SOLUTION
Math 189r
Homework 3
November 7, 2016

There are 5 problems in this set. You need to do 3 problems the first week and 2 the second week. Instead of a sixth problem, youare encouraged to work on your final project. Feel free to work with other students, but make sure you write up the homework and code on your own (no copying homework *or* code; no pair programming). Feel free to ask students or instructors for help debugging code or whatever else, though. When implementing algorithms you may not use any library (such as `sklearn`) that already implements the algorithms but you may use any other library for data cleaning and numeric purposes (`numpy` or `pandas`). Use common sense. Problems are in no specific order.

1 (Murphy 11.2 - EM for Mixtures of Gaussians) Show that the M step for ML estimation of a mixture of Gaussians is given by

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k}$$
$$\boldsymbol{\Sigma}_k = \frac{1}{r_k}\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top = \frac{1}{r_k}\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^\top - r_k\boldsymbol{\mu}_k\boldsymbol{\mu}_k^\top.$$

We have the complete data log likelihood

$$\ell(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_k\sum_i r_{ik}\log\mathbb{P}(\mathbf{x}_i|\boldsymbol{\theta}_k) \tag{1}$$
$$= -\frac{1}{2}\sum_i r_{ik}\left(\log|\boldsymbol{\Sigma}_k| + (\mathbf{x}_i - \boldsymbol{\mu}_k)^\top\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)\right). \qquad \text{(to a constant)}$$

Then differentiating with respect to $\boldsymbol{\mu}_k$ we have

$$\frac{\partial\ell}{\partial\boldsymbol{\mu}_k} = \sum_i r_{ik}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) \tag{2}$$
$$= \boldsymbol{\Sigma}_k^{-1}\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k) = 0 \qquad \text{(since } \boldsymbol{\Sigma}_k^{-1} \text{ is linear)}$$

so at optimality we have

$$\sum_i r_{ik}\mathbf{x}_i = \boldsymbol{\mu}_k\sum_i r_{ik}, \tag{3}$$

which gives the desired result. Differentiating with respect to $\boldsymbol{\Sigma}_k$ we have[1]

$$\frac{\partial\ell}{\partial\boldsymbol{\Sigma}_k} = -\frac{1}{2}\sum_i r_{ik}\left(\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top\boldsymbol{\Sigma}_k^{-1}\right) = 0. \tag{4}$$

[1] `http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf`

This gives us the optimality condition that

$$\sum_i r_{ik} I = \left(\sum_i r_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \right) \boldsymbol{\Sigma}_k^{-1}. \tag{5}$$

Multiplying by $\boldsymbol{\Sigma}_k$ on the right and dividing by $r_k = \sum_i r_{ik}$ gives the desired result.

---

2 (Murphy 11.3 - EM for Mixtures of Bernoullis) Show that the M step for ML estimation of a mixture of Bernoullis is given by

$$\mu_{kj} = \frac{\sum_i r_{ik} x_{ij}}{\sum_i r_{ik}}.$$

Show that the M step for MAP estimation of a mixture of Bernoullis with a $\beta(\alpha, \beta)$ prior is given by

$$\mu_{kj} = \frac{\left(\sum_i r_{ik} x_{ij}\right) + \alpha - 1}{\left(\sum_i r_{ik}\right) + \alpha + \beta - 2}.$$

(a) We have the complete data log likelihood

$$\ell(\boldsymbol{\mu}) = \sum_i \sum_k r_{ik} \log \mathbb{P}(\mathbf{x}_i | \boldsymbol{\theta}_k) \tag{6}$$

$$= \sum_i \sum_k r_{ik} \sum_j \mathbf{x}_{ij} \log \boldsymbol{\mu}_{kj} + (1 - \mathbf{x}_{ij}) \log(1 - \boldsymbol{\mu}_{kj}) \tag{7}$$

where i is the datapoint index, k is the component, and j is the dimension index of the D dimensional bit vectors. Taking the derivative with respect to $\boldsymbol{\mu}_{kj}$ we have

