CPSC 540 Assignment 2 (due February 12 at midnight)

The assignment instructions are the same as for the previous assignment, but for this assignment you can work in groups of 1-3. However, please only hand in one assignment for the group.

1. Name(s): Ricky Ma, Patricia Ye
2. Student ID(s): 82943424, 18139162

1 Convexity

1.1 From Definitions

Show that the following functions are convex, by only using one of the definitions of convexity. In other words, don't use the "operations that preserve convexity" or other convexity results stated in class, but instead show that the function follows from one of the definitions of convexity we gave in class:[1]

1. Upper bound on second-order Taylor expansion: $f(w) = f(v) + \nabla f(v)^T(w-v) + \frac{L}{2}\|w-v\|^2$, for $L > 0$, which is often used in the analysis of gradient descent.

$$\nabla f(w) = \nabla f(v)^T + \frac{L}{2}(2w - 2v)$$

$$\nabla^2 f(w) = LI$$

Since $L > 0$, we know both $\frac{L}{2}\|w-v\|^2$ and $\nabla^2 f(w)$ are always positive. Therefore, $f(w)$ is always above its tangent plane, meaning $f(w)$ is convex.

2. Maximum absolute value: $f(w) = \max_{j\in\{1,2,\ldots,d\}} |w_j|$. Answer: The maximum absolute value of a vector is the equivalent to the ∞-norm of a vector. $f(w) = max_{j\in\{1,2,\ldots d\}}|w_j| = \|w\|_\infty$. All norms are convex since f is always below the "chord".

$$\begin{aligned} f(\theta w + (1-\theta)v) &= \|\theta w + (1-\theta)v\|_\infty \\ &\leq \|\theta w\|_\infty + \|(1-\theta)v\|_\infty \\ &= |\theta|\|w\|_\infty + |(1-\theta)|\|v\|_\infty \\ &= \theta\|w\|_\infty + (1-\theta)\|v\|_\infty \\ &= \theta f(w) + (1-\theta)f(v) \end{aligned}$$

3. The exponential loss, $f(w) = \sum_{i=1}^n \exp(-y^i w^\top x^i)$, used by AdaBoost.

$$\nabla f(w) = -X^T y \sum_{i=1}^n exp(-y^i w^T x^i)$$

$$\nabla^2 f(w) = yX^T X y^T \sum_{i=1}^n exp(-y^i w^T x^i)$$

Since $yX^TXy^T \succeq 0$ and $\sum_{i=1}^n exp(-y^i w^T x^i) \geq 0$ for all w, $\nabla^2 f(w) \succeq 0$ and $f(w)$ is convex.

1.2 Using Operations that Preserve Convexity

Show that the following functions are convex (you can use results from class and operations that preserve convexity if they help):

4. Support vector regression: $f(w) = \sum_{i=1}^N \max\{0, |w^\top x_i - y_i| - \epsilon\} + \frac{\lambda}{2}\|w\|_2^2$.
   - $|w^\top x_i - y_i|$ is convex, since it is a linear function of w
   - both 0 and ϵ are constants, meaning both 0 and $|w^\top x_i - y_i| - \epsilon$ are convex
   - $\max\{0, |w^\top x_i - y_i| - \epsilon\}$ is convex, since the max of a convex function is convex

[1] That C^0 convex functions are below their chords, that C^1 convex functions are above their tangents, or that C^2 convex functions have a positive semidefinite Hessian.

- $\sum_{i=1}^{N} \max\{0, |w^\top x_i - y_i| - \epsilon\}$ is convex, since the sum of convex functions is convex
- $\frac{\lambda}{2}\|w\|_2^2$ is convex, since it is a squared 2-norm
- Therefore, $f(w)$ is convex.

5. Mixed-norm regularization: $f(w) = \|w\|_{p,q} = \left(\sum_{g\in\mathcal{G}} \|w_g\|_q^p\right)^{\frac{1}{p}}$ (for $p \geq 1$ and $q \geq 1$, and $\mathcal{G}$ partitioning the variables into a set of "groups"). This is a generalization of L1-regularization that encourages sparsity in groups of variables (setting entire groups to 0). Hint: show that this is a norm. Answer: Each $\|w_g\|_q$ is a norm of a vector of size $|g|$ and then by the definition of a norm, $\|x\|_p = \left(\sum_i^d x_i^p\right)^{\frac{1}{p}}$, so $f(w)$ is taking the norm of a vector of size $|\mathcal{G}|$ where the terms are $\|w_g\|_q, g \in \mathcal{G}$. All norms are convex since f is always below the "chord".

