CS281A - Problem Set 2

Andrea Bajcsy

September 19, 2016

Problem 2.1.

(a) We can formulate the polynomial regression problem as a form of linear prediction by soliving the general linear model equation $X\alpha = y$ where:

$$X = \begin{bmatrix} 1 & t_1 & t_1^2 & ... & t_1^D \\ 1 & t_2 & t_2^2 & ... & t_2^D \\ 1 & t_3 & t_3^2 & ... & t_3^D \\ ... & & & & \\ 1 & t_n & t_n^2 & ... & t_n^D \end{bmatrix} \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ ... \\ \alpha_D \end{bmatrix} y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ ... \\ y_n \end{bmatrix}$$

(b) Figure 1 shows a plot of the mean-squared error R(D) vs. Degree $D \in 1, 2, ...n - 1$ when using the data in y.dat and t.dat. See back for code that performs least-squares fit of a polynomial of degree D.

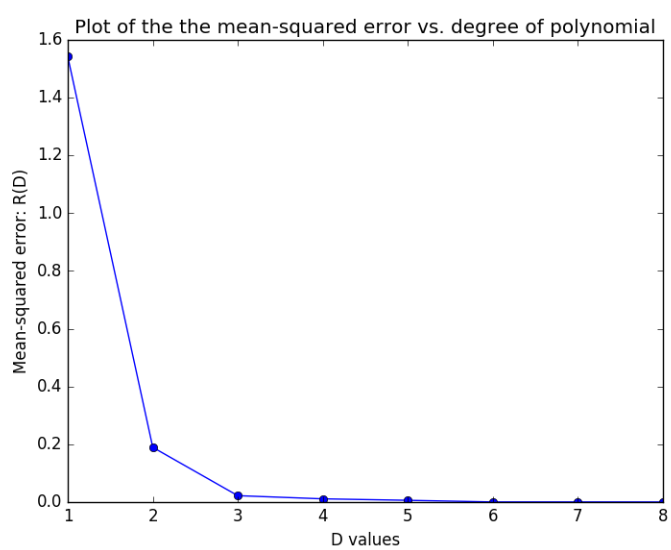

Figure 1: D vs. R(D)

(c) How does the MSE behave as a function of D and why? With the degree n-1 fit, we get (approximately) zero mean-squared error since the function fits exactly to every data point. What happens if you try to fit a polynomial of degree n? Why? To fit a polynomial of degree n, we will be solving $X\alpha = y$, where $X^{n\ x\ n}$.

$$\begin{bmatrix} 1 & t_1 & t_1^2 & ... & t_1^n \\ 1 & t_2 & t_2^2 & ... & t_2^n \\ 1 & t_3 & t_3^2 & ... & t_3^n \\ ... \\ 1 & t_n & t_n^2 & ... & t_n^n \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ ... \\ \alpha_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ ... \\ y_n \end{bmatrix}$$

Using ordinary least-squares, we solve for $\alpha = (X^TX)^{-1}X^Ty$.

(d) Figure 2 shows a plot of the degree $D \in 1, 2, ...n-1$ versus the mean-squared error R(D) and $\tilde{R}$ when using the data in y.dat, yfresh.dat, and t.dat. Why do you think that this plot is qualitatively different from the plot in part (b)? Even though we are fitting D = n-1 degree polynomial to the new yfresh.dat data, the model was trained on y.dat and will approximate yfresh.dat with greater error than the data it was trained on and cannot be a perfect estimator. Thus, the error appears to plateu for the same values of D with yfresh.dat or y.dat but at a higher error value when using yfresh.dat. What does this tell you how the fitted degree D should be chosen? Choose the minimal degree D after which the error doesn't change within some small ϵ bound.

