^i = c', \Theta^l) p(\tilde{y}^i = c' | \Theta^l)} \\ \pi_c^{l+1} & = \frac{n_c + \sum_{j=1}^{t} r^j_c}{n+t} \\ \mu_c^{l+1} & = \frac{\sum_{i \in y_c} x^i + \sum_{i=1}^{t} r^i_c \tilde{x}^i}{n_c + \sum_{i=1}^{t} r^i_c} \\ \Sigma_c^{l+1} & = \frac{\sum_{i \in y_c} (x^i - \mu_c)(x^i - \mu_c)^T + \sum_{i=1}^{t} r^i_c (\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T}{n_c + \sum_{i=1}^{t} r^i_c}. \end{cases}$$

2. If you run the demo *example_SSL.jl*, 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 and assuming that you put your *gda* function in a file named *gda.jl*). 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.
Using an EM approach with 10 iterations, the test error is 0.254.
example_SSL.jl:

```
# Load X and y variable
using JLD
data = load("SSL.jld")
(X,y,Xtest,ytest,Xbar) = (data["X"],data["y"],data["Xtest"],data["ytest"],data["Xba

# Fit a KNN classifier
k = 5
include("knn.jl")
model = knn(X,y,k)

# Evaluate training error
yhat = model.predict(X)
trainError = mean(yhat .!= y)
@printf("Train Error with %d-nearest neighbours: %.3f\n",k,trainError)

# Evaluate test error
yhat = model.predict(Xtest)
testError = mean(yhat .!= ytest)
@printf("Test Error with %d-nearest neighbours: %.3f\n",k,testError)

# Fit GDA model
include("gda.jl")
model = gda(X,y)

# Evaluate training error
yhat = model.predict(X)
trainError = mean(yhat .!= y)
@printf("Train Error with GDA: %.3f\n",trainError)

# Evaluate test error
yhat = model.predict(Xtest)
testError = mean(yhat .!= ytest)
@printf("Test Error with GDA: %.3f\n",testError)

# Fit generative gaussian SSL
include("generativeGaussianSSL.jl")
```

```
model = generativeGaussianSSL(X,y,Xbar)

# Evaluate training error
yhat = model.predict(X)
trainError = mean(yhat .!= y)
@printf("\nTrain Error with generative Gaussian SSL: %.3f\n",trainError)

# Evaluate test error
yhat = model.predict(Xtest)
testError = mean(yhat .!= ytest)
@printf("\nTest Error with generative Gaussian SSL: %.3f\n",testError)
```

