Statistical Machine Learning Assignment 3

Inez Wijnands (s4149696) & Guido Zuidhof (s4160703)

Radboud University Nijmegen

03/12/2015

The entire code listing is included in the zip-file. The listings shown here are merely code snippets.

1 Bayesian linear regression

1.

$$\{x_1, t_1\} = (0.4, 0.05) \tag{1.1}$$

$$\{x_2, t_2\} = (0.6, -0.35) \tag{1.2}$$

$$\alpha = 2 \tag{1.3}$$

$$\beta = 10 \tag{1.4}$$

$$\tag{1.5}$$

We need to compute the predictive distribution $p(t|x, \boldsymbol{t}, \boldsymbol{x}, \alpha, \beta)$ after the two data points (where x is the input and t is the target) are observed.

Step 1: Identify the vector of basis functions $\phi(\boldsymbol{x})$, and write out $\boldsymbol{\Phi}^T \boldsymbol{t}$ and $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ in terms of the data $\{x_n, t_n\}$.

$$\boldsymbol{\Phi}^T \boldsymbol{t} = \sum_n \boldsymbol{\phi}(x_n)\, t_n = N \begin{pmatrix} \bar{\mu}_t \\ \bar{\mu}_{xt} \end{pmatrix} \tag{1.6}$$

$$\boldsymbol{\Phi}^T \boldsymbol{\Phi} = \sum_n \boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T = N \begin{pmatrix} 1 & \bar{\mu}_x \\ \bar{\mu}_x & \bar{\mu}_{xx} \end{pmatrix} \tag{1.7}$$

Where:

$$\bar{\mu}_t = \frac{1}{N}\sum_n t_n \qquad \bar{\mu}_{xt} = \frac{1}{N}\sum_n x_n t_n$$

$$\bar{\mu}_x = \frac{1}{N}\sum_n x_n \qquad \bar{\mu}_{xx} = \frac{1}{N}\sum_n x_n^2$$

Combined:

$$\boldsymbol{\Phi}^T \boldsymbol{t} = \sum_n \boldsymbol{\phi}(x_n)\, t_n = N \begin{pmatrix} \bar{\mu}_t \\ \bar{\mu}_{xt} \end{pmatrix} \tag{1.8}$$

$$= N \begin{pmatrix} \frac{1}{N}\sum_n t_n \\ \frac{1}{N}\sum_n x_n t_n \end{pmatrix} \tag{1.9}$$

$$= 2 \begin{pmatrix} \frac{1}{2}(0.05 + -0.35) \\ \frac{1}{2}(0.4 \cdot 0.05 + 0.6 \cdot -0.35) \end{pmatrix} \tag{1.10}$$

$$= \begin{pmatrix} -0.3 \\ -0.19 \end{pmatrix} \tag{1.11}$$

$$\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \sum_n \phi(x_n)t_n = N\begin{pmatrix} 1 & \bar{\mu}_x \\ \bar{\mu}_x & \bar{\mu}_{xx} \end{pmatrix} \tag{1.12}$$

$$= N\begin{pmatrix} 1 & \frac{1}{N}\sum_n x_n \\ \frac{1}{N}\sum_n x_n & \frac{1}{N}\sum_n x_n^2 \end{pmatrix} \tag{1.13}$$

$$= 2\begin{pmatrix} 1 & \frac{1}{2}(0.4+0.6) \\ \frac{1}{2}(0.4+0.6) & \frac{1}{2}(0.4^2+0.6^2) \end{pmatrix} \tag{1.14}$$

$$= \begin{pmatrix} 2 & 1 \\ 1 & 0.52 \end{pmatrix} \tag{1.15}$$

Step 2: Compute the posterior $p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{x},\alpha,\beta)$.

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|0,\alpha^{-1}\boldsymbol{I}) \tag{1.16}$$

$$p(\boldsymbol{t}|\boldsymbol{w},\boldsymbol{x}) = \mathcal{N}(\boldsymbol{t}|\boldsymbol{\Phi}\boldsymbol{w},\beta^{-1}\boldsymbol{I}) \tag{1.17}$$

$$\rightarrow \tag{1.18}$$

$$p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{x}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N,\boldsymbol{S}_N) \tag{1.19}$$

$$\boldsymbol{S}_N^{-1} = \alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} + N\beta\begin{pmatrix} 1 & \bar{\mu}_x \\ \bar{\mu}_x & \bar{\mu}_{xx} \end{pmatrix} \tag{1.20}$$

$$= \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} + 10\begin{pmatrix} 2 & 1 \\ 1 & 0.52 \end{pmatrix} \tag{1.21}$$