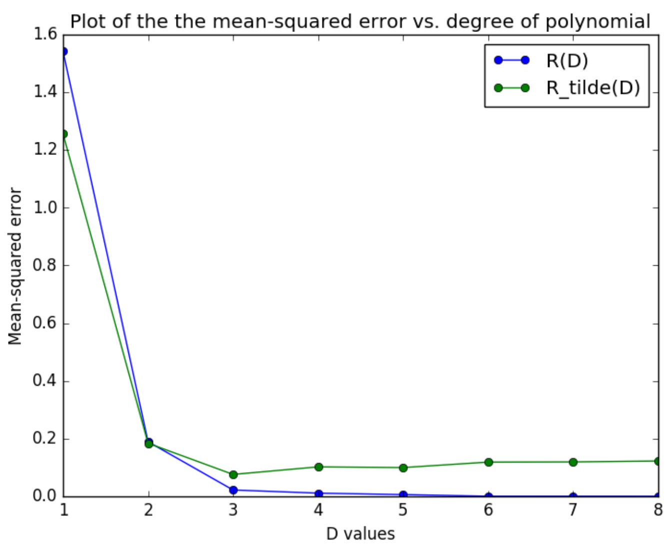

Figure 2: D vs. R(D) and $\tilde{R}$

(e) Figure 3 shows a plot of the degree $D \in 2, ...9$ versus the mean-squared error $\tilde{R}$ and F(D) when using the data in y.dat, yfresh.dat, and t.dat. How are the minimizing arguments of the two functions related? Why is this an interesting observation?

Problem 2.2.

(a) Prove that A is a convex function.

Proof. By definition,

$$A(\eta) = log(\int_{\gamma} h(y)e^{\eta y}\, dy) \quad , \quad p_{\eta}(y) = h(y)e^{\eta y - A(\eta)}$$

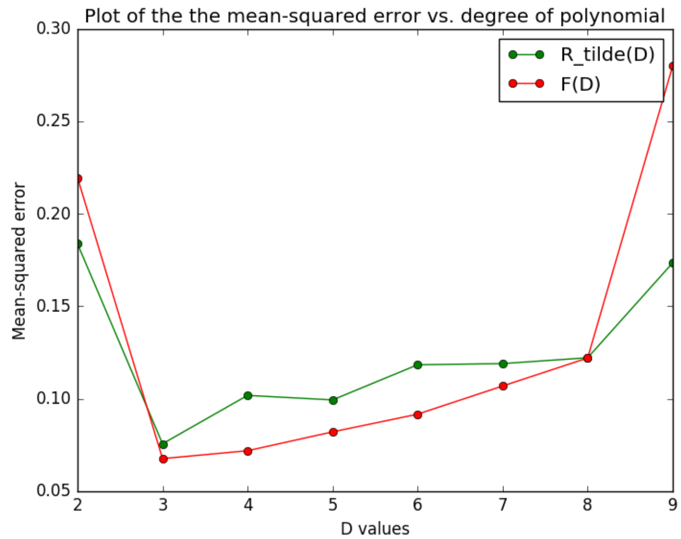

Figure 3: D vs. $\tilde{R}$ and F(D)

To prove convexity, we want to take the second derivative. Let:

$$B(\eta) = \int_\gamma h(y)e^{\eta y}\, dy$$

Then the first derivative we get:

$$\frac{\partial A(\eta)}{\partial \eta} = \left(\frac{1}{B(\eta)}\right)\left(\frac{\partial B(\eta)}{\partial \eta}\right) = \frac{\int_\gamma h(y)e^{\eta y} y\, dy}{\int_\gamma h(y)e^{\eta y}\, dy} = \frac{\int_\gamma h(y)e^{\eta y - A(\eta)} y\, dy}{\int_\gamma h(y)e^{\eta y - A(\eta)}\, dy} = E_{p_\eta}[y]$$

Taking the second derivative we have:

$$\frac{\partial}{\partial \eta}\frac{B'(\eta)}{B(\eta)} = \frac{\partial}{\partial \eta}\left(B'(\eta)\frac{1}{B(\eta)}\right) = \frac{B''(\eta)}{B(\eta)} - \frac{(B'(\eta))^2}{B(\eta)^2}$$

$$= \frac{\int_\gamma h(y)e^{\eta y} y^2\, dy}{\int_\gamma h(y)e^{\eta y}\, dy} \; - \; (E_{p_\eta}[y])^2 = \frac{\int_\gamma h(y)e^{\eta y - A(\eta)} y\, dy}{\int_\gamma h(y)e^{\eta y - A(\eta)}\, dy} \; - \; (E_{p_\eta}[y])^2$$

$$= E_{p_\eta}[y^2] \; - \; (E_{p_\eta}[y])^2 = Var_{p_\eta}[y] \; \succeq \; 0$$

Since Var_{p_η} is positive definite, we have shown that $A(\eta)$ is convex. □

(b) Express KL divergance in terms of $A(\eta)$ and $A'(\eta)$.