generativeGaussianSSL.jl:

```
include("misc.jl") # Includes mode function and GenericModel typedef

function generativeGaussianSSL_predict(Xhat,X,y,Xbar)
  # set number of iterations of EM
  maxIter = 10

  #get dimensions
  (n,d)   = size(X)
  (n2,d2) = size(Xbar)
  (t,d)   = size(Xhat)
  k       = maximum(y)

  # initialize parameters for multivariate gaussian distribution using GDA
  sigma = zeros(d,d,k)
  mu = zeros(d,k)
  mix = zeros(1,k)
  (mix,mu,sigma) = gda_parameters(X,y)

  #run expectation maximization
  for i in 1:maxIter
      (mix, mu, sigma) = EM(mix,mu,sigma,X,y,Xbar,i)
  end

  #make prediction
  yhat = zeros(t)
  for w in 1:t
      argmax = zeros(1,k)
      for j in 1:k
          detlog = logdet(sigma[:,:,j])
```

```
            vec = Xhat[w,:]-mu[:,j]
            argmax[j] = mix[j] - 0.5*detlog -0.5*vec'*(sigma[:,:,j])^(-1)*vec
        end
        yhat[w]= indmax(argmax)
    end
    return yhat
end

function generativeGaussianSSL(X,y,Xbar)
    predict(Xhat) = generativeGaussianSSL_predict(Xhat,X,y,Xbar)
    return GenericModel(predict)
end

function gda_parameters(X,y)
    # same as gda but we just care for the parameters
    (n,d) = size(X)
    k = maximum(y)
    sigma = zeros(d,d,k)
    mu = zeros(d,k)
    nc = zeros(1,k)
    mix = zeros(1,k)
    for i in 1:k
        yc = find(y.==i)
        nc[i]=length(yc)
        mu[:,i] = sum(X[yc,:],1)/nc[i]
        for j in yc
              vec = X[j,:]-mu[:,i]
              sigma[:,:,i] += vec*vec'
        end
        sigma[:,:,i] = sigma[:,:,i]./nc[i]
    end
    mix = nc/n
    return mix,mu,sigma
end

function EM(mix,mu,sigma,X,y,Xbar,iter)
      (n,d) = size(X)
      (t,d) = size(Xbar)
      k = maximum(y)

      # initialize new parameters
      new_sig = zeros(d,d,k)
    new_mu = zeros(d,k)
    new_mix = zeros(1,k)
    r = zeros(t,k)
    nc = zeros(1,k)

    # perform the EM update as introduced in problem 2.1

    # E-step
    @printf("E-step iteration %d *** ",iter)
    print("computing the responsibilities...\n")
    for i in 1:t
        denom = 0
        for c in 1:k
            r[i,c] = mix[c]*gaussianPDF(Xbar[i,:],mu[:,c],sigma[:,:,c])
            denom += r[i,c]
        end
        r[i,:] = r[i,:]/denom
    end

    # M-step
    @printf("M-Step iteration %d *** ",iter)
    print("updating parameters... ")
    for j in 1:k
        yc = find(y.==j)
        nc = length(yc)
        total = nc + sum(r[:,j])
        new_mix[j] = total/(n+t)
        new_mu[:,j] = (sum(X[yc,:],1) + sum(r[:,j].*Xbar,1))./total
        for m in yc
            vec = X[m,:]-mu[:,j]
            new_sig[:,:,j] += vec*vec'
        end
        for m in 1:t
            vec = Xbar[m,:]-mu[:,j]
            new_sig[:,:,j] += r[m,j].*vec*vec'
        end
        new_sig[:,:,j] = new_sig[:,:,j]/total
    end
    print("return parameters...\n")
    return new_mix,new_mu,new_sig
end
```

```
function gaussianPDF(xvec,mu,sigma)
    d = length(xvec)
    y = xvec-mu
    p = -d/2*log(2*pi)-0.5*logdet(sigma) -0.5*y'*sigma^(-1)*y
    return exp(p)
end
```

3. Repeat the previous part, but using the imputation approach ("hard"-EM) where we explicitly classify all the unlabeled examples before each model update. How does this change the performance and the number of iterations?
Using the imputation approach, we can get similar performance with half the number of iterations: test error 0.263 with 5 iterations.
example_SSL.jl:

```
# Fit generative gaussian SSL with imputation
model = generativeGaussianSSLinpute(X,y,Xbar)

# Evaluate training error
yhat = model.predict(X)
trainError = mean(yhat .!= y)
@printf("\nTrain Error with generative Gaussian SSL using imputation: %.3f\n",trainE

# Evaluate test error
yhat = model.predict(Xtest)
testError = mean(yhat .!= ytest)
@printf("\nTest Error with generative Gaussian SSL using imputation: %.3f\n",testEr
```

generativeGaussianSSL.jl:

```
include("misc.jl") # Includes mode function and GenericModel typedef

function generativeGaussianSSLimpute_predict(Xhat,X,y,Xbar)
  # set number of iterations of EM
  maxIter = 2

  #get dimensions
  (n,d) = size(X)
  (n2,d2) = size(Xbar)
  (t,d) = size(Xhat)
  k = maximum(y)

  # initialize parameters for multivariate gaussian distribution using GDA
  sigma = zeros(d,d,k)
  mu = zeros(d,k)
  mix = zeros(1,k)
  (mix,mu,sigma) = gda_parameters(X,y)
  ybar = zeros(n2,1)

  # run expectation maximization with imputation
  for i in 1:maxIter
   # here we use imputation to create more labelled examples
    probs = zeros(1,k)
    for ii in  1:n2
        for j in 1:k
            probs[j] = gaussianPDF(Xbar[ii,:],mu[:,j],sigma[:,:,j])
        end
        ybar[ii] = indmax(probs)
    end
    ybar = trunc.(Int,ybar)
    (mix, mu, sigma) = EM_inpute(mix,mu,sigma,X,y,Xbar,ybar,i)
  end

  #make prediction
  yhat = zeros(t)
  for w in 1:t
      argmax = zeros(1,k)
      for j in 1:k
          detlog = logdet(sigma[:,:,j])
          vec = Xhat[w,:]-mu[:,j]
          argmax[j] = mix[j] - 0.5*detlog -0.5*vec'*(sigma[:,:,j])^(-1)*vec
      end
      yhat[w]= indmax(argmax)
  end
  return yhat
end

function generativeGaussianSSLinpute(X,y,Xbar)
  predict(Xhat) = generativeGaussianSSLimpute_predict(Xhat,X,y,Xbar)
  return GenericModel(predict)
end


function EM_inpute(mix,mu,sigma,X,y,Xbar,ybar,iter)
    new_X = [X;Xbar]
    new_y = [y;ybar]
    (n,d) = size(new_X)
    k = maximum(new_y)

    # initialize new parameters
    new_sig = zeros(d,d,k)
    new_mu = zeros(d,k)
    new_mix = zeros(1,k)

    # perform the EM update as introduced in problem 2.1

    # M-step
    @printf("M-Step iteration %d *** ",iter)
    print("updating parameters... ")
    for j in 1:k
        yc = find(new_y.==j)
        nc = length(yc)
        new_mix[j] = nc/n
        new_mu[:,j] = sum(new_X[yc,:],1)/nc
        for m in yc
            vec = new_X[m,:]-mu[:,j]
            new_sig[:,:,j] += vec*vec'
        end
        new_sig[:,:,j] = new_sig[:,:,j]/nc
    end
    print("return parameters...\n")
    return new_mix,new_mu,new_sig
end
```

