CPSC 540 Assignment 3 (due February 22 at midnight)

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

1. Name(s): Dingqing Yang, Junjie Zhu
2. Student ID(s): 38800141, 30921167

1 Discrete and Gaussian Variables

1.1 MLE for General Discrete Distribution

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

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

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

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

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.

1.1 MLE for General Discrete Distribution.

1. $L(\theta) = \prod_{i=1}^{n} p(x_1^i, x_2^i \mid \theta)$, so we maximize:

$$\ell(\theta, \lambda) = \log\left(\prod_{i=1}^{n} p(x_1^i, x_2^i \mid \theta)\right) - \lambda\left(\sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \theta_{c_1,c_2} - 1\right)$$

$$= \sum_{i=1}^{n} \log(p(x_1^i, x_2^i \mid \theta)) - \lambda\left(\sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \theta_{c_1,c_2} - 1\right)$$

$$= \sum_{i=1}^{n} \log(\theta_{x_1^i, x_2^i}) - \lambda\left(\sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \theta_{c_1,c_2} - 1\right)$$

$$\therefore \frac{d\ell}{d\theta_{c_1,c_2}} = \sum_{i=1}^{n} 1_{(c_1,c_2)}(x_1^i, x_2^i) \cdot \frac{1}{\theta_{c_1,c_2}} - \lambda = \frac{1}{\theta_{c_1,c_2}} \{\# \text{ of } (c_1,c_2) \text{ in } X\} - \lambda.$$

$$\frac{d\ell}{d\lambda} = -\left(\sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \theta_{c_1,c_2} - 1\right)$$

– now set all $\frac{d\ell}{d\theta_{c_1,c_2}}$, $\frac{d\ell}{d\lambda}$ to be 0.

$$\therefore \sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \theta_{c_1,c_2} = 1, \quad \frac{1}{\theta_{c_1,c_2}}\{\# \text{ of } (c_1,c_2) \text{ in } X\} = \lambda \quad \forall (c_1,c_2) \in \{1,2,\dots k\}^2.$$

$$\therefore \frac{\{\# \text{ of } (c_1,c_2) \text{ in } X\}}{\lambda} = \theta_{c_1,c_2}, \quad 1 = \sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \theta_{c_1,c_2} = \sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \frac{\{\# \text{ of } (c_1,c_2) \text{ in } X\}}{\lambda} = \frac{n}{\lambda}.$$

$$\therefore \lambda = n, \quad \theta_{c_1,c_2} = \frac{\{\# \text{ of } (c_1,c_2) \text{ in } X\}}{n} \qquad \forall (c_1,c_2) \in \{1,2,\dots k\}^2.$$

2. 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)

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

– then the log likelihood is $\ell(\theta) = \log\left(\prod_{i=1}^{n} p(x_1^i, x_2^i \mid \theta) \cdot \prod_{c_1=1}^{k}\prod_{c_2=1}^{k} p(\theta_{c_1,c_2})\right)$

$$\therefore \ell(\theta) = \sum_{i=1}^{n} \log(\theta_{x_1^i x_2^i}) + \sum_{c_1=1}^{k}\sum_{c_2=1}^{k} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2})$$

$$\therefore f(\theta) = -\sum_{i=1}^{n} \log(\theta_{x_1^i x_2^i}) - \sum_{c_1=1}^{k}\sum_{c_2=1}^{k} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2}), \text{ subject to } \sum_{c_1=1}^{k}\sum_{c_2=1}^{k} \theta_{c_1,c_2} = 1$$

- $\theta = \text{argmin} - \sum_{i=1}^n \log(\theta_{x_1^i, x_2^i}) - \sum_{c_1=1}^k \sum_{c_2=1}^k (\alpha_{c_1} + \alpha_{c_2} - 2) \log \theta_{c_1,c_2}$, subject to $\sum_{c_1=1}^k \sum_{c_2=1}^k \theta_{c_1,c_2} = 1$

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.

3. $f(\theta, \lambda) = -\sum_{i=1}^n v_i \log p(x_1^i, x_2^i \mid \theta) + \lambda\left(\sum_{c_1=1}^k \sum_{c_2=1}^k \theta_{c_1,c_2} - 1\right)$

$= -\sum_{i=1}^n v_i \log\left(\prod_{c\in[k]^2} \theta_c^{\mathbb{1}[x^i=c]}\right) + \lambda\left(\sum_{c_1=1}^k \sum_{c_2=1}^k \theta_{c_1,c_2} - 1\right)$

$\therefore \frac{df}{d\theta_{c'}} = -\sum_{i=1}^n v_i \frac{\mathbb{1}[x^i=c']}{\prod_{c\in[k]^2} \theta_c^{\mathbb{1}[x^i=c]}} \cdot \frac{\prod_{c\in[k]^2} \theta_c^{\mathbb{1}[x^i=c]}}{\theta_{c'}^{\mathbb{1}[x^i=c']}} + \lambda = -\sum_{i=1}^n \frac{v_i \cdot \mathbb{1}[x^i=c']}{\theta_{c'}^{\mathbb{1}[x^i=c']}} + \lambda$

$\frac{df}{d\lambda} = \sum_{c_1=1}^k \sum_{c_2=1}^k \theta_{c_1,c_2} - 1 = \sum_{c\in[k]^2} \theta_c - 1.$