$$D(p_\eta || p_\eta) = E_\eta\left(log(\frac{h(y)e^{\eta y - A(\eta)}}{h(y)e^{\tilde{\eta} y - A(\tilde{\eta})}})\right)$$

$$= \int_y log\left(e^{(\eta - \tilde{n})y - (A(\eta) - A(\tilde{\eta}))} p_\eta(y)\right)\, dy$$

$$= \int_y ((\eta - \tilde{n})y - (A(\eta) - A(\tilde{\eta}))\, h(y)e^{\eta y - A(\eta)}\, dy$$

$$= (\eta - \tilde{n}) \int_y h(y) e^{\eta y - A(\eta)} y \, dy - (A(\eta) - A(\tilde{\eta})) \int_y h(y) e^{\eta y - A(\eta)} \, dy$$

$$= (\eta - \tilde{n}) A' - A(\eta) + A(\tilde{\eta})$$

Since $A = \int_y h(y) e^{\eta y - A(\eta)} y$ and $\int_y p_\eta(y) dy = 1$ by definition.

(i) Bernoulli random variable:

$$p_\eta(y) = \eta^y (1-\eta)^{1-y}, y \in 0, 1, n \in (0,1)$$

$$= e^{y log(\eta) + (1-y) log(1-\eta)} = e^{y log(\frac{\eta}{1-\eta}) - log(1+e^{\frac{\eta}{1-\eta}})}$$

Thus, we have $A(\eta) = log(1+e^\eta)$ and $A^*(t) = sup_{\eta \in R} \ \{\eta t - log(1+e^\eta\}$. We now take the gradient of A^* with respect to η, set this to 0 in order to solve the optimization problem, and then solve for η in terms of t.

$$\nabla_\eta A^*(t) = t - \frac{e^\eta}{1+e^\eta}$$

$$0 = t - \frac{e^\eta}{1+e^\eta} \implies t = \frac{e^\eta}{1+e^\eta}$$

$$\frac{1}{t} = \frac{1+e^\eta}{e^\eta} = \frac{1}{e^\eta} + 1 \implies \frac{1}{t} - 1 = \frac{1}{e^\eta}$$

$$e^\eta = \frac{1}{\frac{1}{t} - 1} \implies \eta = log(\frac{1}{\frac{1}{t} - 1})$$

$$\eta = -log(\frac{1}{t} - 1))$$

Substituting this back into our equation, we get:

$$A^*(t) = -t log(\frac{1}{t} - 1) - log(1 + e^{-log(\frac{1}{t} - 1)}) = -t log(\frac{1}{t} - 1) + log(1-t)$$

$$= t log(t) - t log(1-t) + log(1-t) = t log(t) + (1-t) log(1-t)$$

(ii) Gaussian random variable:

$$p_\eta(y) = \frac{e^{\frac{-y^2}{2}}}{\sqrt{2\pi}} e^{yn - \frac{\eta^2}{2}}$$

Thus, we have $A(\eta) = \frac{\eta^2}{2}$ and $A^*(t) = sup_{\eta \in R} \ \left\{\eta t - \frac{\eta^2}{2}\right\}$.

$$\nabla_\eta A^*(t) = t - n$$

$$0 = t - n \implies t = n$$

Substituting this back into our equation, we get:

$$A^*(t) = t^2 - \frac{t^2}{2} = \frac{t^2}{2}$$

(iii) Poisson random variable:

$$p_\eta(y) = \frac{1}{y!} e^{y\eta - e^\eta}$$

Thus, we have $A(\eta) = e^{\eta}$ and $A^*(t) = sup_{\eta \in R}\ \{\eta t - e^{\eta}\}$.

$$\nabla_{\eta} A^*(t) = t - e^{\eta}$$

$$0 = t - e^{\eta} \implies log(t) = n$$

Substituting this back into our equation, we get:

$$A^*(t) = tlog(t)e^{log(t)} = tlog(t) - t = t(log(t) - 1)$$

(d) Prove that A^* is always a convex function.