$$\frac{\partial \ell}{\partial \boldsymbol{\mu}_{kj}} = \sum_i r_{ik} \left(\frac{\mathbf{x}_{ij}}{\boldsymbol{\mu}_{kj}} - \frac{1 - \mathbf{x}_{ij}}{1 - \boldsymbol{\mu}_{kj}} \right) \tag{8}$$

$$= \sum_i r_{ik} \left(\frac{\mathbf{x}_{ij} - \boldsymbol{\mu}_{kj}}{\boldsymbol{\mu}_{kj}(1 - \boldsymbol{\mu}_{kj})} \right) \tag{9}$$

$$= \frac{1}{\boldsymbol{\mu}_{kj}(1 - \boldsymbol{\mu}_{kj})} \sum_i r_{ik} \left(\mathbf{x}_{ij} - \boldsymbol{\mu}_{kj} \right) = 0. \tag{10}$$

This gives the optimality condition

$$\sum_i r_{ik} \mathbf{x}_{ij} = \boldsymbol{\mu}_{kj} \sum_i r_{ik} \tag{11}$$

which gives the desired result.

(b) We have the complete data log likelihood plus the log prior (ignoring the $\boldsymbol{\pi}$ terms as we are maximizing without regard to them)

$$\ell(\boldsymbol{\mu}) = \sum_i \sum_k r_{ik} \log \mathbb{P}(\mathbf{x}_i|\boldsymbol{\mu}_k) + \log \mathbb{P}(\boldsymbol{\mu}_k) \tag{12}$$

$$= \sum_i \sum_k r_{ik} \left(\sum_j \mathbf{x}_{ij} \log \boldsymbol{\mu}_{kj} + (1 - \mathbf{x}_{ij}) \log(1 - \boldsymbol{\mu}_{kj}) \right) + (\alpha - 1) \log \boldsymbol{\mu}_{kj} + (\beta - 1) \log(1 - \boldsymbol{\mu}_{kj}). \tag{13}$$

Taking derivatives we have

$$\frac{\partial \ell}{\partial \boldsymbol{\mu}} = \sum_i \left(\frac{r_{ik}\mathbf{x}_{ij} + \alpha - 1}{\boldsymbol{\mu}_{kj}} - \frac{r_{ik}(1 - \mathbf{x}_{ij}) + \beta - 1}{1 - \boldsymbol{\mu}_{kj}} \right) \tag{14}$$

$$= \frac{1}{\boldsymbol{\mu}_{kj}(1 - \boldsymbol{\mu}_{kj})} \sum_i r_{ik}\mathbf{x}_{ij} - r_{ik}\boldsymbol{\mu}_{kj} + \alpha - 1 - \boldsymbol{\mu}_{kj}\alpha + \boldsymbol{\mu}_{kj} - \boldsymbol{\mu}_{kj}\beta + \boldsymbol{\mu}_{kj} \tag{15}$$

$$= \frac{1}{\boldsymbol{\mu}_{kj}(1 - \boldsymbol{\mu}_{kj})} \left[\sum_i r_{ik}\mathbf{x}_{ij} - \left(\sum_i r_{ik} + \alpha + \beta - 2 \right) \boldsymbol{\mu}_{kj} + \alpha - 1 \right] = 0. \tag{16}$$