$$
\begin{aligned}
f(\theta w + (1-\theta)v) = \|\theta w + (1-\theta)v\|_{p,q} &= \left(\sum_{g\in\mathcal{G}} \|\theta w_g + (1-\theta)v_g\|_q^p\right)^{\frac{1}{p}} \\
&\leq \left(\sum_{g\in\mathcal{G}} \|\theta w_g\|_q^p + \|(1-\theta)v_g\|_q^p\right)^{\frac{1}{p}} = \left(\sum_{g\in\mathcal{G}} \theta^p \|w_g\|_q^p + (1-\theta)^p \|v_g\|_q^p\right)^{\frac{1}{p}} \\
&\leq \left(\sum_{g\in\mathcal{G}} \theta^p \|w_g\|_q^p\right)^{\frac{1}{p}} + \left(\sum_{g\in\mathcal{G}} (1-\theta)^p \|v_g\|_q^p\right)^{\frac{1}{p}} \\
&= \theta \left(\sum_{g\in\mathcal{G}} \|w_g\|_q^p\right)^{\frac{1}{p}} + (1-\theta)\left(\sum_{g\in\mathcal{G}} \|v_g\|_q^p\right)^{\frac{1}{p}} \\
&= \theta f(w) + (1-\theta) f(v)
\end{aligned}
$$

6. Negative minimum of entropy over pairs of variables: $f(w) = -\min_{\{i,j \mid i\neq j\}}\{-w_i \log w_i - w_j \log w_j\}$ subject to $w_i > 0$ for all i. Answer:

$$
\begin{aligned}
f(w) &= -min_{\{i,j \mid i\}}\{-w_i \log w_i - w_j \log w_j\} \\
&= max_{\{i,j \mid i\neq j\}}\{w_i \log w_i + w_j \log w_j\} \\
\text{let } g(w) &= w \log w \\
g'(w) &= 1 + \log w \\
g''(w) &= \frac{1}{w} \text{ since } w_i > 0 \text{ for all } i,
\end{aligned}
$$

$g''(w) > 0$ for all w_i and is convex.

Sum of two convex functions is convex, so $w_i \log w_i + w_j \log w_j$ is convex.
Maximum of a convex function preserves convexity, so $max_{\{i,j \mid i\}}\{w_i \log w_i + w_j \log w_j\}$ is convex.
$max_{\{i,j \mid i\neq j\}}\{w_i \log w_i + w_j \log w_j\} = -min_{\{i,j \mid i\}}\{-w_i \log w_i - w_j \log w_j\} = f(w)$ so $f(w)$ is convex.

1.3 Strong Convexity

Which of the functions 1-6 in the previous subsections are strongly convex? 1 and 6 are strongly convex.

2 Discrete and Gaussian Variables

2.1 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

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

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.

$$p(X|\Theta) = \prod_{i=1}^{n} \theta_{x_1^i, x_2^i}$$

$$l = \log p(X|\Theta) = \sum_{i=1}^{n} \log(\theta_{x_1^i, x_2^i}) = \sum_{c_1=1}^{k} \sum_{c_2=1}^{k} n_{c_1,c_2} \log(\theta_{c_1,c_2})$$

Since $\sum_{c_1=1}^{k} \sum_{c_2=1}^{k} \theta_{c_1,c_2} = 1$, we can rewrite the likelihood as $\sum_{c_1=1}^{k} \sum_{c_2=1}^{k} n_{c_1,c_2} \log(\theta_{c_1,c_2})$, where n_{c_1,c_2} represents the number of examples x^i where $x_1^i = c_1$ and $x_2^i = c_2$.

$$\frac{\delta l}{\delta \theta_{c_1,c_2}} = \frac{n_{c_1,c_2}}{\theta_{c_1,c_2}} - \frac{n - n_{c_1,c_2}}{1 - \theta_{c_1,c_2}} = 0$$

Solving for θ_{c_1,c_2}, $\hat{\theta}_{c_1,c_2} = \frac{n_{c_1,c_2}}{n}$

2. Note that this question was updated on February 4th. If we had separate parameters θ_{1,c_1} and θ_{2,c_2} for each variable, a standard for the prior would be a product of independent Dirichlet distributions,

$$p(\Theta) \propto \left[\prod_{c_1} \theta_{1,c_1}^{\alpha_{c_1}-1}\right] \left[\prod_{c_2} \theta_{c_2}^{\alpha_{c_2}-1}\right],$$

where in this case the two variables share k hyper-parameters $\alpha_1, \alpha_2, \ldots, \alpha_k$. Alternately, we could use a Dirichlet distribution over the k^2 variables,

$$p(\Theta) \propto \prod_{c_1=1}^{k} \prod_{c_2=1}^{k} \theta_{c_1,c_2}^{\alpha_{c_1,c_2}-1},$$

with k^2 hyper-parameters. An alternate approach that could be considered is to use

$$p(\Theta) \propto \prod_{c_1=1}^{k} \prod_{c_2=1}^{k} \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2},$$

which only uses k hyper-parameters to define a prior over the k^2 values. Derive the MAP estimate under this prior (assuming we use k^2 variables to parameterize Θ) Answer:

$$\begin{aligned}
p(\Theta \mid X) &= \frac{p(X \mid \Theta)p(\Theta)}{p(X)} \\
&\propto p(X \mid \Theta)p(\Theta) \\
&\propto \prod_{i=1}^{n} \theta_{x_1^i, x_2^i} \prod_{c_1=1}^{k} \prod_{c_2=1}^{k} \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2} \\
&= \prod_{c_1=1}^{k} \prod_{c_2=1}^{k} \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2} \theta_{c_1,c_2}^{n_{c_1,c_2}},
\end{aligned}$$

where n_{c_1,c_2} is the number of observations of (c_1, c_2)

$$= \prod_{c_1=1}^{k} \prod_{c_2=1}^{k} \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2+n_{c_1,c_2}}$$

$$f(\Theta) = \sum_{c_1=1}^{k} \sum_{c_2=1}^{k} (\alpha_{c_1} + \alpha_{c_2} - 2 + n_{c_1,c_2}) \log(\theta_{c_1,c_2})$$

using $\sum_{c_1=1}^{k} \sum_{c_2=1}^{k} \theta_{c_1,c_2} = 1$,

$$\frac{\partial f(\Theta)}{\partial \theta_{c_1,c_2}} = \frac{\alpha_{c_1} + \alpha_{c_2} - 2 + n_{c_1,c_2}}{\theta_{c_1,c_2}} - \frac{\sum_{c_1'=1, c_1' \neq c_1}^{k} \sum_{c_2'=1, c_2' \neq c_2}^{k} \alpha_{c_1'} + \alpha_{c_2'} - 2 + n_{c_1',c_2'}}{1 - \theta_{c_1,c_2}} \overset{\text{set}}{=} 0$$

$$\hat{\theta_{c_1,c_2}} = \frac{\alpha_{c_1} + \alpha_{c_2} - 2 + n_{c_1,c_2}}{\sum_{c_1'=1}^{k} \sum_{c_2'=1}^{k} \alpha_{c_1'} + \alpha_{c_2'} - 2 + n_{c_1',c_2'}}$$

3. We often use discrete distributions as parts of more-complicated distributions (like mixture models and graphical models). In these cases we often need to fit a weighted NLL for the form

$$f(\Theta) = -\sum_{i=1}^{n} v_i \log p(x_1^i, x_2^i \mid \Theta),$$

with n non-negative weights. What is the MLE for the k^2 elements of Θ under this weighting.

$$\frac{\delta f(\Theta)}{\delta \theta_{c_1,c_2}} = \frac{-\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i = c_1 \wedge x_2^i = c_2]}{\theta_{c_1,c_2}} + \frac{-\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i \neq c_1 \wedge x_2^i \neq c_2]}{1 - \theta_{c_1,c_2}} = 0$$

$$\theta_{c_1,c_2}(\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i \neq c_1 \wedge x_2^i \neq c_2]) = (1 - \theta_{c_1,c_2})(-\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i = c_1 \wedge x_2^i = c_2])$$

$$\theta_{c_1,c_2}(\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i \neq c_1 \wedge x_2^i \neq c_2]) = -\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i = c_1 \wedge x_2^i = c_2] + \theta_{c_1,c_2} \sum_{i=1}^{k} v_i \mathbb{I}[x_1^i = c_1 \wedge x_2^i = c_2]$$

$$\theta_{c_1,c_2}(\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i \neq c_1 \wedge x_2^i \neq c_2] - \sum_{i=1}^{k} v_i \mathbb{I}[x_1^i = c_1 \wedge x_2^i = c_2]) = \sum_{i=1}^{k} v_i \mathbb{I}[x_1^i = c_1 \wedge x_2^i = c_2]$$

$$\hat{\theta}_{c_1,c_2} = \frac{\sum_{i=1}^{k} v_i \mathbb{I}[x_1^i = c_1 \wedge x_2^i = c_2]}{\sum_{i=1}^{n} v_i}$$

4. Consider the following alternate parameterization of the joint distribution: we use θ_{c_1} as the ("marginal") probability that variable 1 is in state c_1 ($p(x_1 = c_1) = \theta_{c_1}$) and $\theta_{c_2 \mid c_1}$ as the ("conditional") probability that variable 2 is in state c_2 conditioned on variable 1 being in state c_1 ($p(x_2 = c_2 \mid x_1 = c_1) = \theta_{c_2 \mid c_1}$).[2] Give the MLE for the k elements of θ_{c_1} and k^2 elements of $\theta_{c_2 \mid c_1}$ (you don't need to give the full derivation, but should know how this would be done).

$$\hat{\theta}_{c_1} = \frac{\sum_{i=1}^{n} \mathcal{I}[x_1^i = c_1]}{n} = \frac{n_{c_1}}{n}$$

$$\hat{\theta}_{c_1|c_2} = \frac{\sum_{i=1}^{n} \mathcal{I}[x^i = (c_1, c_2)]}{n_{c_1}} = \frac{n_{c_1,c_2}}{n_{c_1}}$$

Hint: it may be convenient to write the discrete likelihood for an example i in the form

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

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 a Lagrangian to enforce the sum-to-1 constraint on the log-likelihood (bug the TAs to show you how to do this in office hours or tutorial if you don't know how), and you may find it convenient to define $n_c = \sum_{i=1}^{n} \mathcal{I}[x^i = c]$.

