STAT 543/641 — Final Exam — Winter 2021

Problem 1 Solution

```
import torch
import numpy as np
from sklearn.cluster import KMeans

def g(x, mu, sigma_squared):
  return torch.exp(-0.5*(x - mu)**2/sigma_squared)/torch.sqrt(2*np.pi*sigma_squared)

class Model:
  """
  A 1-dimensional mixture of Gaussians.

  Attributes
  ----------
  pi_ : numpy.ndarray
      A numpy array of shape (n_components,).
      The weights of the mixture components.
  mu_ : numpy.ndarray
      A numpy array of shape (n_components,).
      The means of the mixture components.
  sigma_squared_ : numpy.ndarray
      A numpy array of shape (n_components,).
      The variances of the mixture components
  """
  def __init__(self, n_components=None):
    self.n_components = n_components

  def _initialize_with_kmeans(self, x):
    """
    Initialize parameters using K-means clustering.

    Parameters
    ----------
    x : numpy.ndarray
        The data.
    Returns
    -------
    eta_ : torch.tensor
        A pytorch tensor of shape (n_components,).
        Softmax-preimage of an initial estimate of the mixture component weights.
    mu_ : torch.tensor
        A pytorch tensor of shape (n_components,).
        Initial estimate of the means of the mixture components.
```

```
        sigma_squared_ : torch.tensor
            A pytorch tensor of shape (n_components,).
            Initial estimate of the variances of the mixture components.
        """
        K = self.n_components
        kmeans = KMeans(n_clusters=K)
        kmeans.fit(x.reshape(-1, 1))
        eta_ = np.log([np.sum(kmeans.labels_ == k) for k in range(K)])
        eta_ = torch.from_numpy(eta_)
        eta_.requires_grad_()
        mu_ = kmeans.cluster_centers_.squeeze()
        mu_ = torch.from_numpy(mu_)
        mu_.requires_grad_()
        sigma_squared_ = [np.var(x[kmeans.labels_ == k]) for k in range(K)]
        sigma_squared_ = torch.tensor(sigma_squared_)
        sigma_squared_.requires_grad_()
        return eta_, mu_, sigma_squared_

      def fit(self, x, lr=0.001, max_iter=100, rtol=1e-05, atol=1e-08):
        """
        Parameters
        ----------
        x : numpy.ndarray
            the data
        lr : float
            the learning rate
        max_iter : int
            stop training after max_iter iterations
        rtol : float
            relative tolerance for early stopping
        atol : float
            absolute tolerance for early stopping
        """
        eta_, mu_, sigma_squared_ = self._initialize_with_kmeans(x)

        x = torch.from_numpy(x.astype(np.float32)).reshape(-1, 1)
        x = torch.repeat_interleave(x, self.n_components, dim=1)

        for epoch in range(max_iter):
          prev_eta_ = eta_.detach().clone()
          prev_mu_ = mu_.detach().clone()
          prev_sigma_squared = sigma_squared_.detach().clone()

          y = torch.softmax(eta_, dim=-1)
          z = g(x, mu_, sigma_squared_)
          loss = -torch.mean(torch.log(torch.sum(y*z, axis=1)))
          loss.backward()

          with torch.no_grad():
            eta_ -= eta_.grad*lr
            mu_ -= mu_.grad*lr
            sigma_squared_ -= sigma_squared_.grad*lr
```

```
        eta_.grad.zero_()
        mu_.grad.zero_()
        sigma_squared_.grad.zero_()

        if (torch.allclose(prev_eta_, eta_, rtol=rtol, atol=atol) and
            torch.allclose(prev_mu_, mu_, rtol=rtol, atol=atol) and
            torch.allclose(prev_sigma_squared, sigma_squared_, rtol=rtol, atol=atol)):
          break;

    self.pi_ = torch.softmax(eta_, dim=0).detach().numpy()
    self.mu_ = mu_.detach().numpy()
    self.sigma_squared_= sigma_squared_.detach().numpy()

np.random.seed(42)
n0, m0, s0 = 20, -1, 0.20
n1, m1, s1 = 10, 0, 0.15
n2, m2, s2 = 25, 1, 0.35

x0 = np.random.normal(m0, s0, size=n0)
x1 = np.random.normal(m1, s1, size=n1)
x2 = np.random.normal(m2, s2, size=n2)
x = np.concatenate([x0, x1, x2])
np.random.shuffle(x)

model = Model(n_components=3)
model.fit(x, max_iter=1000)
print(model.pi_)
print(model.mu_)
print(np.sqrt(model.sigma_squared_))
```

```
[0.36267454 0.26533611 0.37198936]
[-1.03456067  0.07143523  1.00006633]
[0.18700589 0.21693756 0.29161923]
```