This gives the optimality condition

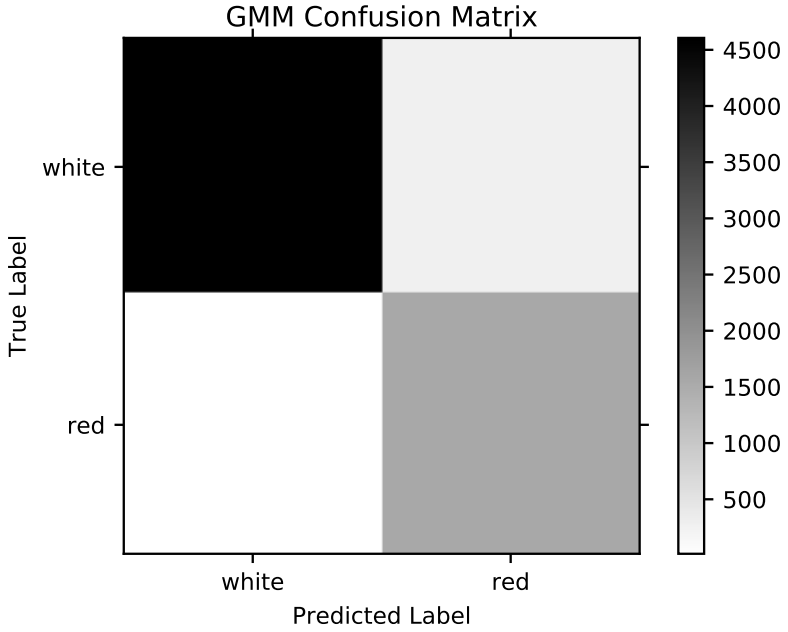
$$\sum_i r_{ik}\mathbf{x}_{ij} + \alpha - 1 = \left(\sum_i r_{ik} + \alpha + \beta - 2 \right) \boldsymbol{\mu}_{kj}, \tag{17}$$

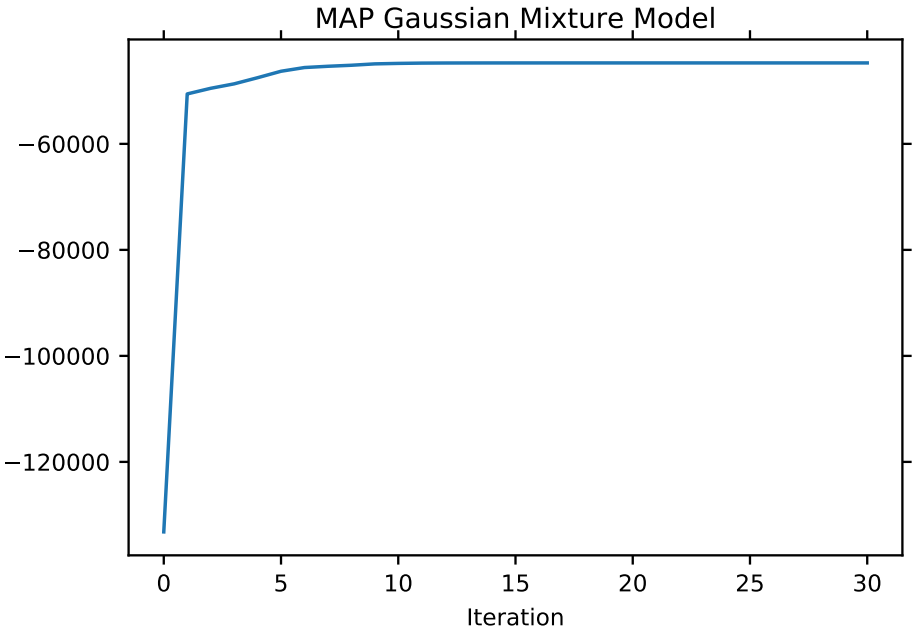
which gives the desired result. Note that if $\alpha = \beta = 1$ we arrive at the original maximum likelihood estimate, which makes sense since $\beta(1, 1)$ is a uniform distribution over $[0, 1]$ so it is as if there is no prior at all.

---

3 (MAP Mixture of Gaussians) Consider a mixture of Gaussians with a Dirichlet prior on the mixture weights $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$ and a Negative Inverse Walshart prior on the mean and covariance within each class $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \sim \text{NIW}(\mathbf{m}_0, \kappa_0, \nu_0, \mathbf{S}_0)$ with $\kappa_0 = 0$ so only the covariance matrices are regularized. Use $\mathbf{S}_0 = \text{diag}(s_1^2, \ldots, s_D^2)/K^{1/D}$ where $s_j = \sum_i (x_{ij} - \overline{x}_j)^2/N$ is the pooled variance in dimension j. Use $\nu_0 = D + 2$, as that is the weakest prior that is still proper. Use $\boldsymbol{\alpha} = \mathbf{1}$. This is all detailed in Murphy 11.4.2.8. Download the wine quality data at `https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv` and `https://archive.ics.uci.edu/ml/machine-learning-data` `wine-quality/winequality-white.csv`. Pool both red and white wine datasets into one dataset and cluster this data using a 2 component MAP Gaussian mixture model with the EM algorithm. Do the clusters roughly correspond to the color of the wine {white,red} (back this with numbers)? Provide a convergence plot of the MAP objective. If it doesn't monotonically increase there is a bug in your code or math.