Proof. If A^* is always a convex function, then it must satisfy the definition of convexity:

$$A^*(\alpha t_1 + (1-\alpha)t_2) \leq \alpha A^*(t_1) + (1-\alpha)A^*(t_2)$$

We know that A^* is defined by:

$$A^*(\alpha t_1 + (1-\alpha)t_2) = sup_{\eta}\, \{\eta(\alpha t_1 + (1-\alpha)t_2) - A(\eta)\}$$

$$= sup_{\eta}\, \{\eta(\alpha t_1 + (1-\alpha)t_2) - \alpha A(\eta) - (1-\alpha)A(\eta)\}$$

$$= sup_{\eta}\, \{\alpha(\eta t_1 - A(\eta)) + (1-\alpha)(\eta t_2 - A(\eta))\}$$

Let $h(\eta) = \eta t_1 - A(\eta)$ and $k(\eta) = \eta t_2 - A(\eta)$. We know that:

$$sup_{\eta}(h(\eta) + k(\eta)) \leq sup_{\eta}(h(\eta)) + sup_{\eta}(k(\eta))$$

By this property and after resubstituting, we have shown:

$$A^*(\alpha t_1 + (1-\alpha)t_2) \leq \alpha A^*(t_1) + (1-\alpha)A^*(t_2)$$

And that A^* is always a convex function.

□

Problem 2.3.

(a) By definition, we have likelihood of η as:

$$l(\eta; y_1, ...y_n) = log(p(y_1, ..., y_n|\eta)) = log(h(y_1, ...y_n)) + \eta^T \sum_{i=1}^{n} y_i - nA(\eta)$$

We differentiate and solve for $\hat{\eta}$ (assuming the inverse function exists under suitable regularity conditions):

$$\frac{\partial}{\partial \eta} l(\eta; y_1, ...y_n) = \sum_{i=1}^{n} y_i - n\frac{\partial}{\partial \eta}A(\eta)$$

$$\frac{\partial}{\partial \eta}A(\eta) = \frac{\sum_{i=1}^{n} y_i}{n} \implies \hat{\eta} = (A'^{-1})(\frac{\sum_{i=1}^{n} y_i}{n})$$

(b) Closed-form estimates for MLE in Poisson, Bernoulli, Gaussian models:

Poisson

$$\frac{\partial}{\partial \eta}A(\eta) = e^{\eta} = \frac{\sum_{i=1}^{n} y_i}{n}$$

$$\hat{\eta} = log(\frac{\sum_{i=1}^n y_i}{n})$$

Bernoulli (where $\frac{\sum_{i=1}^n y_i}{n} \neq 1$)

$$\frac{\partial}{\partial \eta} A(\eta) = \frac{e^\eta}{1+e^\eta} = \frac{\sum_{i=1}^n y_i}{n}$$

$$\hat{\eta} = log(\frac{\frac{\sum_{i=1}^n y_i}{n}}{1 - \frac{\sum_{i=1}^n y_i}{n}})$$

Gaussian

$$\frac{\partial}{\partial \eta} A(\eta) = \eta = \frac{\sum_{i=1}^n y_i}{n}$$

$$\hat{\eta} = \frac{\sum_{i=1}^n y_i}{n}$$

(c)
We know that $E(\bar{y}) = A'(\eta^*)$ and by definition of MLE, we have:

$$max_\eta \left\{ \eta \sum_{i=1}^n y_i - nA(\eta) \right\} = max_\eta \left\{ \eta \bar{y} - A(\eta) \right\} = min_\eta \left\{ -\eta \bar{y} + A(\eta) \right\}$$

Thus, we know that as $n \to \infty$, then $\bar{y} \to A'(\eta^*)$. Additionally, we can add or subtract any terms to this equation that do not depend on η, since they are just constants. We can substitute this into the above equation to get:

$$\hat{\eta} = min_\eta \left\{ -\eta A'(\eta^*) + A(\eta) \right\} = min_\eta \left\{ -\eta A'(\eta^*) + A(\eta) + \eta^* A'(\eta^*) - A(\eta^*) \right\}$$

After some rearranging of terms, we get the final equation:

$$\hat{\eta} = min_\eta \left\{ A(\eta) - A(\eta^*) - A'(\eta^*)(\eta - \eta^*) \right\}$$

(d) Assume A is strictly convex and that η^* is the true parameter. The we can see that η^* minimizes the MLE equation since we have:

$$\eta^* = min_\eta \left\{ A(\eta^*) - A(\eta^*) - A'(\eta^*)(\eta^* - \eta^*) \right\} = 0$$

Hence, η^* ensures that the equation is minimized.

Problem 2.4.