– now set $\frac{df}{dc'}, \frac{df}{d\lambda} = 0,$

$\therefore \lambda = \sum_{i=1}^n \frac{v_i \mathbb{1}[x^i=c']}{\theta_{c'}^{\mathbb{1}[x^i=c']}} \quad \forall c' \in [k]^2, \qquad \therefore \lambda = \frac{1}{\theta_{c'}} \sum_{i=1}^n v_i \mathbb{1}[x^i=c'].$

$\therefore \theta_{c'} = \frac{\sum_{i=1}^n v_i \mathbb{1}[x^i=c']}{\lambda} \qquad \therefore \sum_{c\in[k]^2} \frac{\sum_{i=1}^n v_i \mathbb{1}[x^i=c]}{\lambda} = 1. \qquad \lambda = \sum_{c\in[k]^2} \sum_{i=1}$

$\lambda = \sum_{c\in[k]^2} \sum_{i=1}^n v_i \mathbb{1}[x^i=c] = \sum_{i=1}^n \sum_{c\in[k]^2} v_i \mathbb{1}[x^i=c] = \sum_{i=1}^n v_i$

$\therefore \theta_c = \frac{\sum_{i=1}^n v_i \mathbb{1}[x^i=c]}{\sum_{i=1}^n v_i} \qquad \cancel{\forall}\ \forall c \in [k]^2.$

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

1.2 Gaussian Self-Conjugacy and Posterior Convergence Rate

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.[1]

1. Derive the form of the posterior distribution, $p(\mu \mid X, \sigma^2, \mu_0, \Sigma_0)$.

1.2 Gaussian Self-Conjugacy and Posterior Convergence rate.

1. $p(\mu | X, \sigma^2, \mu_0, \Sigma_0) = \dfrac{p(X | \mu, \sigma^2, \mu_0, \Sigma_0) \cdot p(\mu | \sigma^2, \mu_0, \Sigma_0)}{p(X | \sigma^2, \mu_0, \Sigma_0)}$

$\therefore$ $p(\mu | X, \sigma^2, \mu_0, \Sigma_0)$ has density that is the product of $p(X | \mu, \sigma^2, \mu_0, \Sigma_0)$ and $p(\mu | \sigma^2, \mu_0, \Sigma_0) = p(\mu | \mu_0, \Sigma_0)$ (μ does not depend on σ).

- $p(\mu / \mu_0, \Sigma_0)$ has distribution $N(\mu_0, \Sigma_0)$

- $p(\overset{x^i}{\cancel{x}} | \mu, \sigma^2, \mu_0, \Sigma_0)$ has density $\dfrac{1}{(2\pi)^{\frac{d}{2}} \sigma^{\frac{d}{2}}} \cdot e^{\left(-\frac{1}{2}\left(\frac{1}{\sigma}I(x^i - \mu)\right)^2\right)} = \dfrac{1}{(2\pi)^{\frac{d}{2}} \sigma^{\frac{d}{2}}} e^{-\frac{1}{2}(\mu - x^i)^T(\sigma^2 I)^{-1}(\mu - x^i)}$

- view this as function of $\mu \Rightarrow p(x^i | \mu, \sigma^2, \mu_0, \Sigma_0)$ has densities of ~~a~~ a normal with mean x^i, and variance $\sigma^2 I$ $\therefore \mu \sim N(x^i, \sigma^2 I)$

$\therefore p(\mu | X, \sigma^2, \mu_0, \Sigma_0)$ has posterior distribution ~~and~~ with pdf being product of $N(x^i, \sigma^2 I)$ for $i \in \{1, \dots n\}$ and $N(\mu_0, \Sigma_0)$

$\therefore p(\mu | X^2, \sigma^2, \mu_0, \Sigma_0)$ follows the Gaussian distribution with

- covariance $\Sigma = (n \cdot \frac{1}{\sigma^2} I + \Sigma_0^{-1})^{-1}$
- mean $\mu = \Sigma\left(\frac{1}{\sigma^2} I \cdot \left(\sum_{i=1}^{n} x^i\right) + \Sigma_0^{-1} \mu_0\right)$ ~~$\therefore p(\mu | X^2, \sigma^2, \mu_0, \Sigma_0)$~~

~~$\therefore p(\mu | X^2, \sigma^2, \mu_0, \Sigma_0) \sim N((n$~~

$\therefore p(\mu | X^2, \sigma^2, \mu_0, \Sigma_0) \sim N\left((\frac{n}{\sigma^2} I + \Sigma_0^{-1})^{-1} \left(\frac{1}{\sigma^2}\left(\sum_{i=1}^{n} x^i\right) + \Sigma_0^{-1} \mu_0\right), (\frac{n}{\sigma^2} I + \Sigma_0^{-1})^{-1} \right)$.

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

2. To measure the speed at which the posterior converges to μ, we can consider the variance of the posterior. Consider a version of the previous question where $d = 1$: how many examples n (in O-notation) do we need before we have $|\Sigma| < \epsilon$ (for the posterior variance Σ).