Code is below. Note the convergence plot below is monotonically increasing, so we can be more sure that our implementation is correct. Also note that the cluster assigned for the points is highly correlated with being either white or red wine, so the clusters do roughly correspond to the wine color!






```
from collections import namedtuple

import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd
from scipy import stats
from sklearn import metrics

plt.style.use("default")

def em(X, k, theta, objective,
       likelihood, m_step, max_iter=100,
       print_freq=10):
    """ em
    EM algorithm to compute theta which minimizes objective.

    :param X: data matrix
    :type X: array-like of shape [datapoints, features]
    :param k: number of mixture components
    :type k: int
    :param objective: objective (MLE or MAP) for the problem
    :type objective: func[X, r, pi, theta] -> float
    :param likelihood: (vectorized) P(x_i | theta)
    :type likelihood: func[X, theta] -> np.ndarray[X.shape[0],k]
    :param m_step: take in r_ik and data and compute the new theta
    :type m_step: func[X, r] -> pi, theta
    :param max_iter: max number of iterations to take
    :type max_iter: int

    :returns: list of all objectives for each iteration, r, pi, and the final theta
    :rtype: (list[float], np.ndarray, np.ndarray, tuple)
    """
    r = np.ones((X.shape[0], k)) / k
    pi = np.ones(k) / k
    objectives = [objective(X, r, pi, theta)]

    for i in range(max_iter):
        if (i % print_freq) == 0:
            print("[i={}] objective={}".format(i, objectives[-1]))
        # e-step
        r = likelihood(X, theta) * pi
        r = r / r.sum(axis=1)[:,np.newaxis]

        # m-step
        pi, theta = m_step(X, r)

        objectives.append(objective(X, r, pi, theta))

    return (objectives, r, pi, theta)
```

```
def gmm(X, k, prior_alpha, max_iter=100, print_freq=10):
    """ gmm
    Gaussian mixture model with priors.

    :param X: data matrix
    :type X: array-like of shape [datapoints, features]
    :param k: number of mixture components
    :type k: int
    :param prior_alpha: Dirichlet prior alpha for pi
    :type prior_alpha: np.ndarray[k]
    :param max_iter: max number of iterations to take
    :type max_iter: int
    """
    S_0 = np.diag(np.std(X, axis=0)**2) / k**(1/X.shape[1])
    Theta = namedtuple("GMM", "mean cov")
    theta = Theta(
        mean=X[np.random.randint(0, X.shape[0], (k,))],
        cov=np.tile(S_0, (k,1,1)),
    )

    def likelihood(X, theta):
        p = np.zeros((X.shape[0], k))
        for i in range(k):
            p[:,i] = stats.multivariate_normal.pdf(
                X, theta.mean[i], theta.cov[i] + 1e-4*np.eye(X.shape[1]),
            )
        return p

    denominator = X.shape[0] + prior_alpha.sum() - k
    prior_nu = X.shape[1] + 2
    def m_step(X, r):
        r_sum = r.sum(axis=0)
        pi = (r_sum + prior_alpha - 1) / denominator
        mu = ((X[:,:,np.newaxis] * r[:,np.newaxis,:]).sum(axis=0) / r_sum).T
        sigma = np.zeros((k, X.shape[1], X.shape[1]))
        for i in range(k):
            diff = (X - mu[i]) * np.sqrt(r[:,i])[:,np.newaxis]
            sigma[i] = (diff.T @ diff + S_0) / (prior_nu + r_sum[i] + X.shape[1] + 2)

        return pi, Theta(mean=mu, cov=sigma)

    def objective(X, r, pi, theta):
        log_prior = sum(np.log(stats.invwishart.pdf(
               theta.cov[i], df=prior_nu, scale=S_0,
```

