SML_assignment_3

November 22, 2016

1 Exercise 1 - Bayesian Linear Regression

1.1 1.1

We begin by computing p(t, x, **x,t**), using the results from exercise 2 week 8:

$$\begin{aligned}
p(t \mid x, \boldsymbol{x}, \boldsymbol{t}) &= \mathcal{N}(t \mid m(x), s^2(x)) \\
m(x) &= \phi(x)^T \boldsymbol{m_n} && = N\beta(1, x)\boldsymbol{S_N}\begin{pmatrix} \bar{\mu}_t \\ \bar{\mu}_{xt} \end{pmatrix} \\
s^2(x) &= \beta^{-1} + \phi(x)^T \boldsymbol{S_N}\phi(x) && = \beta^{-1} + (1, x)\boldsymbol{S_N}\begin{pmatrix} 1 \\ x \end{pmatrix} \\
\boldsymbol{S_N}^{-1} &= \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} + N\beta\begin{pmatrix} 1 & \bar{\mu}_x \\ \bar{\mu}_x & \bar{\mu}_{xx} \end{pmatrix}
\end{aligned}$$

We are given

$$x = \begin{pmatrix} .4 \\ .6 \end{pmatrix}$$

$$t = \begin{pmatrix} .05 \\ -.35 \end{pmatrix}$$

Hence, we compute :

$$\begin{aligned}
\bar{\mu}_t &= \frac{1}{N}\sum_n t_n && = -.15 \\
\bar{\mu}_x &= \frac{1}{N}\sum_n x_n && = .5 \\
\bar{\mu}_{xt} &= \frac{1}{N} x_n t_n && = -.095 \\
\bar{\mu}_{xx} &= \frac{1}{N} x_n^2 && = .26
\end{aligned}$$

Confirmation with python 3.5:

```
In [1]: # START PRE-AMBLE
        import numpy as np
        from pylab import *
        # use latex interpreter
        rc('text', usetex=True)
        import scipy.stats as stats
        import seaborn as sb
        # END - PREAMBLE

        ar = np.array; inv = np.linalg.inv
        # given parameters
        a = 2 ; b = 10
        x = ar([0.4, .6])
        t = ar([.05, -.35])

        # set number of observed points
        N = len(x)
        # define mu parameters
        mu_bar_t  = np.mean(t)
        mu_bar_xt = np.mean(t * x)
        mu_bar_x  = np.mean(x)
        mu_bar_xx = np.mean(x**2)

        print('mu_bar_t ={0}\nmu_bar_x = {1}\nmu_bar_xt ={2}\nmu_bar_xx= {3}'\
             .format(mu_bar_t, mu_bar_x, mu_bar_xt, mu_bar_xx))
```

```
mu_bar_t =-0.15
mu_bar_x = 0.5
mu_bar_xt =-0.095
mu_bar_xx= 0.26
```

Now we can compute S_n, $m(x)$, and $s^2(x)$:

$$S_N^{-1} = \begin{pmatrix} 22 & 10 \\ 10 & 7.2 \end{pmatrix}$$

$$S_n = \begin{pmatrix} .1233 & -.1712 \\ -.1712 & .3767 \end{pmatrix}$$

$$\begin{aligned}
s^2(x) &= \beta^{-1} + (1,x)\boldsymbol{S_N}\begin{pmatrix} 1 \\ x \end{pmatrix} \\
&= \frac{1}{10} + (1,x)\begin{pmatrix} .1233 & -.1712 \\ -.1712 & .3767 \end{pmatrix}\begin{pmatrix} 1 \\ x \end{pmatrix} \\
&= .1 + .1233 - .1712x - .1712x + .3767x^2 \\
&= .2233 - .3425x + .3767x^2
\end{aligned}$$

$$\begin{aligned}
m(x) &= N\beta(1,x)\boldsymbol{S_N}\begin{pmatrix} \bar{\mu}_t \\ \bar{\mu}_{xt} \end{pmatrix} \\
&= 20(1,x)\begin{pmatrix} .1233 & -.1712 \\ -.1712 & .3767 \end{pmatrix}\begin{pmatrix} -.15 \\ -.095 \end{pmatrix} \\
&= (1,x)\begin{pmatrix} -0.04452055 \\ -0.20205479 \end{pmatrix} \\
&= -0.04452055 - 0.20205479x
\end{aligned}$$