Problem 2 Solution

a)

Let $g(x \mid \mu_k, \Sigma)$ be the density of the Gaussian distribribution with mean μ_k and covariance matrix Σ. By Bayes' Theorem and the fact that

$$\mathbb{P}[Y = 0] = \frac{1}{2} = \mathbb{P}[Y = 1],$$

we have

$$\mathbb{P}[Y = 0 \mid X = x] \propto g(x \mid \mu_0, \Sigma)$$

and

$$\mathbb{P}[Y = 1 \mid X = x] \propto g(x \mid \mu_1, \Sigma),$$

with the same constant of proportionality. Therefore,

$$\mathbb{P}[Y = 0 \mid X = x] = \mathbb{P}[Y = 1 \mid X = x]$$

if and only if:

$$\begin{aligned}
g(x \mid \mu_0, \Sigma) &= g(x \mid \mu_1, \Sigma) \\
\frac{\exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)}{\sqrt{(2\pi)^p \det \Sigma}} &= \frac{\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)}{\sqrt{(2\pi)^p \det \Sigma}} \\
\exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) &= \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right) \\
-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) &= -\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) \\
(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) &= (x-\mu_1)^T\Sigma^{-1}(x-\mu_1) \\
x^T\Sigma^{-1}x - 2x^T\Sigma^{-1}\mu_0 + \mu_0^T\Sigma^{-1}\mu_0 &= x^T\Sigma^{-1}x - 2x^T\Sigma^{-1}\mu_1 + \mu_1^T\Sigma^{-1}\mu_1 \\
2x^T\Sigma^{-1}(\mu_1 - \mu_0) &= \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0
\end{aligned}$$

The set of $x \in \mathbb{R}^p$ satisfying the equation

$$2x^T\Sigma^{-1}(\mu_1 - \mu_0) = \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0 \tag{$*$}$$

is a hyperplane perpendicular to $\Sigma^{-1}(\mu_1 - \mu_0)$. By the symmetry of Σ,

$$(\mu_1 - \mu_0)^T\Sigma^{-1}(\mu_0 + \mu_1) = \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0.$$

Thus, the plane $(*)$ passes through $\frac{1}{2}(\mu_0 + \mu_1)$.

b)

The log-likelihood function associated to $(x_1, y_1), \ldots, (x_n, y_n)$ is:

$$\begin{aligned}
\ell(\mu_0, \mu_1, \Sigma) &= \sum_i \log p(x_i, y_i) \\
&= \sum_i \log p(x_i \mid y_i)p(y_i) \\
&= \sum_i \log p(x_i \mid y_i) + n \log \frac{1}{2} \\
&= \sum_{i:y_i=0} \log g(x \mid \mu_0, \Sigma) + \sum_{i:y_i=1} \log g(x \mid \mu_1, \Sigma) + n \log \frac{1}{2} \\
&= -\frac{pn}{2}\log 2\pi - \frac{n}{2}\log \Sigma - \frac{1}{2}\sum_{i:y_i=0}(x_i - \mu_0)^T\Sigma^{-1}(x_i - \mu_0) - \frac{1}{2}\sum_{i:y_i=1}(x_i - \mu_1)^T
\end{aligned}$$

Differentiate:

$$\frac{\partial \ell}{\partial \mu_k} = \frac{1}{2} \sum_{i:y_i=k} (\Sigma^{-1}\mu_k - \Sigma^{-1}x_i),$$
$$\frac{\partial \ell}{\partial \Sigma} = -\frac{n}{2}\Sigma^{-1} + \frac{1}{2}\sum_{i:y_i=0} \Sigma^{-1}(x_i - \mu_0)(x_i - \mu_0)^T\Sigma^{-1} + \frac{1}{2}\sum_{i:y_i=1} \Sigma^{-1}(x_i - \mu_1)(x_i - \mu_1)^T\Sigma^{-1}$$

(Here, I used the differentiation formulas you proved on Assignment 1.)