(a) Based on the definition of MLE in GLM as well as stochastic gradient, we can redefine the function $L(\theta)$, the gradient $\triangle^t L(\theta)$, and the $\tilde{\theta}^{t+1}$ update step as:

$$L(\theta) = -y_I x_I^T \theta^t + A(x_I^T \theta^t)$$
$$\triangle^t L(\theta) = -y_I x_I^T + x_I^T A'(x_I^T \theta^t)$$
$$\tilde{\theta}^{t+1} = \hat{\theta}^t - \gamma^t \triangle^t L(I)$$

(b) Explicit updates for Possion and Logistic cases:

Poisson

$$A(t) = e^t \implies \tilde{\theta}^{t+1} = \hat{\theta}^t - \gamma^t x_I^T e^{x_I^T \theta^t}$$

Logistic

$$A(t) = log(1 + e^t) \implies \tilde{\theta}^{t+1} = \hat{\theta}^t - \gamma^t x_I^T (\frac{e^{x_I^T \theta^t}}{1 + e^{x_I^T \theta^t}})$$

(c) Figure 4 shows a histogram plot of the probabilities $P[y_i|x_i; \hat{\theta}]$ based on fitted vector $\hat{\theta}$ and files Xone.dat and yone.dat.

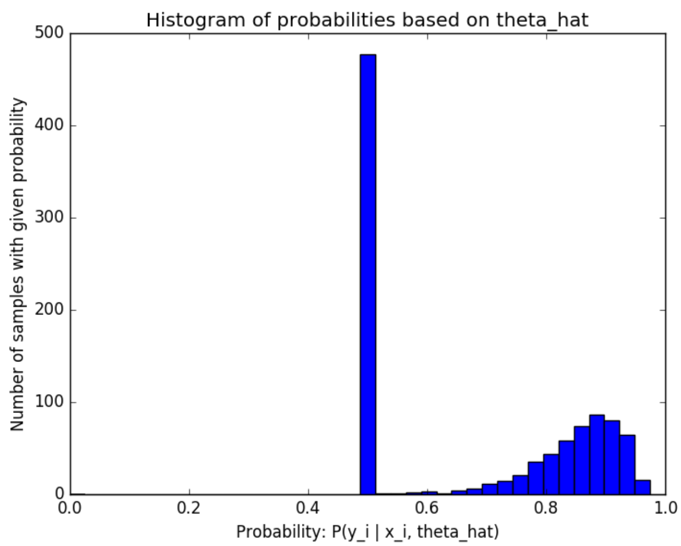

Figure 4: Histogram of probabilities using Xone.dat and yone.dat

(d) Figure 5 shows a histogram plot of the probabilities $P[y_i|x_i; \hat{\theta}]$ based on fitted vector $\hat{\theta}$ and files Xtwo.dat and ytwo.dat. The differences in Figure 4 and 5 are [ ] and suggest about the accuracy of the fits.

(e) See Figure 6 for visualization of 2-component GMM to Xone.dat and Xtwo.dat.

(f) In part (e), the fitted mean vectors are:

$$\mu_{GMM1} = \begin{bmatrix} 3.03708769 \\ -3.03958106 \\ 0.07311787 \\ -0.03310467 \end{bmatrix} \mu_{GMM2} = \begin{bmatrix} 2.96847677 \\ 3.01983729 \\ -0.02951317 \\ -0.06022483 \end{bmatrix}$$