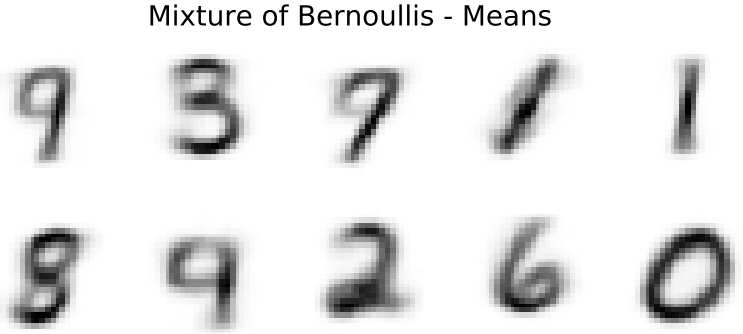
```
            )) for i in range(k)
        ) + np.log(stats.dirichlet.pdf(pi, alpha=prior_alpha))
        pi_term = (r * np.log(pi)[np.newaxis,:]).sum()
        likelihood_term = r * np.log(likelihood(X, theta))
        likelihood_term = likelihood_term[r > 1e-12].sum()
        return likelihood_term + pi_term + log_prior

    return em(
        X, k, theta, objective,
        likelihood, m_step, max_iter=max_iter,
        print_freq=print_freq,
    )


#### RUN ON DATA ####

red = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-qualit
white = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-qual

X = np.concatenate((red.as_matrix(), white.as_matrix()), axis=0)
y = np.zeros((X.shape[0],))
y[:red.shape[0]] = 1

k = 2
obj, r, pi, theta = gmm(
    X, k, np.ones(2), max_iter=30, print_freq=10,
)

plt.plot(obj)
plt.title("MAP Gaussian Mixture Model")
plt.xlabel("Iteration")
plt.ylabel("Complete Data Log Likelihood")
plt.savefig("nov_7/gmm_convergence.pdf")

y_pred = np.argmax(r,axis=1)
plt.imshow(metrics.confusion_matrix(
    y, ~y_pred.astype(bool), # cluster assignments are arbitrary hence the flip
), cmap=plt.cm.gray_r)
plt.title("GMM Confusion Matrix")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.colorbar()

ticks = np.arange(2)
classes = ["white", "red"]
```

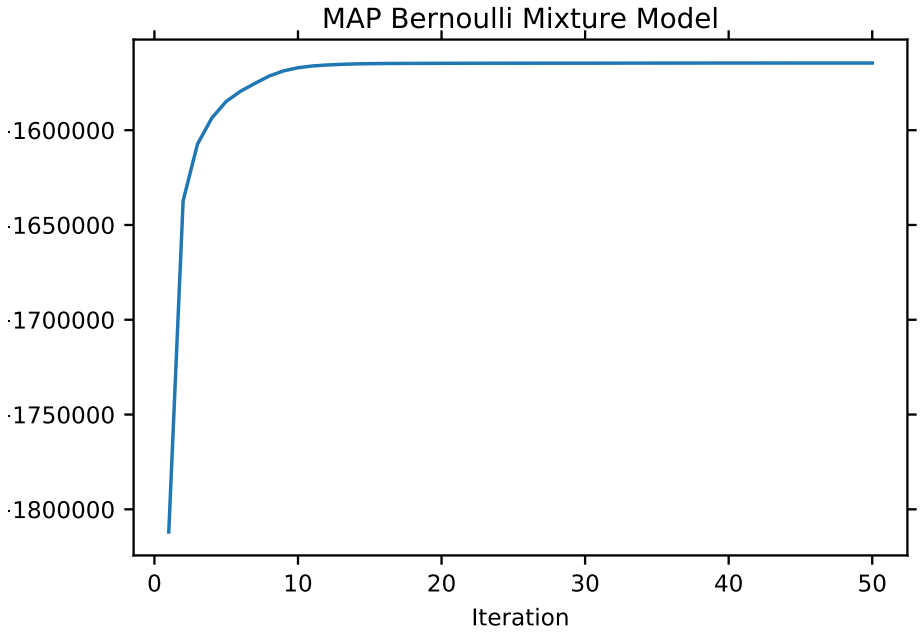
```
plt.xticks(ticks, classes)
plt.yticks(ticks, classes)
plt.savefig("nov_7/3_confusion_matrix.pdf")
```