$$= \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} + \begin{pmatrix} 20 & 10 \\ 10 & 5.2 \end{pmatrix} \tag{1.22}$$

$$= \begin{pmatrix} 22 & 10 \\ 10 & 7.2 \end{pmatrix} \tag{1.23}$$

$$\boldsymbol{m}_N = \beta\boldsymbol{S}_N\boldsymbol{\Phi}^T\boldsymbol{t} = N\beta\boldsymbol{S}_N\begin{pmatrix} \bar{\mu}_t \\ \bar{\mu}_{xt} \end{pmatrix} \tag{1.24}$$

$$= 10\begin{pmatrix} 22 & 10 \\ 10 & 7.2 \end{pmatrix}^{-1}\begin{pmatrix} -0.3 \\ -0.19 \end{pmatrix} \tag{1.25}$$

$$= \begin{pmatrix} -0.0445 \\ -0.2021 \end{pmatrix} \tag{1.26}$$

Step 3: Compute the predictive distribution $p(t|x,\boldsymbol{t},\boldsymbol{x},\alpha,\beta) = \mathcal{N}(t|m(x),s^2(x))$ in terms of known or computable quantities, we do this the same way as we obtained the posterior distribution in step 2.

$$p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{x}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N,\boldsymbol{S}_N) \tag{1.27}$$

$$p(t|\boldsymbol{w},\boldsymbol{t},\boldsymbol{x}) = \mathcal{N}(t|\boldsymbol{\phi}(x)^T\boldsymbol{w},\beta^{-1} \tag{1.28}$$

$$\rightarrow \tag{1.29}$$

$$p(t|x,\boldsymbol{t},\boldsymbol{x}) = \mathcal{N}(t|\boldsymbol{m}_N^T\boldsymbol{\phi}(x),\sigma_N^2(x)) \tag{1.30}$$

With $\boldsymbol{m}_N$ and $\boldsymbol{S}_N$ defined as before, and

$$\sigma_N^2(x) = \frac{1}{\beta} + \boldsymbol{\phi}(x)^T\boldsymbol{S}_N\boldsymbol{\phi}(x) \tag{1.31}$$

$$p(t|x, \boldsymbol{t}, \boldsymbol{x}, \alpha, \beta) = \mathcal{N}(t|m(x), s^2(x)) \tag{1.32}$$

$$m(x) = \boldsymbol{m}_N^T \boldsymbol{\phi}(x) \tag{1.33}$$

$$= \begin{pmatrix} -0.0445 \\ -0.2021 \end{pmatrix}^T \begin{pmatrix} 1 \\ x \end{pmatrix} \tag{1.34}$$

$$s(x) = \sigma_N^2 \tag{1.35}$$

$$= \frac{1}{10} + \begin{pmatrix} 1 \\ x \end{pmatrix}^T \begin{pmatrix} 22 & 10 \\ 10 & 7.2 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ x \end{pmatrix} \tag{1.36}$$

2. See Figure 1.1 where the mean of the predictive Gaussian distribution and one standard deviation on both sides are plotted as a function over x over the interval $[0, 1]$, with the two observed data points. When we compare this plot with Figure 3.8b (Bishop, p.157), is our linear mean. This can be explained by the nature of our basis function ϕ_j, which is two-dimensional in our case and Bishop uses a 9-dimensional function.