$$\hat{\theta}_{Log1} = \begin{bmatrix} 0.6304399 \\ 0.05635908 \end{bmatrix} \hat{\theta}_{Log2} = \begin{bmatrix} 0.32693246 \\ 0.09650988 \end{bmatrix}$$

```
'''
Implementation of polynomal fit.
'''

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

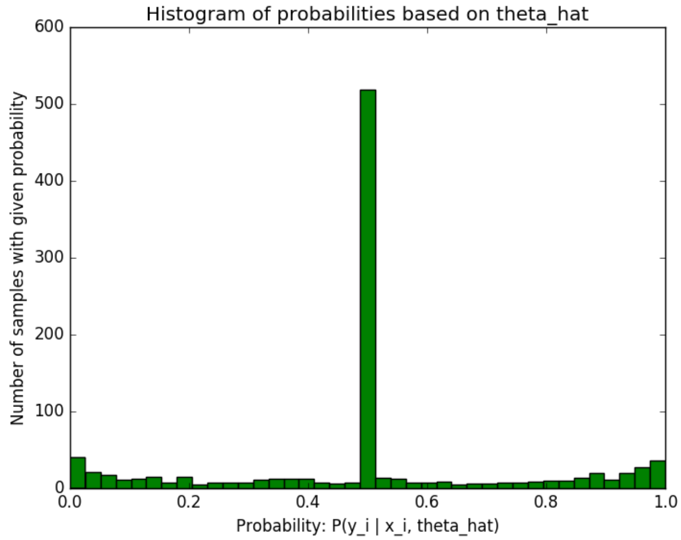

Figure 5: Histogram of probabilities using Xtwo.dat and ytwo.dat

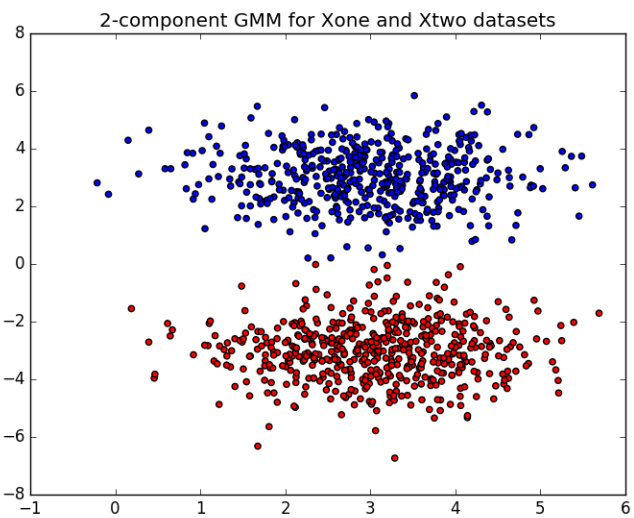

Figure 6: Results of fitting a 2-component GMM to Xone.dat and Xtwo.dat

```
# fit a polynomial of degree to t and y data
def polyfit_D(t, y, degree):
   n = len(y)

   # make a [N x (Degree+1)] matrix
   X = np.zeros((n, degree+1))
   for i in range(n):
      for j in range(degree+1):
         X[i][j] = (t[i])**j
   #print X
```

```
    # using ordinary least squares
    # compute (X^T*X)^(-1) * X^T * y
    XtX = np.dot(np.transpose(X),X)
    inv_XtX = np.linalg.inv(XtX)
    Xty = np.dot(np.transpose(X),y)
    coeff = np.dot(inv_XtX, Xty)

    return coeff

# returns MSE values for polynomial fitting with degree = [deg_min, deg_max]
# adjusted parameter determines if to add (sigma^2)*DLog(n)/n to MSE value
def polyfit_D_range(t,y,y_tilde,deg_min,deg_max,adjusted):
    i = 0

    MSE_vals = np.zeros(deg_max-deg_min)
    MSE_vals_ytilde = np.zeros(deg_max-deg_min)
    D_vals = np.zeros(deg_max-deg_min)
    # fit the data with a D degree polynomial
    for D in range(deg_min, deg_max):
        # compute coefficients for polynomial of degree D
        coeffs = polyfit_D(t,y,D)
        if(D == 3):
            print coeffs
        # reverse order of coefficients for poly1d function
        rev_coeffs = np.fliplr([coeffs])[0]

        # construct the polynomial for graphing
        polyn = np.poly1d(rev_coeffs)
        # compute mean squared error
        MSE_vals[i] = MSE(t,y,polyn,D)
        MSE_vals_ytilde[i] = MSE(t,y_tilde,polyn,D)
        if(adjusted):
            sigma_2 = 0.25**2
            MSE_vals[i] += (sigma_2*D)*np.log(len(y))/len(y)
        D_vals[i] = D
        i += 1
    return (D_vals, MSE_vals, MSE_vals_ytilde)

# plot MSE vs degree for R(D) and F(D)
def plot_MSE2(D_vals, MSE_vals_y1, MSE_vals_ytilde1):
    # visualize degree vs. MSE
    plt.plot(D_vals, MSE_vals_y1, 'o-b', D_vals, MSE_vals_ytilde1, 'o-g')
    plt.title('Plot of the the mean-squared error vs. degree of polynomial')
    plt.legend(['R(D)', 'R_tilde(D)'])
    plt.xlabel('D values')
    plt.ylabel('Mean-squared error')
    plt.show()

# compute mean squared error for estimated polynomial of degree D
def MSE(t, y, polyn, D):
    n = len(y)
    sum = 0.0
    for i in range(0,n):
```