[2] So we could use $p(x_1, x_2) = p(x_2 \mid x_1)p(x_1)$ to get the joint probability.

2.2 Gaussian Self-Conjugacy

Consider n IID samples x^i distributed according to a Gaussian with mean μ and covariance $\sigma^2 I$,

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

Assume that μ itself is distributed according to a Gaussian

$$\mu \sim \mathcal{N}(\mu_0, \Sigma_0),$$

with mean μ_0 and (positive-definite) covariance Σ_0. In this setting, the posterior for μ also follows a Gaussian distribution.[3]

Derive the form of the posterior distribution, $p(\mu \mid X, \sigma^2, \mu_0, \Sigma_0)$.
Hint: the posterior is a product of Gaussian densities. Answer:

$$\begin{aligned}
p(\mu \mid X, \sigma^2, \mu_0, \Sigma_0) &\propto p(X \mid \mu, \sigma^2, \mu_0, \Sigma_0) p(\mu \mid \mu_0, \Sigma_0) \\
&= \prod_{i=1}^{n} p(x^i \mid \mu, \sigma^2, \mu_0, \Sigma_0) p(\mu \mid \mu_0, \Sigma_0) \\
&= \left(\frac{1}{(2\pi)^{\frac{k}{2}} \det(\sigma^2 I)^{\frac{1}{2}}} \right)^n e^{\left(\frac{-1}{2} \sum_{i=1}^{n} (x^i - \mu)^\top (\sigma^2 I)(x^i - \mu)\right)} \frac{1}{(2\pi)^{\frac{k}{2}} \det(\Sigma)^{\frac{1}{2}}} e^{\left(\frac{-1}{2}(\mu - \mu_0)^\top \Sigma_0^{-1} (\mu - \mu_0)\right)} \\
&\propto \exp\left(-\frac{1}{2} \left[\sum_{i=1}^{n} (x^i - \mu)^\top (\sigma^2 I)^{-1} (x^i - \mu) + (\mu - \mu_0)^\top \Sigma^{-1} (\mu - \mu_0) \right] \right) \\
&\propto e^{\left(-\frac{1}{2}\left[(\mu - ((\Sigma_0^{-1} + (n\sigma^2 I)^{-1})^{-1}((\Sigma_0^{-1}\mu_0) + (n\sigma^2 I)^{-1}\bar{x}))^\top (\Sigma_0^{-1} + (n\sigma^2 I)^{-1})(\mu - (\Sigma_0^{-1} + (n\sigma^2 I)^{-1})^{-1}((\Sigma_0^{-1}\mu_0) + (n\sigma^2 I)^{-1}\bar{x}))\right]\right)}
\end{aligned}$$

This gives a posterior distribution to be of the form

$$\mu \mid X, \sigma^2, \mu_0, \Sigma_0 \sim \mathcal{N}((\Sigma_0^{-1} + (n\sigma^2 I)^{-1})^{-1}((\Sigma_0^{-1}\mu_0) + (n\sigma^2 I)^{-1}\bar{x}), (\Sigma_0^{-1} + (n\sigma^2 I)^{-1})^{-1})$$

, where $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x^i$

[3] Because of the prior and posterior coming from the same family, 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.

2.3 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

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

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

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

where in our case the total number of classes k is 3.[4] Modeling $p(y^i = c \mid \Theta)$ is easy: we can just use a k-state categorical distribution,

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

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 $p(x^i \mid y^i = c, \Theta)$ 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,

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

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

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

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

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

In class we derived the MLE for this model under the assumption that we use full covariance matrices and that each class has its own covariance.

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

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.)

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

$$\begin{aligned}\nabla_{\mu_c} l &= \nabla_{\mu_c} \sum_{i=1}^{n} \frac{1}{2}(x^i - \mu_{y^i})^T D^{-1}(x^i - \mu_{y^i}) \\ &= \sum_{i \in y_c} D^{-1}(x^i - \mu_c) = \sum_{i \in y_c} x^i - \sum_{i \in y_c} \mu_c = \sum_{i \in y_c} x^i - \mu_c n_c = 0\end{aligned}$$