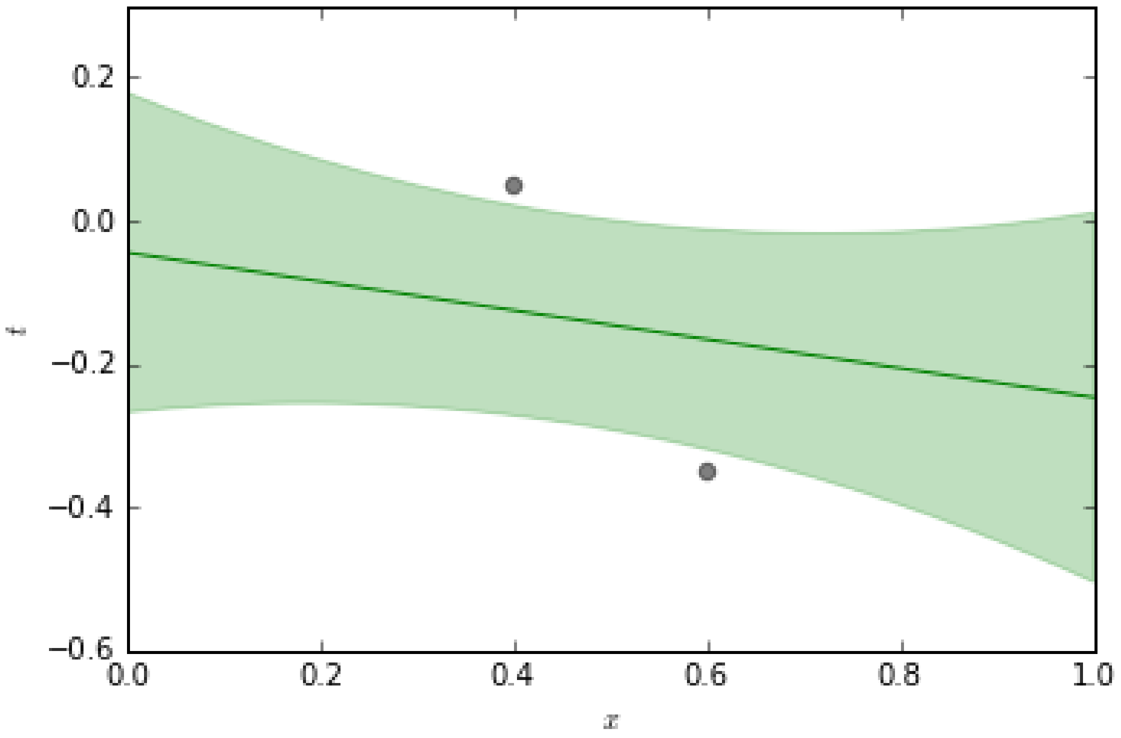

Figure 1.1: The mean of the predictive Gaussian distribution is represented by the green line, plotted against x. The two grey dots are the data points $\{x_1, t_1\}$ and $\{x_2, t_2\}$. The light green area indicates the standard deviation and is bound by the mean plus the standard deviationd and the mean minus the standard deviation.

3. We sampled five functions $y(x, \boldsymbol{w})$ from the posterior distribution over $\boldsymbol{w}$ (see Exercise 1.1 step 2) for this data set and plotted them with the predictive distribution (see Figure 1.2).

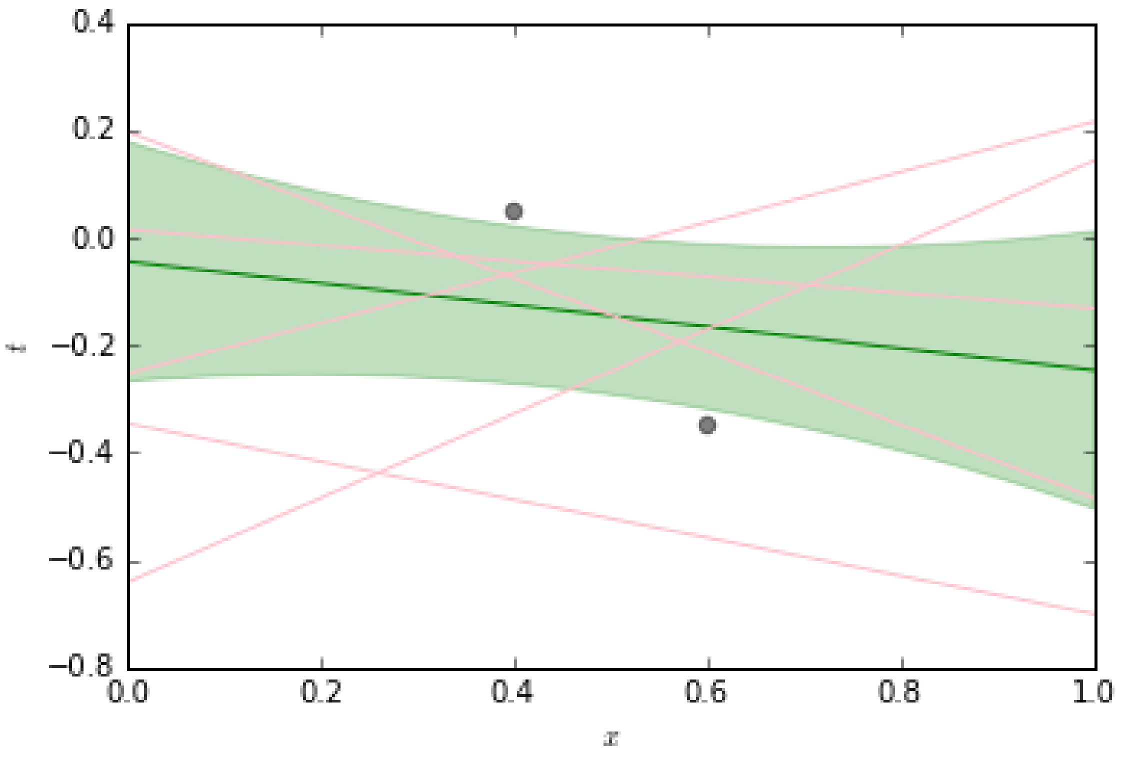

Figure 1.2: The pink lines represent $y(x, \boldsymbol{w})$ where the weights $\boldsymbol{w}$ are sampled from the posterior distribution with mean m_N and variance S_N.