```
        sum += (y[i] - polyn(t[i])) ** 2

    return sum/n

if __name__ == "__main__":
    t = np.loadtxt('data_problem2.1/t.dat')
    y_orig = np.loadtxt('data_problem2.1/y.dat')
    y_fresh = np.loadtxt('data_problem2.1/yfresh.dat')

    # choose a y data source
    y = y_orig
    n = len(y) # 9 in this case

    # R(D), R_tilde(D)
    (D_vals1, MSE_vals_y1, MSE_vals_ytilde1) = polyfit_D_range(t, y_orig, y_fresh, 2, 10,
        0);
    # F(D), F_tilde(D)
    (D_vals2, MSE_vals_y2, MSE_vals_ytilde2) = polyfit_D_range(t, y_orig, y_fresh, 2, 10,
        1);

    plot_MSE2(D_vals1, MSE_vals_y1, MSE_vals_ytilde1)
    plot_MSE2(D_vals2, MSE_vals_ytilde1, MSE_vals_y2)
```

```
"""
Implementation of stochastic gradient descent.
"""
import numpy as np
from numpy import *
import matplotlib.pyplot as plt

def log_loss(X,y,theta):
    sum = 0
    for i in range(m):
        sum += log(1+np.exp(-y[i]*((theta.T)*X[i] + b)))
    sum = -sum

def stoch_gd(X, y, numIter, stepSize, epsilon):
    (m,n) = np.shape(X)
    theta = np.random.rand(n)

    # begin iterations
    for i in range(numIter):
        I = np.random.randint(0,m)

        extheta = np.exp((theta.T)*X[I])
        # compute the gradient at the current location
        g = -y[I]*X[I] + X[I]*extheta/(1+extheta)

        # step in the direction of the gradient
        theta2 = theta - (stepSize/(i+1))*g

        if(np.dot(theta2-theta, theta2-theta) < epsilon):
            return theta
```

```
        else:
            theta = theta2

    # return the solution
    return theta


if __name__ == '__main__':
    Xone = np.loadtxt('data_problem2.4/Xone.dat')
    yone = np.loadtxt('data_problem2.4/yone.dat')

    Xtwo = np.loadtxt('data_problem2.4/Xtwo.dat')
    ytwo = np.loadtxt('data_problem2.4/ytwo.dat')

    (m,n) = np.shape(Xone)
    # take number of iterations to be number of examples
    numIter = m
    epsilon = 0.0000000000001
    stepSize = 0.01

    # data set #1
    theta_hat1 = stoch_gd(Xone, yone, numIter, stepSize, epsilon)
    e_yxtheta1 = np.exp(yone*np.dot(Xone,theta_hat1))
    p_one = e_yxtheta1/(1+e_yxtheta1)

    # data set #2
    theta_hat2 = stoch_gd(Xtwo, ytwo, numIter, stepSize, epsilon)
    e_yxtheta2 = np.exp(ytwo*np.dot(Xtwo,theta_hat2))
    p_two = e_yxtheta2/(1+e_yxtheta2)

    bins = np.linspace(0, 1, 40)

    plt.title("Histogram of probabilities based on theta_hat")
    plt.xlabel("Probability: P(y_i | x_i, theta_hat)")
    plt.ylabel("Number of samples with given probability")
    #plt.hist(p_one, bins)
    plt.hist(p_two, bins, facecolor='green')
    plt.show()
```

```
"""
Implementation of 2-component GMM.
"""
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
from sklearn import mixture

def run_gmm(X, num_components):
    gmm = mixture.GMM(n_components=num_components, covariance_type='full')
    gmm.fit(X)
    print gmm.means_
    colors = ['r' if i==0 else 'b' for i in gmm.predict(X)]
    p = plt.gca()
```

```
    p.scatter(X[:,0], X[:,1], c=colors)
    plt.title("2-component GMM for Xone and Xtwo datasets")
    plt.show()


if __name__ == '__main__':
    Xone = np.loadtxt('data_problem2.4/Xone.dat')
    Xtwo = np.loadtxt('data_problem2.4/Xtwo.dat')

    yone = np.loadtxt('data_problem2.4/yone.dat')
    ytwo = np.loadtxt('data_problem2.4/ytwo.dat')

    X = np.concatenate((Xone, Xtwo), axis=1)

    run_gmm(X, 2)
```