2. when $d=1$, $\Sigma = \left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0}\right)^{-1}$ $(\Sigma_0 = \sigma_0)$
if $|\Sigma| < \varepsilon$, then $\left|\frac{n}{\sigma^2} + \frac{1}{\sigma_0}\right| > \frac{1}{\varepsilon} \Rightarrow \left|\frac{n}{\sigma^2}\right| > \frac{1}{\varepsilon} - \frac{1}{\sigma_0}$ $\therefore n > \sigma^2\left(\frac{1}{\varepsilon} - \frac{1}{\sigma_0}\right)$
$\therefore$ at least $\left\lceil \sigma^2\left(\frac{1}{\varepsilon} - \frac{1}{\sigma_0}\right)\right\rceil$ example is needed for $|\Sigma| < \varepsilon$ (Assuming $\varepsilon < \sigma_0 = \Sigma_0$)

1.3 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.[2] 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 $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^n$,

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

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

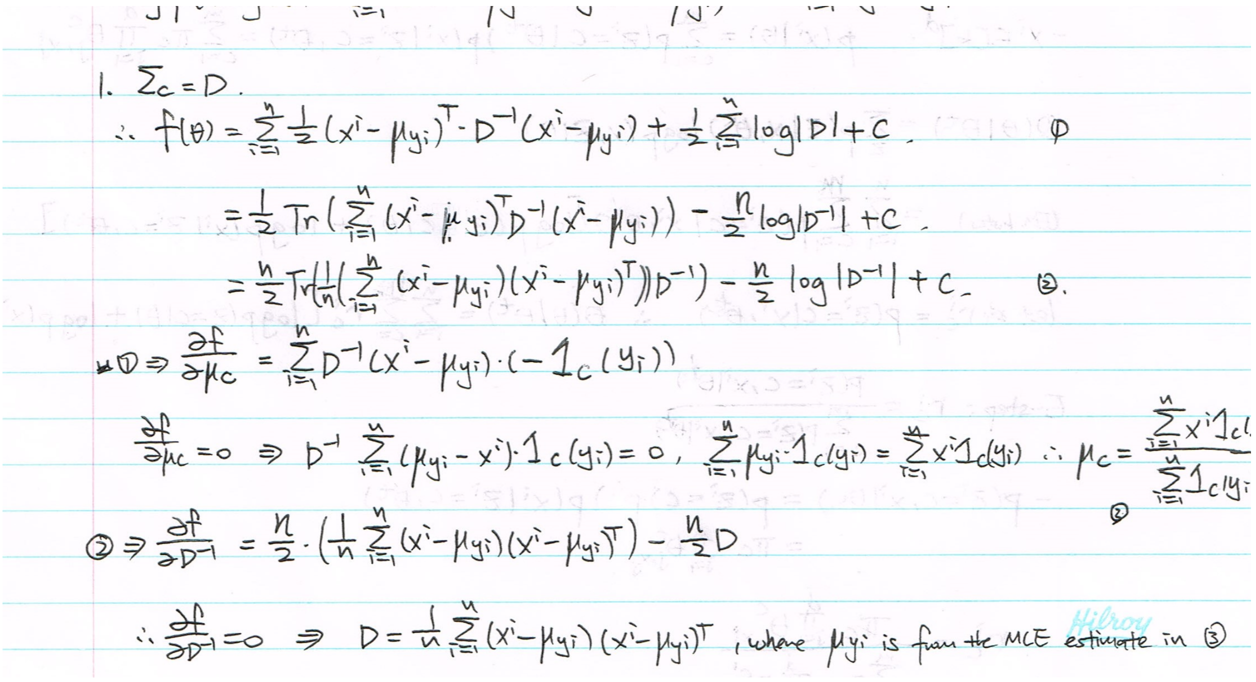
$$\underset{\mu_1,\mu_2,\ldots,\mu_k,\Sigma_1,\Sigma_2,\ldots,\Sigma_k}{\arg\max} \prod_{i=1}^{n} p(x^i \mid y^i, \mu_{y^i}, \Sigma_{y^i}).$$

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

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

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.

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.)



1. $\Sigma_c = D$.

$$\therefore f(\theta) = \sum_{i=1}^{n} \frac{1}{2}(x^i - \mu_{y_i})^T \cdot D^{-1}(x^i - \mu_{y_i}) + \frac{1}{2}\sum_{i=1}^{n} \log|D| + C. \quad \textcircled{1}$$

$$= \frac{1}{2} \mathrm{Tr}\left(\sum_{i=1}^{n}(x^i - \mu_{y_i})^T D^{-1}(x^i - \mu_{y_i})\right) - \frac{n}{2}\log|D^{-1}| + C.$$

$$= \frac{n}{2}\mathrm{Tr}\left(\frac{1}{n}\left(\sum_{i=1}^{n}(x^i - \mu_{y_i})(x^i - \mu_{y_i})^T\right)D^{-1}\right) - \frac{n}{2}\log|D^{-1}| + C. \quad \textcircled{2}$$

$$\textcircled{1} \Rightarrow \frac{\partial f}{\partial \mu_c} = \sum_{i=1}^{n} D^{-1}(x^i - \mu_{y_i}) \cdot (-\mathbb{1}_c(y_i))$$

$$\frac{\partial f}{\partial \mu_c} = 0 \Rightarrow D^{-1}\sum_{i=1}^{n}(\mu_{y_i} - x^i) \cdot \mathbb{1}_c(y_i) = 0, \quad \sum_{i=1}^{n}\mu_{y_i} \cdot \mathbb{1}_c(y_i) = \sum_{i=1}^{n} x^i \mathbb{1}_c(y_i) \quad \therefore \mu_c = \frac{\sum_{i=1}^{n} x^i \mathbb{1}_c(y_i)}{\sum_{i=1}^{n} \mathbb{1}_c(y_i)} \quad \textcircled{3}$$