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

$$\begin{aligned}\nabla_D l &= \sum_{i=1}^{n} \frac{1}{2}(x^i - \mu_{y^i})(x^i - \mu_{y^i})^T - \frac{1}{2} D \\ &= \frac{1}{2}(\sum_{i=1}^{n}((x^i - \mu_{y^i})(x^i - \mu_{y^i})^T) - nD) = 0\end{aligned}$$

$$\hat{D} = \frac{1}{n} \sum_{i=1}^{n} (x^i - \mu_{y^i})(x^i - \mu_{y^i})^T$$

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

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

$$\begin{aligned}\nabla_{\mu_c} l &= \nabla_{\mu_c} \sum_{i \in y_c} \frac{1}{2}(x^i - \mu_{y^i})^T \Sigma_c^{-1}(x^i - \mu_{y^i}) \\ &= \sum_{i \in y_c} \Sigma_c^{-1}(x^i - \mu_c) = \sum_{i \in y_c} x^i - n\mu_c = 0\end{aligned}$$

$$\hat{\mu_c} = \frac{\sum_{i=1}^{n} x^i}{n} = \bar{x}$$

$$\nabla_{\Sigma_c} l = \frac{1}{2} \sum_{i \in y_c} (x^i - \mu_c)(x^i - \mu_c)^T - \frac{1}{2} \sum_{i \in y_c} \Sigma_c = 0$$

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

3. 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.

   Test set accuracy: 0.624

```
function gda(X, y)
    # Filter and density estimation to get mu and Sigma for each class
    k = length(unique(y))
    d = size(X)[2]
    X_with_labels = [y X]
    mus = Array{Float64}(undef,k,d)
    Sigmas = Array{Float64}(undef,k,d,d)
    pis = Array{Float64}(undef,k)
    globalMu = mean(X, dims=1)
    for c in 1:k
        # filter data by class label, remove label column, and center
        Xc = X_with_labels[X_with_labels[:,1] .== c, :]
        Xc = Xc[:, 1:end .!= 1] .- globalMu
        pis[c] = length(Xc)/length(X)
        # mean vector containing mean of each feature, by label
        mu = mean(Xc, dims=1)
        mus[c,:] = mu
        # calculate Sigma for this class
        Sigmas[c,:,:] = cov(Xc)
    end

    function PDF(mu, Sigma, pic, xi)
        deter = -0.5 * logdet(Sigma)
        expon = -0.5 * ((xi - mu)'*inv(Sigma)*(xi - mu))
        p = log(pic) .+ expon .+ deter
        return p[1][1]
    end

    function predict(Xhat)
        xhatMu = mean(Xhat, dims=1)
        yhat = Vector{Int32}()
        for xi in eachrow(Xhat)
            probabilities = Vector{Float64}()
            for c in 1:k
                p = PDF(mus[c,:], Sigmas[c,:,:], pis[c], xi - xhatMu')
                append!(probabilities, p)
            end
            # Use PDF and responsibility to get categorical distribution
            yi = findmax(probabilities)[2]
            append!(yhat, yi)
        end
        return yhat
    end
    return GenericModel(predict)
end
```

4. 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.
Test set accuracy: 0.826

```
function tda(X, y, k)
    X_with_labels = [y X]
    subModels = Array{DensityModel}(undef,k)
    priors = Array{Float64}(undef,k)
    for c in 1:k
        # filter data by class label, remove label column, and center
        Xc = X_with_labels[X_with_labels[:,1] .== c, :][:, 1:end .!= 1]
        # prior probability pi_c
        priors[c] = length(Xc)/length(X)
        # create and store density model for this specific class
        subModels[c] = studentT(Xc)
    end

    function predict(Xhat)
        n,d = size(Xhat)
        # calculate pdfs for each class
        probabilties = Array{Float64}(undef,n,k)
        for c in 1:k
            pdfs = subModels[c].pdf(Xhat)
            probabilties[:,c] = pdfs .* priors[c]
        end
        # for each xi, yi = index of largest pdf
        yhat = Array{Int32}(undef,n)
        for (i,row) in enumerate(eachrow(probabilties))
            val, yi = findmax(row)
            yhat[i] = yi
        end
        return yhat
    end
    return GenericModel(predict)
end
```

Hints: you may be able to substantially simplify the notation in the MLE derivations 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 the implementation 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 may find it helpful to define an empty array that can be filled with k *DensityModel* objects using:

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

3 Mixture Models and Expectation Maximization

3.1 Categorical Mixture Model

Consider density estimation with examples $x^i \in \{1, 2, \dots, k\}^d$ representing a set of d categorical variables. In this setting, a natural way to model an individual variable x_j^i would be with a categorical distribution,

$$p(x_j^i = c \mid \theta_{j,1}, \theta_{j,2}, \dots, \theta_{j,k}) = \theta_{j,c},$$

so the joint distribution would be

$$p(x^i \mid \theta_{j,1}, \theta_{j,2}, \dots, \theta_{j,k}) = \prod_{j=1}^{d} \theta_{j,x_j^i},$$

where all $\theta_{j,c} \geq 0$ and $\sum_{c=1}^{k} \theta_{j,c} = 1$ for each feature j. However, in this distribution the variables would are independent. One way to model dependent count variables would be with a mixture of m independent categorical distributions,

$$p(x^i \mid \Theta) = \sum_{c=1}^{m} p(z^i = c \mid \Theta^t) p(x^i \mid z^i = c, \Theta^t) = \sum_{c=1}^{m} \pi_c \prod_{j=1}^{d} \theta_{j,x_j^i}^c,$$

where:

- π_c is the probability that the examples comes from mixture c, $p(z^i = c \mid \Theta) = \pi_c$.
- $\theta_{j,c'}^c$ is the probability that x_j^i is c' for examples from mixture c, $p(x_j^i = c' \mid z^i = c, \Theta)$.
- We use Θ as the set containing all the π_c and $\theta_{j,c'}^c$ values.