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 $p(x^i|y^i, \Theta)$.

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 combinations of updates on their own to make sure they are correct.

2.2 Poisson Mixture Model

Consider a density estimation with examples $x^i \in \{\mathbb{Z}_{\geq 0}\}^d$ representing *counts* of d variables. In this setting, a natural way to model an individual variable x^i_j would be with a Poisson distribution,[3]

$$p(x^i_j = \alpha \mid \lambda_j) = \frac{\lambda_j^\alpha e^{-\lambda_j}}{\alpha!}.$$

However, if we assume this structure for each variable then the variables would be independent. One way to model dependent count variables would be with a mixture of independent Poisson distributions,

$$p(x^i|\Theta) = \sum_{c=1}^k \theta_c \prod_{j=1}^d p(x^i_j \mid \lambda_{jc}),$$

where Θ contains all the θ_c and λ_{jc} values. Derive the EM update for this.

Hint: most of the work has been done for you in the EM notes on the course webpage.

To start, we must find our Q function. Following the guidelines from the course EM notes,

$$\begin{aligned}
p(x^i|\Theta) =& \sum_{c=1}^k \theta_c \prod_{j=1}^d p(x^i_j|\lambda_{jc}),\\
=& \sum_{c=1}^k p(z^i = c|\Theta)p(x^i|z^i,\Theta),\\
=& \mathbb{E}_{z^i|x^i,\Theta}(p(x^i|z^i,\Theta)).
\end{aligned}$$

We define our Q function to be,

$$\begin{aligned}
Q(\Theta|\Theta^l) =& \mathbb{E}_{Z|X,\Theta^l}(\log p(X,Z|\Theta)),\\
=& \sum_Z p(Z|X,\Theta^l)\log(p(X,Z|\Theta)),\\
=& \sum_{i=1}^n \sum_Z p(Z|x^i,\Theta^l)\log(p(x^i,Z|\Theta)),\\
=& \sum_{i=1}^n \sum_{z^i=1}^k \prod_{j=1}^n p(z^j|x^i,\Theta^l)\left[\log(p(x^i|z^i,\Theta)) + \log(p(z^i|\Theta))\right], \qquad (*)\\
=& \sum_{i=1}^n \sum_{z^i=1}^k p(z^i|x^i,\Theta^l)\log(p(x^i|z^i,\Theta)) + \sum_{i=1}^n \sum_{z^i=1}^k p(z^i|x^i,\Theta^l)\log(p(z^i|\Theta)),\\
=& \sum_{i=1}^n \sum_{z^i=1}^k (r^i_{z^i})^l \log(p(x^i|z^i,\Theta)) + \sum_{i=1}^n \sum_{z^i=1}^k (r^i_{z^i})^l \log(p(z^i|\Theta)),
\end{aligned}$$

[3]Note that the Poisson distribution makes certain assumptions about count data, which may not always be true, so in practice you may want to think about other possible likelihoods.