$$\textcircled{2} \Rightarrow \frac{\partial f}{\partial D^{-1}} = \frac{n}{2} \cdot \left(\frac{1}{n}\sum_{i=1}^{n}(x^i - \mu_{y_i})(x^i - \mu_{y_i})^T\right) - \frac{n}{2}D$$

$$\therefore \frac{\partial f}{\partial D^{-1}} = 0 \Rightarrow D = \frac{1}{n}\sum_{i=1}^{n}(x^i - \mu_{y_i})(x^i - \mu_{y_i})^T$$, where μ_{y_i} is from the MLE estimate in $\textcircled{3}$

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

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

$$\therefore f(\theta) = \sum_{i=1}^{n} \frac{1}{2\sigma_{y_i}^2}(x^i - \mu_{y_i})^T(x^i - \mu_{y_i}) + \frac{1}{2}\sum_{i=1}^{n} \log|\sigma_{y_i}^2 I| + C. \quad ①.$$

$$= \sum_{i=1}^{n} \frac{1}{2\sigma_{y_i}^2}(x^i - \mu_{y_i})^T(x^i - \mu_{y_i}) + \frac{1}{2}\sum_{i=1}^{n} \log(\sigma_{y_i}^{2d}) + C$$

$$= \sum_{i=1}^{n} \frac{1}{2\sigma_{y_i}^2}(x^i - \mu_{y_i})^T(x^i - \mu_{y_i}) + \frac{d}{2}\sum_{i=1}^{n} d\log(\sigma_{y_i}^2) + C \quad ②.$$

$$① \Rightarrow \frac{\partial f}{\partial \mu_c} = \sum_{i=1}^{n} \frac{1}{\cancel{2}\sigma_{y_i}^2}(x^i - \mu_{y_i})(-1_c(y^i))$$

$$\therefore \frac{\partial f}{\partial \mu_c} = 0 \Rightarrow \sum_{i=1}^{n} \frac{1}{\sigma_{y_i}^2}(\mu_{y_i} - x^i)1_c(y^i) = 0, \quad \mu_c = \frac{\sum_{i=1}^{n} x^i 1_c(y^i)}{\sum_{i=1}^{n} 1_c(y^i)} \quad ③$$

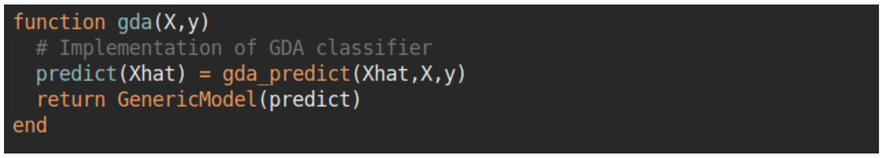
$$② \Rightarrow \frac{\partial f}{\partial \sigma_c} = \sum_{i=1}^{n} \frac{-1}{\sigma_{y_i}^3}(1_c(y^i))(x^i - \mu_{y_i})^T(x^i - \mu_{y_i}) + \frac{d}{2}\sum_{i=1}^{n} \frac{2\cdot\sigma_{y_i} 1_c(y^i)}{\sigma_{y_i}^2}$$

$$= \frac{-1}{\sigma_c^3}\left(\sum_{i=1}^{n}(x^i - \mu_{y_i})^T(x^i - \mu_{y_i})\cdot 1_c(y^i) - d\sigma_c^2 \cdot \sum_{i=1}^{n} 1_c(y_i)\right)$$

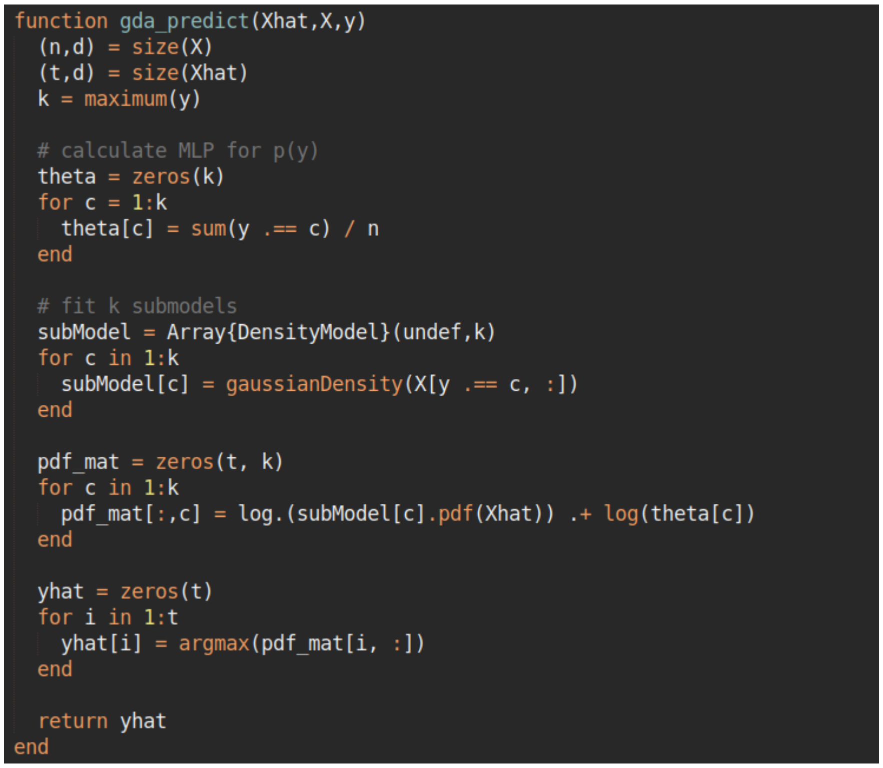
$$\therefore \frac{\partial f}{\partial \sigma_c} = 0 \Rightarrow \sigma_c^2 = \frac{1}{d}\cdot\frac{\sum_{i=1}^{n}(x^i - \mu_{y_i})^T(x^i - \mu_{y_i})1_c(y_i)}{\sum_{i=1}^{n} 1_c(y^i)}, \quad \text{where } \mu_{y_i} \text{ from MLE in } ③$$