1.2 1.2

Note we must use the square root of $s^2(x)$ as this would be the standard deviation. Using the results from question 1 we plot:

```
In [2]: # see previous exercise for these results:
        m_x  = lambda x: -0.04452055 -0.20205479 *x
        s2_x = lambda x: .2233 -.3425 * x + .3767 * x**2

        # domain over [0,1]
        x_domain = np.linspace(0,1)

        mu = m_x(x_domain)
        std = np.sqrt(s2_x(x_domain))

        # %matplotlib inline
        # plot results
        fig = figure();
        ax = fig.add_subplot(111)
        ax.plot(\
                x_domain, mu,\
                'b-', label = 'mean')
        # draw +- std as an area like in bisschop
        ax.fill_between(x_domain, mu + std, mu - std , \
                        facecolor = 'red', alpha =.2)
```

```
ax.plot(x, t,'g.', label = 'data');
ax.set_xlabel('x');
ax.legend()
ax.set_title('Ex. 1.2');
savefig('Figures/12.png')
```

1.3 1.3

We know that $p(w \mid, \alpha) = \mathcal{N}(\boldsymbol{w}, \mid 0, \alpha^{-1}, I)$; From week 8 exercise 2, we see that the posterior for w is given as $p(w \mid t, x) = \mathcal{N}(w \mid m_n, S_n)$, with $m_N = \beta S_n \phi^T t$ and $S_n^{-1} = \alpha I + \beta \phi^T \phi$.

Thus I will write code that will: - Create multivariate normal object with $\mu = m_N$, $\Sigma = S_N$ - Draw 5 samples from this distribution and plot according to :$ y = w0 + w1 * x$

```
In [3]: # load multivariate normal object
        from scipy.stats import multivariate_normal as mv

        phi = np.vstack((np.ones(x.shape), x))
        # print(phi); assert 0
        s_n = np.linalg.inv(a * np.eye(N) + b * phi.T.dot(phi))
        m_n = b * s_n .dot(phi.T.dot(t))

        # create object for p(w | 0, a^-1 I)

        # p_w = mv([0,0], 1/a * np.eye(N))
        p_w = mv(m_n, s_n)

        # draw 5 random samples
        random_w = p_w.rvs(5).T
        # create inline plot function for the line
        plot_w = lambda x, w0, w1: w0 + x * w1

        # open figure and start plotting
        fig = figure()
        ax = fig.add_subplot(111)

        # plot mean and standard deviation
        # mean
        ax.plot(\
                x_domain, m_x(x_domain),\
                'b-', label = 'optimal');
        # standard deviation; plotting as an area
        ax.fill_between(x_domain, mu + std, mu-std, \
                        facecolor = 'r', alpha = .2)
        # plot the data
        ax.plot(x, t, 'g.', label = 'data')

        # create 5 straight lines
        [ax.plot(\
```

```
                 x_domain, plot_w(x_domain, w[0], w[1]),\
                 '--', label = 'sample {}'.format(idx)\
                )\
                for idx, w in enumerate(random_w.T)];

        # plot formatting
        ax.legend(loc = 0,\
                  ncol = 1,\
                  frameon = True,\
                  framealpha = 1,\
                  bbox_to_anchor = (1., 1.));

        ax.grid('off');
        sb.set_style('white');
        # ax.set_ylim([-2,2])
        ax.set_xlabel('x');
```

2 Exercise 2 - Logistic Regression

$$
\begin{aligned}
f(x) &= \sin(x) \\
x^{(n+1)} &= x^{(n)} - H^{-1}\nabla f(x^{(n)}) \\
&= x^{(n)} + \frac{\cos(x^{(n)})}{\sin(x^{(n)})}
\end{aligned}
$$

2.1 1.1

```
In [4]: def newton_sin(x0, maxiter = 100):
            i = 0
            stop_cond = True
            while stop_cond:
                x = x0 + np.cos(x0) / np.sin(x0)
                if abs(x - x0) < 1e-4:
                    stop_cond = False
                else:
                    x0 = x
                if i > maxiter:
                    print('fail to converge')
                    stop_cond = False
                i += 1
            return x,i
        print(newton_sin(x0 = 1))
        print(newton_sin(x0 = -1))
```

```
(1.5707963267948966, 4)
(-1.5707963267948966, 4)
```