$$
\begin{aligned}
&= \sum_{i=1}^{n} \sum_{z^i=1}^{k} \sum_{j=1}^{d} (r_{z^i}^i)^l \log(p(x_j^i|\lambda_{jz^i})) + \sum_{i=1}^{n} \sum_{z^i=1}^{k} (r_{z^i}^i)^l \log(\theta_{z^i}), \\
&= \sum_{i=1}^{n} \sum_{z^i=1}^{k} \sum_{j=1}^{d} (r_{z^i}^i)^l \left[x_j^i \log(\lambda_{jc}) - \lambda_{jc} - \log(x_j^i!)\right] + \sum_{i=1}^{n} \sum_{z^i=1}^{k} (r_{z^i}^i)^l \log(\theta_{z^i}).
\end{aligned}
$$

Where in (*) we used that each product would sum to one. We now have something that is differentiable in both parameters of interest. To find the MLE for θ_c we must impose a restriction that this is actually a probability, $\sum \theta_c = 1$, and we can do this with a typical Lagrangian approach,

$$
\begin{aligned}
\Lambda(\Theta, \xi) =& Q(\Theta|\Theta^l) + \xi \left(\sum_{c=1}^{k} \theta_c - 1 \right), \\
\nabla_{\theta_c} \Lambda =& \frac{1}{\theta_c} \sum_{i=1}^{n} (r_c^i)^l + \xi = 0, \\
\nabla_{\xi} \Lambda =& \sum_{c=1}^{k} \theta_c - 1 = 0.
\end{aligned}
$$

Solving this system leads to,

$$
\theta_c = \frac{1}{n} \sum_{i=1}^{n} (r_c^i)^l
$$

For λ_{jc} we can just differentiate directly as there are no other conditions imposed,

$$
\begin{aligned}
\nabla_{\lambda_{jc}} Q =& \sum_{i=1}^{n} (r_c^i) l \left(\frac{x_j^i}{\lambda_{jc}} - 1 \right) = 0, \\
\lambda_{jc} =& \frac{\sum_{i=1}^{n} (r_c^i)^l x_j^i}{\sum_{i=1}^{n} (r_c^i)^l} = \frac{1}{n\theta_c} \sum_{i=1}^{n} (r_c^i)^l x_j^i.
\end{aligned}
$$

Thus our update rule is,

$$
\begin{cases}
(r_c^i)^{l+1} & = p(\tilde{y}^i = c|\tilde{x}^i, \Theta^l) = \frac{p(\tilde{y}^i=c, \tilde{x}^i|\Theta^l)}{\sum_{c'=1}^{k} p(\tilde{y}^i=c', \tilde{x}^i|\Theta^l)}, \\
\theta_c^{l+1} & = \frac{1}{n} \sum_{i=1}^{n} (r_c^i)^l, \\
\lambda_{jc}^{l+1} & = \frac{\sum_{i=1}^{n} (r_c^i)^l x_j^i}{n\theta_c^l}.
\end{cases}
$$

3 Very-Short Answer Questions

Give a short and concise 1-sentence answer to the below questions.

1. Which of the following models cannot be kernelized?
   (a) K-nearest neighbours. Yes
   (b) L2-regularized least squares. Yes
   (c) L1-regularized least squares. No
   (d) L2-regularized logistic regression. No
   (e) PCA. Yes
2. How can we use density estimation for supervised learning?
   We can use density estimation for to label data for supervised techniques. Or we can find outliers in data.
3. For categorical variables, which prior leads to Laplace smoothing as the MAP estimation?
   The beta distribution leads to Laplace smoothing as a prior for $\alpha = \beta = 2$.
4. What is an advantage and a disadvantage of density estimation using a product of independent distributions?
   The assumption of having independent distributions is that it is very cheap to estimate each feature but can get expensive for general discrete problems and has limitations for what distributions can be modeled.
5. Describe a one-dimensional dataset where fitting a Gaussian distribution to the data would be innappropriate.
   Uniform data, or any data set where its okay to see data far from mean (heavy-tailed).
6. In the graphical LASSO, what do edges in the graph correspond to?
   The edges are the values of the parameters, they represent the dependence on one another.
7. List the 4 operations we discussed that give Gaussian distributions by transforming existing Gaussian variables/distributions.
   (a) Affine transformation: If $p(x)$ is Gaussian then $p(Ax + b)$ is.
   (b) Marginalization: If $p(x, z)$ is Gaussian, then $p(x)$ is.
   (c) Conditioning: If $p(x, z)$ is Gaussian, then $p(x|z)$ is.
   (d) Product: If $p(x)$ and $p(z)$ are Gaussian, then $p(x)p(z)$ is proportionally Gaussian.
8. Why do the π_c need to be a convex combination in mixture models?
   Because under a convex combination, the probability property is preserved (i.e summing to one).
9. What is the difference between "missing at random" and "missing completely at random"?