2 Logistic regression

2.1 The IRLS algorithm

1. Because our function $f(\boldsymbol{x}_n)$ is one-dimensional, we can take the second derivate of $f(\boldsymbol{x}_n)$ as our Hessian function, and rewrite Eq. 2.1 as Eq. 2.2.

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \boldsymbol{H}^{-1}\nabla f(\boldsymbol{x}_n) \tag{2.1}$$

$$= \boldsymbol{x}_n - \frac{\frac{df(\boldsymbol{x}_n)}{dx_n} f(\boldsymbol{x}_n)}{\frac{d^2 f(\boldsymbol{x}_n)}{d^2 \boldsymbol{x}_n} f(\boldsymbol{x}_n)} \tag{2.2}$$

$$f(\boldsymbol{x}_n) = sin(\boldsymbol{x}_n) \tag{2.3}$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \frac{cos(\boldsymbol{x}_n)}{-sin(\boldsymbol{x}_n)} = \boldsymbol{x}_n + \frac{cos(\boldsymbol{x}_n)}{sin(\boldsymbol{x}_n)} \tag{2.4}$$

We implemented the Newton-Raphson iterative method in the functions depicted in Listing 1:

Listing 1: Python code for function *hessian_sin(x)* and *run_hessian_sin(x, n_iter=5)*.

```python
def hessian_sin(x):
    return x + (np.cos(x)/np.sin(x))

def run_hessian_sin(x, n_iter=5):
    print "x_0:", x

    for i in xrange(n_iter):
        x = hessian_sin(x)
        print 'x_'+str(i+1)+":",x
```

For $x_0 = 1$ this resulted in:

$$x_1 : 1.64209261593$$

$$x_2 : 1.57067527716$$

$$x_3 : 1.5707963268$$

$$x_4 : 1.57079632679$$

$$x_5 : 1.57079632679$$

The algorithm converges to around 1.5708 after 4 iterations.
For $x_0 = -1$ this resulted in:

$$x_1 : -1.64209261593$$

$$x_2 : -1.57067527716$$

$$x_3 : -1.5707963268$$

$$x_4 : -1.57079632679$$

$$x_5 : -1.57079632679$$

This is the same, except the negative value. For $x_0 = 1$, the value is the maximum of $sin(x)$, whereas for $x_0 = -1$, the value is the minimum of $sin(x)$, this is to be expected. Since this method finds the minima and maxima of a function, this is to be expected.

2. See Listing 2 for the code to calculate the IRLS algorithm on the given data.

Listing 2: Python code for function *sigmoid(x)* and *H(phi, y)* and the $main$.

```
def sigmoid(x):
    return 1.0/(1.0+np.exp(-x))

def H(phi, y):
    R = np.diag(np.ndarray.flatten(y * (1-y)))
    return np.dot(phi.T,(np.dot(R,phi)))


if __name__ == '__main__':
    phi = np.array([[1,0.3],[1,0.44],[1,0.46],[1,0.6]])
    t = np.array([[1],[0],[1],[0]])
    w = np.array([[1],[1]])

    for i in range(10):
        y = sigmoid(np.dot(phi,w))
        h = H(phi, y)

        w = w - np.dot(np.linalg.inv(h) , (np.dot(phi.T,y-t)))
        print i+1, w
```

After 6 iterations our algorithm converges to $\hat{\boldsymbol{w}} = \begin{pmatrix} 9.78228 \\ -21.73839 \end{pmatrix}$, which corresponds to $\hat{\boldsymbol{w}} \approx \begin{pmatrix} 9.8 \\ -21.7 \end{pmatrix}$ as mentioned in the assignment as the optimal solution.