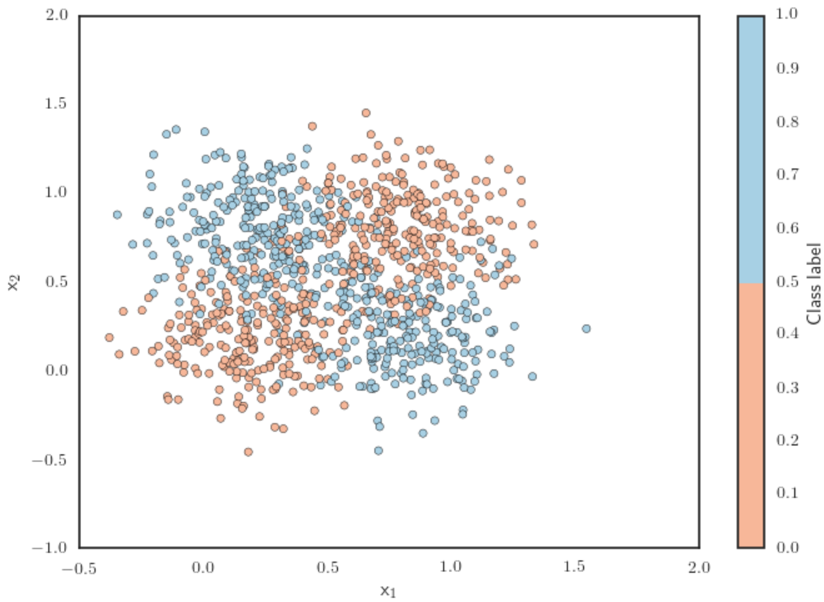
2.2 2.2

```
In [5]: data = np.array([[.3, .44, .46, .6], [1, 0, 1, 0]])
        w0 = np.array([1. ,1. ]).T
        # print(data.shape)
        # define sigmoid function:
        sigmoid = lambda x: 1 / (1 + np.exp(-x))


        def RLS(y,\
                t,\
                phi,\
                w0, \
                threshold = 1e-10, \
                maxiter = 100,\
                cond = True):

            R = np.diag(y * (1 - y))
            i = 0;
            while cond:
                y = sigmoid(phi.dot(w0))
                tmp = np.multiply(y, (1 - y))
                R = np.diag(tmp);

                Z = phi.dot(w0) - np.linalg.pinv(R).dot(y - t)
                w = np.linalg.inv(phi.T.dot(R.dot(phi))).dot(phi.T.dot(R.dot(Z)))
                if np.sum(abs(w - w0)) < threshold: cond = False;
                if i > maxiter : cond = False
                w0 = w; i += 1
            return w0, i
        y, t = data
        phi = np.stack((np.ones(y.shape), y))
        # print(phi)
        # print(phi.shape)
        w0, i = RLS(y, t, phi.T, w0)

        print('Iterations = {0}, w = {1}'.format(i,w0.T))
```

```
Iterations = 7, w = [  9.78227684 -21.73839298]
```

2.3 2.1

```
In [6]: iris_data = np.loadtxt('iris_data.txt')
        x = iris_data[:,:2]; c = iris_data[:,-1]


        # set colormaps for labels and probabilities
        from matplotlib.colors import ListedColormap
```



test, aklfajkldfjlakdfjklafljk

```
cmap = sb.color_palette('RdBu', 2)
cmap_class = ListedColormap(cmap)
cmap = sb.color_palette('RdBu', 100)
cmap_prob = ListedColormap(cmap)

fig  = figure();
ax   = fig.add_subplot(111)
p    = ax.scatter(x[:,0 ], x[:,1], c = c, cmap = cmap_class);

ax.set_xlabel('x_1');
ax.set_ylabel('x_2');
cb = fig.colorbar(p); cb.set_label('Class label')
savefig('Figures/21.png')
```