Derive the EM update for this mixture model (treating the z^i as missing values).

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

$$
\begin{aligned}
Q(\Theta \mid \Theta^t) &= \sum_{i=1}^{n}\sum_{c=1}^{m} p(z^i = c \mid x^i, \Theta^t) \log p(z^i = c, x^i \mid \Theta) \\
r_c^i &= p(z^i = c \mid x^i, \Theta^t) = \frac{p(z^i = c, x^i \mid \Theta^t)}{p(x^i \mid \Theta^t)} \\
&= \frac{\pi_c^t \prod_{j=1}^{d} (\theta^c_{j,x_j^i})^t}{\sum_{c'=1}^{m} \pi_{c'}^t \prod_{j=1}^{d} (\theta^{c'}_{j,x_j^i})^t} \\
\log(p(z^i = c, x^i \mid \Theta) &= \log(\pi_c \prod_{j=1}^{d} \theta^c_{j,x_j^i}) \\
&= \log(\pi_c) + \sum_{j=1}^{d} \log(\theta^c_{j,x_j^i}) \\
Q(\Theta \mid \Theta^t) &= \sum_{c=1}^{m}\sum_{i=1}^{n} r_i^c \left(\log \pi_c + \sum_{j=1}^{d} \log(\theta^c_{j,x_j^i}) \right) \\
\frac{\partial Q(\Theta \mid \Theta^t)}{\partial \pi_c} &= \frac{\sum_{i=1}^{n} r_c^i}{\pi_c} - \frac{n - \sum_{i=1}^{n} r_c^i}{1 - pi_c} \stackrel{\text{set}}{=} 0 \\
\pi_c^{t+1}(n - \sum_{i=1}^{n} r_c^i) &= (1 - \pi_c^{t+1}) \sum_{i=1}^{n} r_c^i \\
n\pi_c^{t+1} &= \sum_{i=1}^{n} r_c^i \\
\pi_c^{t+1} &= \frac{\sum_{i=1}^{n} r_c^i}{n} \\
\frac{\partial Q(\Theta \mid \Theta^t)}{\partial \theta^c_{j,k}} &= \frac{\sum_{i=1}^{n} r_c^i \mathcal{I}[x_k^i = k]}{\theta^c_{j,k}} - \frac{\sum_{i=1}^{n} r_c^i - \sum_{i=1}^{n} r_c^i \mathcal{I}[x_k^i = k]}{1 - \theta^c_{j,k}} \stackrel{\text{set}}{=} 0 \\
\sum_{i=1}^{n} r_c^i (\theta^c_{j,k})^{t+1} &= \sum_{i=1}^{n} r_c^i \mathcal{I}[x_j^i = k] \\
(\theta^c_{j,k})^{t+1} &= \frac{\sum_{i=1}^{n} r_c^i \mathcal{I}[x_j^i = k]}{\sum_{i=1}^{n} r_c^i}
\end{aligned}
$$

3.2 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 techniques. 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. Answer:

$$\text{let } pnorm(x^i \mid \mu, \Sigma) \text{ be the Gaussian PDF with parameters } \mu, \Sigma$$

$$Q(\Theta \mid \Theta^t) = \sum_{i=1}^{n} \log p(y^i, x^i \mid \Theta) + \sum_{i=1}^{t} \sum_{c=1}^{k} r_c^i \log p(\tilde{y}^i = c, \tilde{x}^i \mid \Theta)$$

$$r_{y^i}^i = p(y^i = c \mid x^i, \Theta) = \frac{p(y^i = c)p(x^i \mid y^i = c, \Theta)}{\sum_{c'=1}^{m} p(y^i = c')p(x^i \mid y^i = c', \Theta)}$$

$$= \frac{\pi_c pnorm(x^i \mid \mu_c, \Sigma_c)}{\sum_{c'=1}^{m} \pi_{c'} pnorm(x^i \mid \mu_{c'}, \Sigma_{c'})}$$

$$Q(\Theta \mid \Theta^t) \propto \sum_{i=1}^{n} \left(\log(\pi_{y^i}) + \frac{-1}{2} \log(\det(\Sigma_{y^i})) + \frac{-1}{2} (x^i - \mu_{y^i})^\top \Sigma_{y^i}^{-1} (x^i - \mu_{y^i}) \right)$$