MCAR is what it seems like, just random data missing from the data set. But MAR is missing data that has some kind of conditional dependence on another variable.

10. In what setting does it make sense to use the EM algorithm?
    Any type of problem that could have complete data or allows optimizing over MAR data.

11. What is an advantage and a disadvantage of using a homogeneous Markov chain instead of an inhomogeneous Markov chain?
    Pro: run fast, and easy to train parameters, need less data. Con: lose position information.

4 Project Proposal

For the final part 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 in 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 (and often unavoidable).

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 where people aren’t using ML, 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, or could be a comparison of different ways to address the problem.
- **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, or could be a comparison of different ways to address the problem.
- **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 could be the implementation itself or a comparison of different ways to solve the problem.

- **Theory**: you pick a theoretical topic (like the variance of cross-validation or the convergence of proximal stochastic gradient in the non-convex setting), 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.

Project Proposal

How to use Machine Learning Techniques in Combating Crime

Remark: Our group is interested in the following possible projects:

- How to use machine learning techniques in combating crime
- Combining machine learning and signal processing
- Proof (or counterexample) of the convergence of proximal stochastic gradient in the non-convex setting

We are not entirely decided yet but we tend to work on the first option.

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.

Serial criminal pattern detection

Intro

There is a project called Murder Accountability Project (MAP). Ordinary murders can result from lovers triangles, gang fights, robberies, or brawls. Each year, about five thousand people kill someone and don't get caught, and a percentage of these men and women have undoubtedly killed more than once. They usually exhibit patterns in their operations, by specializing in a particular crime category, and applying a specific method for implementing their crimes. And thus extract the pattern behind then do a classification and prediction is important.

Research so far

Right now there are a lot of paper working on this topic. From what we have learned so far, there tend to be two main categories: one is classify the criminal pattern, doing a clustering activity in the area that is concerned with detecting trends in the data by classifying and grouping similar records; The other is predict next possible crime location. The methods used including Bayesian learning theory, k-means clustering, semi-supervised learning, neural network ect.
Crime Pattern Detection Using Data Mining.
some discussion about serial killer algorithm
Classification system for serial criminal patterns
A Novel Serial Crime Prediction Model Based on Bayesian Learning Theory

Natural Bayesian Killers

What we plan to do
Doing the prediction followed by classification, which we believe will increase the accuracy of prediction greatly since we operate new case under similar cases' data. And also try out a bunch of techniques that might work in our case. Pick out the one that works best. For now We like to apply several algorithms that we learned in class on crime data. In particular the EM-algorithm from this assignment is of interest as we have a huge variety of examples and features given and just a small subset of these has assigned a label (the criminal or the gang). Density estimation techniques might be of interest as well, to generate predictions of the number of crimes in a certain area and to prevent this with an increase of police patrols or a more detailed monitoring.
Another possibility is to get in contact with a member of the predictive policing unit here in Vancouver (Ryan Prox) and see what components of the project are currently implementing ML techniques and compare/contrast or improve current techniques. There was a recent colloquium held and here is the abstract: PIMS - BC Data Colloquium: Ryan Prox
The Vancouver Police Department is known worldwide for its pioneering work in the field of intelligence-led policing, and is the first police service in Canada to deploy a machine-learning predictive system directly on police mobile computers. With a focus on high-end analytics, combined with competitive technology, the Department has achieved stunning results in reducing crime rates. In the first quarter of 2016, the City of Vancouver encountered the highest number of residential burglaries ever recorded. Following the implementation of predictive technology and the consequent deployment of resources based on the forecasting, this trend reversed and the following quarter saw the lowest residential burglaries ever on record. What set this deployment apart from previous attempts at predictive policing is the way the technology was used. Innovative policing practices, combined with an advanced evaluative methodology that leveraged ArcGIS Desktop model testing and Geocortex technology, validated the system and outcomes under real-world conditions where the most successful strategies were advanced, and less successful approaches were abandoned. This learning process resulted in a highly effective crime prevention strategy that reversed a record increase in burglaries for the City.