1
Explain why log. regression is a good approach for this data set

2.4 2.2

```
In [7]: w0 = np.array([0,0,0])
        phi = np.hstack((np.ones( (x.shape[0],1) ) ,  x)).T
        w, i = RLS(x, c, phi.T, w0)

        # sigmoid function
        # probability before optimization
        prob_classified_before_opt = sigmoid(w0.dot(phi))
```

```
        prob_classified_after_opt  = sigmoid(w.dot(phi))
        c1 = np.mean(prob_classified_after_opt[c == 1])
        c2 = 1 - c1
        print(('The class probabilities are :'\
               ' class 1 = {0}, class 2 = {1}').\
              format(c1, 1-c1))

        print('w0 = {0}, i = {1}'.format(w, i))
```

```
The class probabilities are : class 1 = 0.49205021284958317, class 2 = 0.5079497871
w0 = [ 0.00440664 -0.02139153 -0.04930069], i = 3
```

```
In [8]: # create feature matrix with bias
        tmp = np.ones((1, x.shape[0]))
        phi = np.vstack([tmp, x.T])


        # after optimization
        prob_classified_after_opt =  sigmoid(w.dot(phi))

        # equation 4.90 Bisschop
        cross_entropy = lambda y, t:\
                -np.sum( t * np.log(y) + (1 - t) * np.log(1 - y))
        # cross entropy with w = [0,0,0]
        cross_entropy_before_opt = cross_entropy(prob_classified_before_opt, c)

        # cross entropy after optimization using IRS
        cross_entropy_after_opt  = cross_entropy(prob_classified_after_opt, c)
        print('cross-entropy : before = {0}, after = {1} optimization'\
              .format(cross_entropy_before_opt, cross_entropy_after_opt))

        # plot before optimization
        fig = figure(); fig.clf()
        ax = fig.add_subplot(2,1,1)
        p = ax.scatter(x[:,0], x[:,1], c = prob_classified_before_opt, cmap = cmap_
        cb = fig.colorbar(p); cb.set_label('p($c_1 \mid \phi$)')
        ax.set_xlabel('x_1')
        ax.set_ylabel('x_2')
        sb.set_context('poster')
        ax.set_title('Before optimization')

        # plot after optimization
        ax = fig.add_subplot(2,1,2)
        p = ax.scatter(x[:,0], x[:,1], c = prob_classified_after_opt, cmap = cmap_p
        cb = fig.colorbar(p); cb.set_label('p($c_1 \mid \phi$)')
        ax.set_xlabel('x_1')
        ax.set_ylabel('x_2')
```

```
        sb.set_context('poster')
        ax.set_title('After optimization')

        fig.tight_layout()
        savefig('Figures/22.png')
```

```
cross-entropy : before = 693.1471805599454, after = 692.969359482537 optimization
```

```
In [9]: from scipy.stats import multivariate_normal as mv
        create_basis_vector = lambda mu, sigma: mv(mu, sigma)
        mu1 = np.array([0,0]); mu2 = np.array([1,1])
        sigma2 = .2

        phi1 = create_basis_vector(mu1, sigma2 * np.eye(2))
        phi2 = create_basis_vector(mu2, sigma2 * np.eye(2))

        phi1_pdf = phi1.pdf(x);
        phi2_pdf = phi2.pdf(x);

        phi =  np.vstack((np.ones(phi1_pdf.shape), phi1_pdf, phi2_pdf))
        # print(phi.shape, phi1_pdf.shape); assert 0
        w, i  = RLS(x, c, phi.T, w0)


        prob_gaussian_after_opt = sigmoid(w.dot(phi))
        cross_entropy_gaussian = cross_entropy(prob_gaussian_after_opt, c)
        print('Cross-entropy with gaussian basis function = {0}'\
             .format(cross_entropy_gaussian))

        fig = figure()
        # subplot: origingal inputspace
        ax = fig.add_subplot(211)
        p = ax.scatter(x[:,0], x[:,1], \
                       c = c, cmap = cmap_class);
        cb = fig.colorbar(p); cb.set_label('Class label')
        ax.set_title('Original input space')
        ax.set_xlabel('$x_1$')
        ax.set_ylabel('$x_2$')

        # basis function decomposition
        ax = fig.add_subplot(212)
        ax.set_xlabel('$x_1$')
        ax.set_ylabel('$x_2$')
        sb.set_context('poster')

        p = ax.scatter(phi1.pdf(x), phi2.pdf(x), \
                       c = c, cmap = cmap_class);
```

```
cb = fig.colorbar(p); cb.set_label('Class label')

ax.set_xlabel('$\phi_1$')
ax.set_ylabel('$\phi_2$')
ax.set_title('basis vector space')
sb.set_context('poster')
fig.tight_layout()
savefig('Figures/24.png')
```

```
Cross-entropy with gaussian basis function = 346.50408046148465
```

```
In [10]: fig = figure()

         # optimization results using basis functions
         ax = fig.add_subplot(211)
         p = ax.scatter(phi1.pdf(x), phi2.pdf(x),\
            c = prob_gaussian_after_opt ,\
                        cmap = cmap_prob);
         cb = fig.colorbar(p); cb.set_label('p($c_1 \mid t)$')
         ax.set_xlabel('$\phi_1$')
         ax.set_ylabel('$\phi_2$')
         ax.set_title('$P(c_1 \mid t)$ in basis  space')

         ax = fig.add_subplot(212)
         p = ax.scatter(x[:,0], x[:,1],\
            c = prob_gaussian_after_opt ,\
                        cmap = cmap_prob);
         cb = fig.colorbar(p); cb.set_label('p($c_1 \mid t)$')

         ax.set_xlabel('$x_1$')
         ax.set_ylabel('$x_2$')
         ax.set_title('$P(c_1 \mid t)$ in original space')
         sb.set_context('poster')
         fig.tight_layout()
         savefig('Figures/24.png')
```