$$+ \sum_{i=1}^{t} \sum_{c=1}^{k} r_c^i \left(\log(\pi_c) + \frac{-1}{2} \log(\det(\Sigma_c)) + \frac{-1}{2} (\tilde{x}^i - \mu_c)^\top \Sigma_c^{-1} (\tilde{x}^i - \mu_c) \right)$$

$$\frac{\partial Q(\Theta \mid \Theta^t)}{\partial \pi_c} = \frac{n_c + \sum_{i=1}^{t} r_c^i}{\pi_c} - \frac{n - n_c + t - \sum_{i=1}^{t} r_c^i}{1 - \pi_c} \stackrel{\text{set}}{=} 0$$

$$(n + t)\pi_c^{t+1} = n_c + \sum_{i=1}^{t} r_c^i$$

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

$$\frac{\partial Q(\Theta \mid \Theta^t)}{\partial \mu_c} = \sum_{i:y^i=c} \Sigma_c^{-1}(x^i - \mu_c) + \sum_{i=1}^{t} r_c^i \Sigma_c^{-1}(\tilde{x}^i - \mu_c) \stackrel{\text{set}}{=} 0$$

$$0 = \sum_{i:y^i=c} x^i - n_c \mu_c^{t+1} + \sum_{i=1}^{t} r_c^i \tilde{x}^i - \sum_{i=1}^{t} r_c^i \mu_c^{t+1}$$

$$\mu_c^{t+1} = \frac{\sum_{i:y^i=c} x^i + \sum_{i=1}^{t} r_c^i \tilde{x}^i}{n_c + \sum_{i=1}^{t} r_c^i}$$

$$\frac{\partial Q(\Theta \mid \Theta^t)}{\partial \Sigma_c^{-1}} = \frac{1}{2} \sum_{i:y^i=c} (x^i - \mu_c)(x^i - \mu_c)^\top - \frac{1}{2} \sum_{i:y^i=c} \Sigma_c + \frac{1}{2} \sum_{i=1}^{t} r_c^i (\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^\top - \frac{1}{2} \sum_{i=1}^{t} r_c^i \Sigma_c \stackrel{\text{set}}{=} 0$$

$$(n_c + \sum_{i=1}^{t} r_c^i)\Sigma_c^{t+1} = \sum_{i:y^i=c} (x^i - \mu_c)(x^i - \mu_c)^\top + \sum_{i=1}^{t} r_c^i (\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^\top$$

$$\Sigma_c^{t+1} = \frac{\sum_{i:y^i=c} (x^i - \mu_c)(x^i - \mu_c)^\top + \sum_{i=1}^{t} r_c^i (\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^\top}{n_c + \sum_{i=1}^{t} r_c^i}$$