---

4 (MAP Mixture of Bernoullis) Consider a mixture of Bernoullis with a Dirichlet prior on the mixture weights $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$ and a Beta prior on the mean parameter $\mu_{kj} \sim \beta(\alpha, \beta)$. Use $\boldsymbol{\alpha} = \mathbf{1}$ and choose an appropriate (α, β) pair for your prior (back this up). Note that the M step for the mean is given in problem 2 (Murphy 11.3). Cluster the MNIST training dataset we used from homework 1 (`http://pjreddie.com/media/files/mnist_train.csv`) using this mixture with 10 components. Provide a convergence plot of the MAP objective (which must monotonically increase) and plot the mean images for each component. Do the clusters roughly correspond to different digits (back this up somehow)?

Code is below. Note that the complete data log likelihood monotonically increases, so we can be more sure that our implementation is correct. We chose a $\beta(1, 1)$ prior on each mean μ_{kj} since it is the weakest proper prior (in fact, it is a uniform distribution - ! - and hence corresponds exactly to the MLE estimate). Look at our plot of the mean parameters below. We can see that the means roughly correspond to the digits, but that some digits are morphed into others that look similar to account for variation within other digits (for example, the digit 9). Also note that we used the `em()` function and imports defined in problem 3 since the implementations are so similar.






```
def bernoullis(
    X, k, prior_alpha, prior_a, prior_b,
    max_iter=100, print_freq=10,
):
    """ bernoullis
    Bernoulli mixture model with priors.

    :param X: data matrix
    :type X: array-like of shape [datapoints, features]
    :param k: number of mixture components
    :type k: int
    :param prior_alpha: Dirichlet prior alpha for pi
    :type prior_alpha: np.ndarray[k]
    :param prior_a: a in the mu ~ Beta(a,b) prior
    :type prior_a: float
    :param prior_b: b in the mu ~ Beta(a,b) prior
    :type prior_b: float
    :param max_iter: max number of iterations to take
    :type max_iter: int
    """
    S_0 = np.diag(np.std(X, axis=0)**2) / k**(1/X.shape[1])
    Theta = namedtuple("BMM", "mean")

    # compute initial means by partitioning data arbitrarily
    theta = Theta(
        mean=X[:k*np.floor(X.shape[0]/k)].reshape(
            k, -1, X.shape[1],
```

```
        ).mean(axis=1),
    )

    def likelihood(X, theta):
        p = np.tile(theta.mean.T, (X.shape[0],1,1))
        p[X == 0] = 1 - p[X == 0]
        p = p.prod(axis=1)
        return p

    denominator = X.shape[0] + prior_alpha.sum() - k
    def m_step(X, r):
        r_sum = r.sum(axis=0)
        pi = (r_sum + prior_alpha - 1) / denominator
        mu = (
            ((X[:,:,np.newaxis] * r[:,np.newaxis,:]).sum(axis=0) + prior_a - 1) /
            (r_sum + prior_a + prior_b - 2)
        ).T
        return pi, Theta(mean=mu)

    def objective(X, r, pi, theta):
        log_prior = np.log(stats.beta.pdf(
                theta.mean, prior_a, prior_b,
            )).sum() + np.log(
                stats.dirichlet.pdf(pi, alpha=prior_alpha),
        )
        pi_term = (r * np.log(pi)[np.newaxis,:]).sum()
        likelihood_term = r * np.log(likelihood(X, theta))
        likelihood_term = likelihood_term[r > 1e-12].sum()
        return likelihood_term + pi_term + log_prior

    return em(
        X, k, theta, objective,
        likelihood, m_step, max_iter=max_iter,
        print_freq=print_freq,
    )

def plot_image(img):
    plt.imshow(img.reshape(28,28), cmap="Greys")
    plt.axis("off")


### RUN ON DATA ###

train = pd.read_csv("http://pjreddie.com/media/files/mnist_train.csv", header=None)
X = train.iloc[:,1:].as_matrix()
```

```
X = (X > X[X > 0].mean()).astype(float)
y = train.iloc[:,0].as_matrix()
del train

# downsample data by 1/6 to run a bit faster
np.random.seed(1)
N = int(10000)
subset_ix = np.random.randint(0,X.shape[0],(N,))
X_downsample = X[subset_ix]

k = 10
obj, r, pi, theta = bernoullis(
    X_downsample, k, prior_alpha=np.ones(10),
    prior_a=1, prior_b=1,
    max_iter=50, print_freq=10,
)

plt.plot(obj)
plt.title("MAP Bernoulli Mixture Model")
plt.xlabel("Iteration")
plt.ylabel("Complete Data Log Likelihood")
plt.savefig("nov_7/bernoulli_convergence.pdf")

plt.figure(figsize=(5,2))
for i in range(10):
    plt.subplot(2,5,i+1)
    plot_image(theta.mean[i])

plt.suptitle("Mixture of Bernoullis - Means")
plt.savefig("nov_7/bernoulli_means.pdf")
```