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.

Answer: Test error obtained by gda is 0.369.
gda function and gda_predict function is shown below:



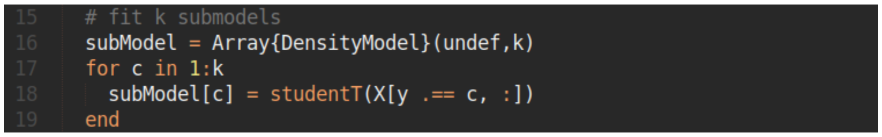
```
function gda(X,y)
  # Implementation of GDA classifier
  predict(Xhat) = gda_predict(Xhat,X,y)
  return GenericModel(predict)
end
```



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

  # calculate MLP for p(y)
  theta = zeros(k)
  for c = 1:k
    theta[c] = sum(y .== c) / n
  end

  # fit k submodels
  subModel = Array{DensityModel}(undef,k)
  for c in 1:k
    subModel[c] = gaussianDensity(X[y .== c, :])
  end

  pdf_mat = zeros(t, k)
  for c in 1:k
    pdf_mat[:,c] = log.(subModel[c].pdf(Xhat)) .+ log(theta[c])
  end

  yhat = zeros(t)
  for i in 1:t
    yhat[i] = argmax(pdf_mat[i, :])
  end

  return yhat
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.

Answer: Test error obtained by tda is 0.195.
tda is very similar to gda, and the only difference is to fit student_t for each label:



```
  # fit k submodels
  subModel = Array{DensityModel}(undef,k)
  for c in 1:k
    subModel[c] = studentT(X[y .== c, :])
  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)
```

2 Mixture Models and Expectation Maximization

2.1 Categorical Mixture Model

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

$$p(x^i_j = c \mid \theta_{j,1}, \theta_{j,2}, \ldots, \theta_{j,k}) = \theta_{j,c},$$

so the joint distribution would be

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

where all $\theta_{j,c} \geq 0$ and $\sum_{c=1}^{k} \theta_{j,c} = 1$ for each feature j. 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 m independent categorical distributions,

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

where:

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

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

2.1 Categorical Mixture Model

$- x^i \in [k]^d$, $\quad p(x^i|\theta) = \sum_{c=1}^{m} p(z^i = c|\theta^t)\, p(x^i | z^i = c, \theta^t) = \sum_{c=1}^{m} \pi_c \prod_{j=1}^{d} \theta^c_{j,x^i_j}$

$Q(\theta|\theta^t) = \sum_z p(z|x, \theta^k) \log p(x, z|\theta)$

(EM notes) $= \sum_{i=1}^{n} \sum_{c=1}^{m} p(z^i = c| x^i, \theta^t) [\log p(z^i = c|\theta) + \log p(x^i | z^i = c, \theta)]$

let ~~z~~ $r^i_c = p(z^i = c|x^i, \theta^t)$ $\quad \therefore\ Q(\theta|\theta^t) = \sum_{i=1}^{n}\sum_{c=1}^{m} r^i_c [\log p(z^i=c|\theta) + \log p(x^i|z^i=c,\theta$

E-step: $r^i_c = \dfrac{p(z^i = c, x^i|\theta^t)}{\sum_{c=1}^{m} p(z^i=c, x^i|\theta^t)}$

$- p(z^i = c, x^i|\theta_k) = p(z^i = c|\theta^t)\, p(x^i|z^i = c, \theta^t)$
$\qquad = \pi_c \cdot \prod_{j=1}^{d} \theta^c_{j,x^i_j}$

$\therefore\ r^i_c = \dfrac{\pi_c \cdot \prod_{j=1}^{d} \theta^c_{j,x^i_j}}{\sum_{c'=1}^{m} \pi_{c'} \cdot \prod_{j=1}^{d} \theta^{c'}_{j,x^i_j}}$

- $r_c^i = \dfrac{\pi_c \prod_{j=1}^d \theta^c_{j,x^i_j}}{\sum_{c'=1}^m \pi_{c'} \prod_{j=1}^d \theta^{c'}_{j,x^i_j}}$

M-step: $Q(\theta|\theta^t) = \sum_{i=1}^{n}\sum_{c=1}^{m} r^i_c (\log(\pi_c) + \log(\prod_{j=1}^{d} \theta^c_{j,x^i_j}))$