It follows that

$$\frac{\partial \ell}{\partial \mu_k} = 0 \Longleftrightarrow \mu_k = \frac{1}{n_k}\sum_{i:y_i=k} x_i, \qquad n_k := \sum_{i:y_i=k} 1,$$

Problem 3 Solution

For $\alpha \geq 0$, let

$$\beta_\alpha \in \mathbb{R}^{p\times 1}$$

be the vector of ridge (equivalently, L^2) regression coefficients fit, without intercept term, to a data set

$$(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^{1\times p} \times \mathbb{R}.$$

a)

By definition, the least squares solution to $Ax = b$ is the minimizer of $\|Ax - b\|^2$.

We have:

$$\begin{aligned}
\left\| \begin{bmatrix} X \\ \sqrt{\alpha} I \end{bmatrix} \beta - \begin{bmatrix} Y \\ 0 \end{bmatrix} \right\|^2 &= \left\| \begin{bmatrix} X\beta - Y \\ \sqrt{\alpha} I\beta - 0 \end{bmatrix} \right\|^2 \\
&= \left\| \begin{bmatrix} X\beta - Y \\ \sqrt{\alpha}\beta \end{bmatrix} \right\|^2 \\
&= \|X\beta - Y\|^2 + \|\sqrt{\alpha}\beta\|^2 \\
&= \|X\beta - Y\|^2 + \alpha\|\beta\|^2
\end{aligned}$$

This final expression is precisely the objective function from a).

b)

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from matplotlib import pyplot as plt

def make_dataset():
  n = 200; n_tr = 100; d = 20; s = 0.1; np.random.seed(42)
  x = np.random.uniform(size=n)
  P = PolynomialFeatures(degree=d, include_bias=False)
```

```
  X = P.fit_transform(x.reshape(-1, 1))
  beta_true = np.random.normal(size=d)
  y = X.dot(beta_true) + np.random.normal(0, s, size=n)
  return X[:n_tr], X[n_tr:], y[:n_tr], y[n_tr:]

X_tr, X_te, y_tr, y_te = make_dataset()

I = np.eye(20)
alphas = np.arange(0, 0.5, 0.01)
errs = []
for alpha in alphas:
  X = np.vstack([X_tr, np.sqrt(alpha)*I])
  y = np.concatenate([y_tr, np.zeros(20)])
  beta_alpha = np.linalg.lstsq(X, y, rcond=None)[0]
  err = np.mean(np.square(X_te.dot(beta_alpha) - y_te))
  errs.append(err)

alpha = alphas[np.argmin(errs)]
print(f"alpha = {alpha}")
```

```
alpha = 0.27
```

Problem 4 Solution

SHOW CODE

SHOW HIDDEN OUTPUT

```
import torch
import requests
import numpy as np
from PIL import Image
from io import BytesIO
from torchvision.models import vgg19

def get_image():
  URL = "https://upload.wikimedia.org/wikipedia/commons/4/44/Jelly_cc11.jpg"
  response = requests.get(URL)
  img = Image.open(BytesIO(response.content)).resize((224, 224))
  x = np.array(img)
  x = np.moveaxis(x, -1, 0)
  x = torch.from_numpy(x)/255
  return img, x

img, x = get_image()

model = vgg19(pretrained=True)
print(model)
```

SHOW HIDDEN OUTPUT

```
y = torch.squeeze(model(x.reshape(-1, *x.shape))).detach()
I = torch.argsort(y, descending=True)
print("classes: ", [classes[I[i]] for i in range(5)])
print("probabilities: ", torch.softmax(y[I], dim=0)[:5])
```

```
classes:  ['jellyfish', 'balloon', 'parachute', 'conch', 'chambered nautilus']
probabilities:  tensor([0.9052, 0.0415, 0.0150, 0.0148, 0.0078])
```