---

5 (Operations Preserving Kernels) Let $\kappa(\cdot,\cdot)$ and $\lambda(\cdot,\cdot)$ be valid positive semi-definite (Mercer) kernels mapping from a sample space $\mathcal{S}$ to $\mathbb{R}$. Let $\alpha \geq 0$ be a real number and let x and y be elements of $\mathcal{S}$. Prove that

(a) $\alpha\kappa(x,y)$ is a valid kernel.

(b) $\kappa(x,y) + \lambda(x,y)$ is a valid kernel.

(c) $\kappa(x,y)\lambda(x,y)$ is a valid kernel. *Hint:* consider the Cholesky decomposition of the corresponding covariance matrix generated by the product of the kernels.

(d) $p(\kappa(x,y))$ is a valid kernel where $p(\cdot)$ is a polynomial with non-negative coefficients.

(e) $\exp(\kappa(x,y))$ is a valid kernel.

(f) $f(x)\kappa(x,y)f(y)$ for all $f : \mathcal{S} \to \mathbb{R}$.

To prove these, the main method would be to consider an arbitrary covariance matrix generated by these kernels and assert that the conditions on this covariance matrix (being symmetric and positive semi-definite) still hold.

(a) Let $\gamma(x,y) = \alpha\kappa(x,y)$. Now consider arbitrary covariance matrices Γ and $\mathbf{K}$ generated by the same dataset X such that $\Gamma_{ij} = \gamma(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Then $\Gamma = \alpha\mathbf{K}$ and clearly Γ is symmetric. Now consider an arbitrary vector $\mathbf{v}$ of suitable dimension. Then $\mathbf{v}^\top\Gamma\mathbf{v} = \mathbf{v}^\top\alpha\mathbf{K}\mathbf{v} = \alpha\mathbf{v}^\top\mathbf{K}\mathbf{v} \geq 0$ if $\alpha \geq 0$ since $\mathbf{K}$ is positive semidefinite. It follows that Γ is symmetric and positive definite for any dataset X, so γ must be a valid kernel.