2. If you run the demo *example_SSL.jl*, it will load a variant of the dataset from the GDA, 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 the GDA question 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 the GDA question. Write a function *gdaSSL* 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. Answer: Test error: 0.352

```
function logPDF(xi,pic,mu,Sigma)
    deter = -0.5 * logdet(Sigma)
    expon = -0.5 * ((xi - mu)'*inv(Sigma)*(xi - mu))
    p = log(pic) .+ expon .+ deter
    return p[1][1]
end

function PDF(xi,pic,mu,Sigma)
    (d,) = size(mu)
    temp = 1/((2*pi)^(d/2)*det(Sigma)^(1/2))
    p = temp * exp(-((Sigma \ xi)' * xi) ./2)
    return p[1] * pic
end

function responsibilities(X,pis,mus,Sigmas,t,k)
    r = Array{Float64}(undef,t,k)
    for i in 1:t
        pxy = Array{Float64}(undef,k)
        for c in 1:k
            pxy[c] = PDF(X[i,:], pis[c], mus[c,:], Sigmas[c,:,:])
        end
        r[i,:] = pxy ./ sum(pxy)
    end
    return r
end

# E-step: expectation of complete log-likelihood given last parameters Theta^t
function Qfunction(X,y,Xtest,pis,mus,Sigmas,r,n,t,k)
    q = 0
    for i in 1:n
        q += logPDF(X[i,:],pis[y[i]],mus[y[i],:],Sigmas[y[i],:,:])
    end
    for i in 1:t
        for c in 1:k
            q += r[i,c] * logPDF(Xtest[i,:],pis[c],mus[c,:],Sigmas[c,:,:])
        end
    end
    return q
end
```

```
# M-step: update parameters Theta^t+1 given last parameters Theta^t
function emUpdate(X,y,Xtest,pis,mus,Sigmas,r,n,t,k,ncs)
    for c in 1:k
        Xc = X[y.==c,:]
        pis[c] = (ncs[c] + sum(r[:,c])) / n+t
        muSums = sum(Xc, dims=1) + sum(r[:,c].*Xtest, dims=1)
        mus[c,:] = muSums ./ (ncs[c] + sum(r[:,c]))
        varSums = (Xc' .- mus[c,:]) * (Xc' .- mus[c,:])'
        varSums += r[:,c]' .* (Xtest' .- mus[c,:]) * (Xtest' .- mus[c,:])'
        Sigmas[c,:,:] = varSums ./ (ncs[c] + sum(r[:,c]))
    end
    return pis,mus,Sigmas
end

function gdaSSL(X,y,Xtest;maxIters=50,epsilon=1e-5)
    k = length(unique(y))
    n,d = size(X)
    t,_ = size(Xtest)
    mus = Array{Float64}(undef,k,d)
    Sigmas = Array{Float64}(undef,k,d,d)
    pis = ones(k)./k
    ncs = counts(y,k)
    # Center data and initialize mus and Sigmas
    X = X .- mean(X,dims=1)
    Xtest = Xtest .- mean(Xtest,dims=1)
    for c in 1:k
        mus[c,:] = mean(X, dims=1)
        Sigmas[c,:,:] = cov(X)
    end
    # Expectation maximization
    Q = -Inf
    Qs = Array{Float64}(undef,maxIters)
    for iter in ProgressBar(1:maxIters)
        r = responsibilities(Xtest,pis,mus,Sigmas,t,k)
        Qt = Qfunction(X,y,Xtest,pis,mus,Sigmas,r,n,t,k) # E-step
        pis,mus,Sigmas = emUpdate(X,y,Xtest,pis,mus,Sigmas,r,n,t,k,ncs) # M-step
        Qs[iter] = Qt
        if Qt - Q <= epsilon
            println("Iterations: $iter\nQt-Q=$(Qt-Q)")
            break
        end
        Q = Qt
    end
    display(plot((1:maxIters),Qs))
    predict(Xhat) = gdaPredict(Xhat,k,mus,Sigmas,pis)
    return GenericModel(predict)
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 resulted in fewer iterations (from 46 to 43) and worse performance, training error increased to 0.377

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 combination of updates on its own to make sure they are correct.

4 Very-Short Answer Questions

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

1. What is a situation where we can violate the golden rule on our test set, and still have it be a reasonable approximation of test error? We violate the golden rule when using a validation set since the validation set influences training, but we use the validation set error as an approximation of test error.
2. Suppose we have a way to generate new IID samples for our problem. If we are fitting a model with SGD, what would be the key advantage of using new IID samples as opposed to sampling from a fixed training set? Adding new IID samples will likely improve test error, since test error decreases proportionally to the size of the training set, $O(1/n)$. Sampling from a fixed training set would not introduce any new information to the model.
3. Is the minimum of a convex function achieved by a unique input value? What about a strictly-convex function or a strongly-convex function? The minimum of: a convex function is achieved by any stationary point, a strictly-convex function is achieved by at most one stationary point, a strongly-convex function is achieved by at least one stationary point.
4. How can we use density estimation for supervised learning? Given data x^i and targets y^i, we can perform density estimation on (x^i, y^i), conditioning to give the probability of a target y^i given example x^i, $p(y^i|x^i)$.
5. How many parameters does the general discrete distribution have when each feature is a categorical variable that can take up to k values. d features, k values. There is some probability $\theta_{d,k}$ for each value, k, of each feature, d. There are a total of dk parameters.
6. What is the relationship between the covariance matrix Σ and the precision matrix Θ of a multivariate Gaussian? $\Theta = \Sigma^{-1}$

7. Suppose we run the graphical LASSO method and it returns a tri-diagonal precision matrix. What would the graph look like? We would have a three-dimensional graph that forms a cubic lattice structure.

8. Suppose we have a lot of extreme outliers in our dataset. Why is this less of a problem for a mixture of Gaussians than if we use a single Gaussian? The mean is not a robust measurement, so it is easily skewed by outliers. When we have a mixture of Gaussians, the distribution of the outliers can be captured by separate Gaussians such that it does not skew our final distribution.

9. Why do the π_c need to be a convex combination in mixture models? The π_c need to be a convex combination because they must be non-negative and sum to 1.

10. What is the relationship between the supervised naive Bayes model and the unsupervised mixture of Bernoullis model? In naive Bayes, we assume $x^i|y^i$ is a product of Bernoullis where we know the categories y^i. In a mixture of Bernoullis, we don't know y^i.

11. What is the difference between "missing at random" and "missing completely at random"? MCAR is a stronger version of MAR where there is absolutely no relationship between the missing data and any values in the data set, missing or observed. The missing data are just a random subset of the data. MAR only means that there is no relationship between the missing data and the missing values, but there is some relationship to some of the observed data.

12. What is an advantage of the Epanechnikov kernel over the Gaussian kernel. The Epanechnikov kernel is much faster, since it only depends on nearby points. With the Gaussian kernel, we need to compute the distance of each test point to every training point.

13. How does parameter tieing allow us to model training examples that have different sizes? Since parameter tieing means using the same parameters for different features j, different parts of the model use the same parameters which allows us to use the same model for any number of features. The features of the data can be expanded to form one long data set.