$\qquad = \sum_{i=1}^{n}\sum_{c=1}^{m} r^i_c [\log(\pi_c) + \sum_{j=1}^{d} \log(\theta^c_{j,x^i_j})]$

$\therefore\ \dfrac{dQ}{d\pi_c} = \sum_{i=1}^{n} r^i_c \cdot \dfrac{1}{\pi_c} = 0 \qquad \therefore\ \pi_c = \sum_{i=1}^{n} r^i_c$

$\dfrac{dQ}{d\theta^c_{j,p}} = \sum_{i=1}^{n} r^i_c \dfrac{1}{\theta^c_{j,p}} \cdot 1_p(x^i_j) \qquad \therefore\ \theta^c_{j,p} = \sum_{i=1}^{n} r^i_c \cdot 1_p(x^i_j)$ for $p \in [k]$

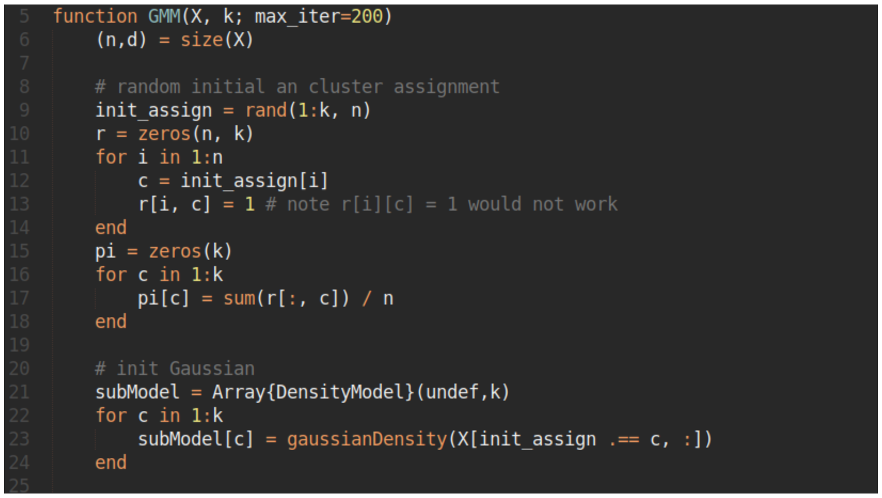
2.2 Gaussian Mixture Model Implementation

The script *example_gaussian* fits a Gaussian distribution to a dataset that is not uni-modal, and giving a bad fit. Implement the EM for fitting a mixture of Gaussians to this data. Hand in your code and the updated plot when using a mixture of 3 Gaussians.

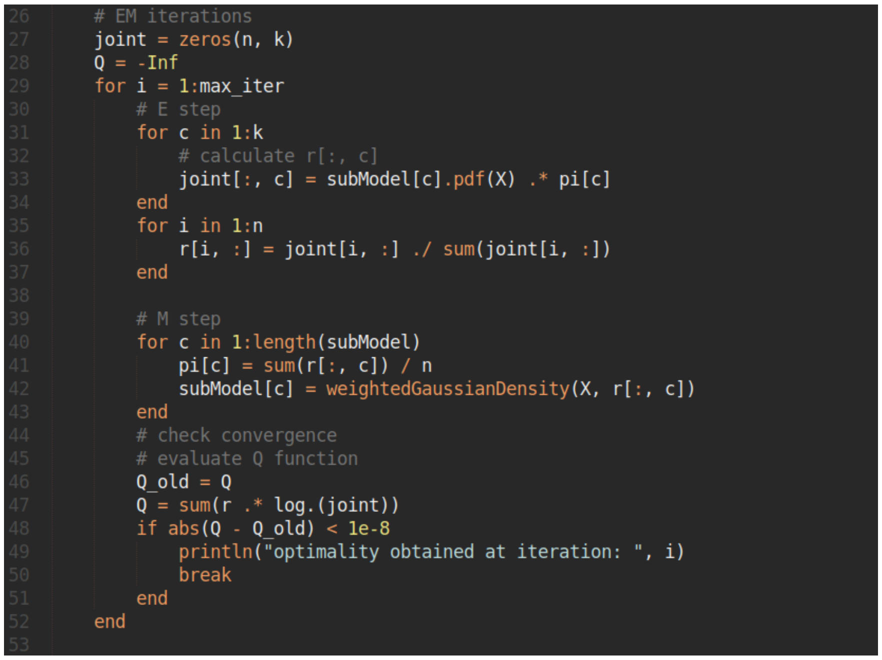
Hint: you may want to start by implementing the PDF ("predict") function; you can then use this with the monotonicity of EM to debug your implementation.

It is possible that Σ_k^{t+1} may not be positive-definite, if you encounter this the standard fixes are to remove such clusters or use a MAP estimate for Σ where you add a small multiple of the identity matrix to the estimate.