(b) Let $\gamma(x,y) = \kappa(x,y) + \lambda(x,y)$. As above, let Γ, $\mathbf{K}$, and Λ be covariance matrices generated by these kernels. Then $\Gamma = \mathbf{K} + \Lambda$. Clearly Γ is symmetric. Consider some arbitrary $\mathbf{v}$ of suitable dimension. Then $\mathbf{v}^\top\Gamma\mathbf{v} = \mathbf{v}^\top(\mathbf{K}+\Lambda)\mathbf{v} = \mathbf{v}^\top\mathbf{K}\mathbf{v} + \mathbf{v}^\top\Lambda\mathbf{v} \geq 0$ since $\mathbf{K}$ and Λ are both positive semidefinite. Therefore Γ is symmetric and positive semidefinite for an arbitrary dataset X, so γ must be a valid kernel.

(c) Let $\gamma(x,y) = \kappa(x,y)\lambda(x,y)$. As above, let Γ, $\mathbf{K}$, and Λ be covariance matrices generated by these kernels. Then $\Gamma_{ij} = \mathbf{K}_{ij}\Lambda_{ij}$. Since κ and λ are valid kernels, we know that $\mathbf{K}$ and Λ are positive semi-definite so they must have Cholesky decompositions $\mathbf{K}_{ij} = \mathbf{k}_i^\top\mathbf{k}_j$ and $\Lambda_{ij} = \boldsymbol{\lambda}_i^\top\boldsymbol{\lambda}_j$. Consider an arbitrary vector $\mathbf{v}$ of suitable dimension. Then $\mathbf{v}^\top\Gamma\mathbf{v} = \sum_{ij}\mathbf{v}_i\mathbf{v}_j\Gamma_{ij} = \sum_{ij}\mathbf{v}_i\mathbf{v}_j\mathbf{k}_i^\top\mathbf{k}_j\boldsymbol{\lambda}_j^\top\boldsymbol{\lambda}_i = \sum_i\mathbf{v}_i\mathbf{k}_i^\top\boldsymbol{\lambda}_i\sum_j\mathbf{v}_i\boldsymbol{\lambda}_j^\top\mathbf{k} = \left(\sum_i\mathbf{v}_i\mathbf{k}_i^\top\boldsymbol{\lambda}_i\right)^2 \geq 0$ so Γ must be positive semi-definite. Since Γ is symmetric as well, all for an arbitrary dataset X, we know γ must be a valid kernel.

(d) This is a direct corollary of (a), (b), and (c).

(e) This is a direct corollary of (d) since $\exp(x) = \sum_{i=1}^{\infty}\frac{x^n}{n!}$, a polynomial with non-negative coefficients.

(f) Let $\gamma(x,y) = f(x)\kappa(x,y)f(y)$. As above, let Γ be an arbitrary covariance matrix generated from some dataset X such that $\Gamma_{ij} = \gamma(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i)\kappa(\mathbf{x}_i, \mathbf{x}_j)f(\mathbf{x}_j)$. Clearly Γ is symmetric. Consider an arbitrary vector $\mathbf{v}$ of suitable dimension and another vector $\mathbf{u}$ such that $\mathbf{u}_i = \mathbf{v}_i f(\mathbf{x}_i)$. Then $\mathbf{v}^\top\Gamma\mathbf{v} = \sum_{ij}\mathbf{v}_i\mathbf{v}_j f(\mathbf{x}_i)\kappa(\mathbf{x}_i,\mathbf{x}_j)f(\mathbf{x}_j) = \sum_{ij}\left[\mathbf{v}_i f(\mathbf{x}_i)\right]\left[\mathbf{v}_j f(\mathbf{x}_j)\right]\kappa(\mathbf{x}_i,\mathbf{x}_j) = \sum_{ij}\mathbf{u}_i\mathbf{u}_j\kappa(\mathbf{x}_i,\mathbf{x}_j) \geq 0$ since κ is positive definite. Then Γ is symmetric and positive semi-definite for an arbitrary covariance matrix X, so γ must be a valid kernel. Then $\mathbf{v}^\top$

---

6 (Totally Optional Extra Credit) Prove (a), (b), and (c) from problem 5 (above) for the special case of stationary kernels $\kappa(x,y) = \kappa(x-y)$ using Bochner's Theorem and properties of Fourier transforms. Note that for (c) the Convolution Theorem might be helpful.