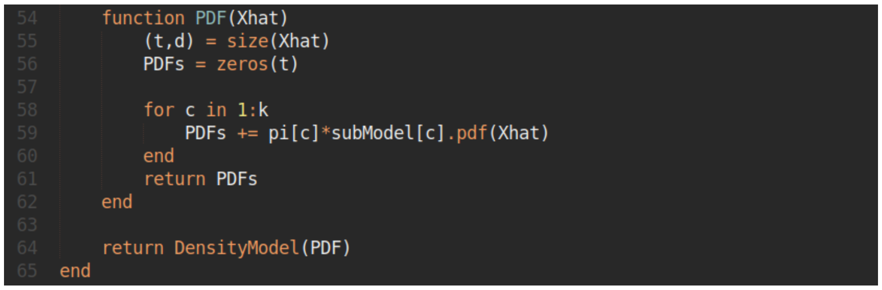
Answer: GMM function is shown below:



```
function GMM(X, k; max_iter=200)
    (n,d) = size(X)

    # random initial an cluster assignment
    init_assign = rand(1:k, n)
    r = zeros(n, k)
    for i in 1:n
        c = init_assign[i]
        r[i, c] = 1 # note r[i][c] = 1 would not work
    end
    pi = zeros(k)
    for c in 1:k
        pi[c] = sum(r[:, c]) / n
    end

    # init Gaussian
    subModel = Array{DensityModel}(undef,k)
    for c in 1:k
        subModel[c] = gaussianDensity(X[init_assign .== c, :])
    end

```

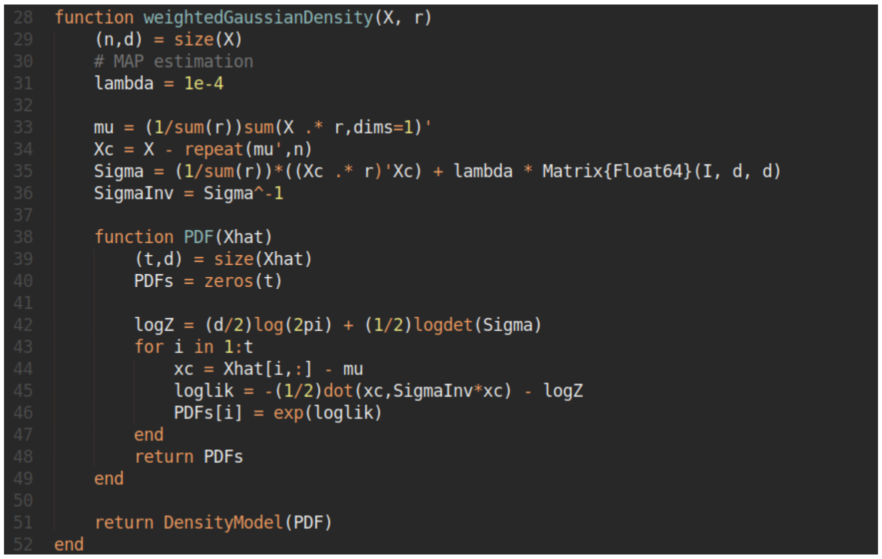


```
    # EM iterations
    joint = zeros(n, k)
    Q = -Inf
    for i = 1:max_iter
        # E step
        for c in 1:k
            # calculate r[:, c]
            joint[:, c] = subModel[c].pdf(X) .* pi[c]
        end
        for i in 1:n
            r[i, :] = joint[i, :] ./ sum(joint[i, :])
        end

        # M step
        for c in 1:length(subModel)
            pi[c] = sum(r[:, c]) / n
            subModel[c] = weightedGaussianDensity(X, r[:, c])
        end
        # check convergence
        # evaluate Q function
        Q_old = Q
        Q = sum(r .* log.(joint))
        if abs(Q - Q_old) < 1e-8
            println("optimality obtained at iteration: ", i)
            break
        end
    end

```



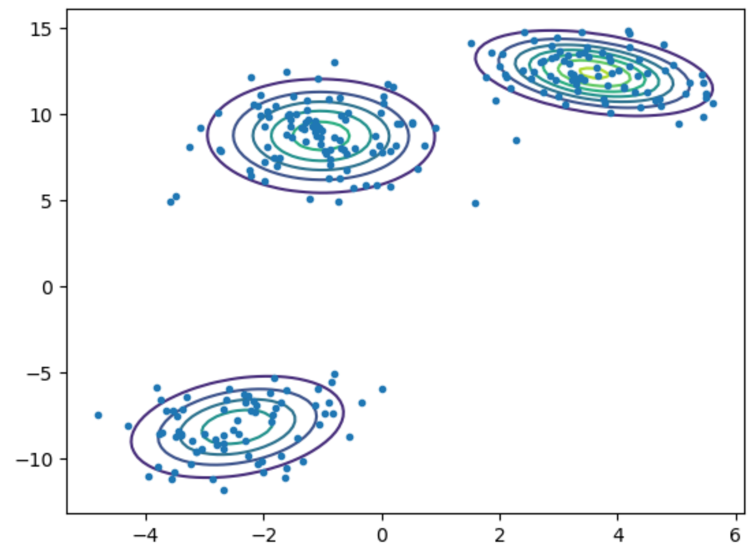
```
    function PDF(Xhat)
        (t,d) = size(Xhat)
        PDFs = zeros(t)

        for c in 1:k
            PDFs += pi[c]*subModel[c].pdf(Xhat)
        end
        return PDFs
    end

    return DensityModel(PDF)
end
```

Weighted Guassian function is the following



```julia
function weightedGaussianDensity(X, r)
    (n,d) = size(X)
    # MAP estimation
    lambda = 1e-4

    mu = (1/sum(r))sum(X .* r,dims=1)'
    Xc = X - repeat(mu',n)
    Sigma = (1/sum(r))*((Xc .* r)'Xc) + lambda * Matrix{Float64}(I, d, d)
    SigmaInv = Sigma^-1

    function PDF(Xhat)
        (t,d) = size(Xhat)
        PDFs = zeros(t)

        logZ = (d/2)log(2pi) + (1/2)logdet(Sigma)
        for i in 1:t
            xc = Xhat[i,:] - mu
            loglik = -(1/2)dot(xc,SigmaInv*xc) - logZ
            PDFs[i] = exp(loglik)
        end
        return PDFs
    end

    return DensityModel(PDF)
end
```

Updated plots are:



3 Very-Short Answer Questions

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

1. If we have n samples of d discrete features that can each take k values, what is the cost of computing the MLE if we use a product of independent distributions? (Try to find the minimum dependence on n, d, and k.)
    - For each feature, summing the count for k values (classes) take $O(n)$, then taking averages for k values take $O(k)$. We need to perform the same procedure to all d features.

- Thus the total runtime is $O(d(n+k))$, or $O(nd)$ if $k \leq n$

2. Describe a setting where it would make sense to use the general discrete distribution for density estimation with binary features.
    - We want to model dependency between features.
    - Finding the density of whether Vancouver rains or no over 5 days (so record rain/not rain for 5 days, which can be viewed as 5 featurs)
3. What is the relationship between using a product of independent Gaussians and using a multivariate Gaussian.
    - Product of independent Gaussian is the same as multivariate Gaussian with diagonal covariance matrix.
4. Suppose we run the graphical LASSO method and it returns a tri-diagonal precision matrix. What would the graph look like?
    - The first node will at most have one edge connecting to the second node, and the last node will at most have one edge connectin to the second to last node. Each node in between will at most connect to the node right in front an the node right in the back.
    - Thus the graph will look like a single chain.
5. 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?
    - Only one Gaussian: all the outliers will contribute to disrupt the only Gaussian.
    - Mixture of Gaussian: The disruption from outliers is somehow spread out. Also some Gaussians may be used to fit the outliers, so the rest of the Gausians can fit the rest of the data that is without outliers.
6. If we used GDA with where each class has its own diagonal covariance matrix, would this give linear or quadratic decision boundaries? What about if the covariance matrices are multiples of the identity matrix?
    - Own diagonal covariance: very likely to be quadratic
    - multiples of identity matrix: linear
7. What is an advantage and a disadvantage of the EM algorithm over the imputation approach to handling MAR variables
    - Advantage: unlike imputation, EM does not induce the propagation of error.
    - Disadvantage: longer runtime
8. What is an advantage of the Epanechnikov kernel over the Gaussian kernel.
    - much faster to use since it only depends on nearby points.
9. What is an advantage and a disadavantage of parameter tieing?
    - advantage: more data to train for each parameter
    - disadvantage: less flexible, could not find out if the data or pattern changes as time goes on.
10. What is the difference between computing marginals and computing the stationary distribution of a Markov chain.

- marginal: probability of states of specific time t. Can be computed using Chapman-Kolmogorov equation.
- stationary distribution: probability of states as time goes to infinity. May not even exist.

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 short 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

Dingqing Yang 38800141
Junjie Zhu 30921167

February 2019

1 Introduction

Deep learning has achieved great success in recent years and has made a lot of breakthroughs in AI applications. However, it suffers from some major drawbacks suck as requiring excessive computing power and a lack of interpretability. Another major problem is that optimizing hyperparameters for deep neural networks (DNNs) is a real headache. Exhaustive searching hyperparameters is extremely time-consuming and further exaggerates computation and memory requirements. Moreover, optimization bias is unavoidable when making too many attempts. Another annoying fact is that DNNs are usually very sensitive to some hyperparameters like learning rate. Therefore, fine-tuning DNNs requires expert knowledge to set the right hyperparameters and extra human labour to "babysit" the entire training procedures. AutoML [3] brings us the opportunity to overcome this bottleneck and even offers models that outperform human-designed ones.

2 Proposed Plan

Give that autoML is able to generate great model with a lot of hyperparameters, we would like to apply it to some new settings. We are interested in applying autoML for efficient DNN training. DNN training is computationally intensive and power hungry. Therefore, training is typically deployed in datacenter with clusters of GPUs. In order to reduce resource demand and energy consumption, some work such as Meprop [2] and Dropback [1] is proposed. Meprop and Dropback reduce computation cost and memory footprint respectively with negligible loss of accuracy (even better accuracy achieved in some model due to less overfitting). However, pruning scheme used in both algorithm are based on some heuristics and may not be optimal. We would like to use autoML to discover better pruning policy in order to achieve better compression ratio. Both algorithms introduce extra hyperparmaters, so this is a natural fit for autoML given its hyperparmater optimization ability. Finally, we would like to use autoML as a tool to help us understand the tradeoff between accuracy, energy, and storage etc. to guide us towards deploying optimal training scheme in different application domain.

References

[1] Maximilian Golub, Guy Lemieux, and Mieszko Lis. Dropback: Continuous pruning during training. *arXiv preprint arXiv:1806.06949*, 2018.

[2] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3299–3308. JMLR. org, 2017.

[3] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.