We will show that this corresponds to a decision boundary of $\phi = 0.45$. Since $p(\mathscr{C}_1|\boldsymbol{\phi}) = p(\mathscr{C}_2|\boldsymbol{\phi}) = 0.5$ and we can fill in $\hat{\boldsymbol{w}} = \begin{pmatrix} 9.8 \\ -21.7 \end{pmatrix}$, we can solve the following equation to find ϕ:

$$p(\mathscr{C}_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}) \tag{2.5}$$

$$0.5 = \frac{1}{1+\exp(\boldsymbol{w}^T\boldsymbol{\phi})} \tag{2.6}$$

$$= \frac{1}{1+\exp(\begin{pmatrix} 9.8 & -21.7 \end{pmatrix}\begin{pmatrix} 1 \\ \phi \end{pmatrix})} \tag{2.7}$$

$$2 = 1+\exp(\begin{pmatrix} 9.8 & -21.7 \end{pmatrix}\begin{pmatrix} 1 \\ \phi \end{pmatrix}) \tag{2.8}$$

$$1 = \exp(\begin{pmatrix} 9.8 & -21.7 \end{pmatrix}\begin{pmatrix} 1 \\ \phi \end{pmatrix}) \tag{2.9}$$

$$0 = \begin{pmatrix} 9.8 & -21.7 \end{pmatrix}\begin{pmatrix} 1 \\ \phi \end{pmatrix} \tag{2.10}$$

$$= 9.8 \cdot 1 - 21.7\phi \tag{2.11}$$

$$21.7\phi = 9.8 \tag{2.12}$$

$$\phi = 0.45 \tag{2.13}$$

2.2 Two-class classification using logistic regression

1. The data is plotted and colored according to their labels in Figure 2.1. We think logistic regression is suitable for this problem, since there is a clear separation between classes. The data is not linearly separable by their classes, and thus using the standard feature space with a dummy basis function will not work.

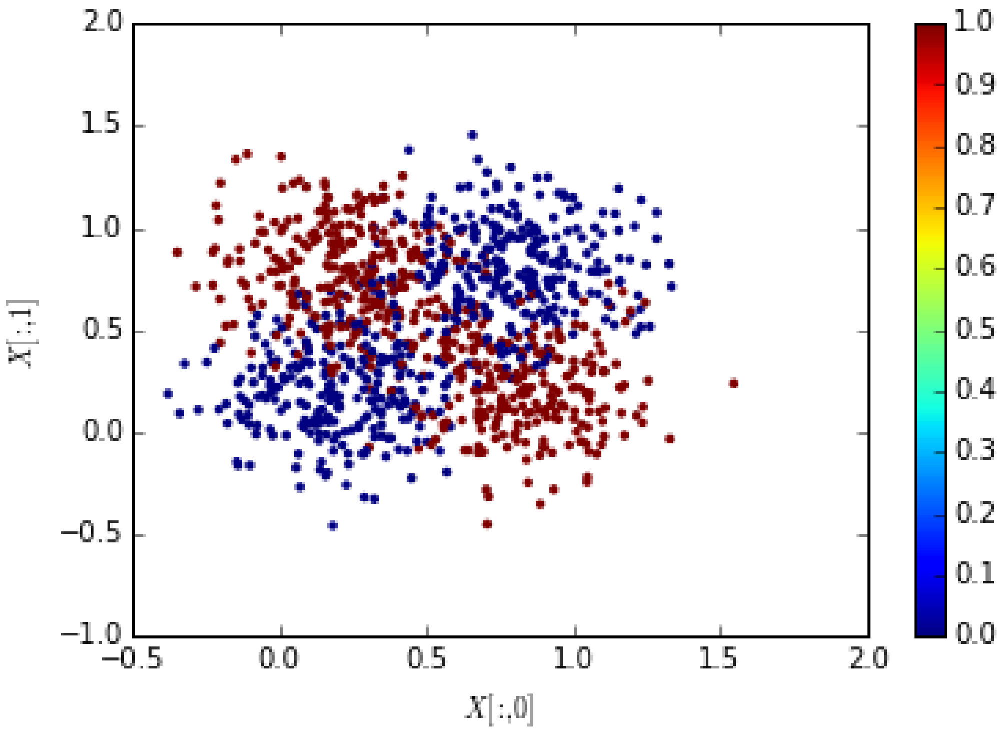

Figure 2.1: Two-class data.

2. In the IRLS algorithm, we modified the initialization of $\boldsymbol{w}$ and $\boldsymbol{\phi}$, and changed the algorithm a bit to be able to handle the new data as input. The resulting class probabilities are all 0.5:

$$p(\mathscr{C}_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}) \tag{2.14}$$

$$= \sigma\begin{pmatrix}0\\0\\0\end{pmatrix}^T\boldsymbol{\phi}) = \sigma(0) \tag{2.15}$$

$$= \frac{1}{1+\exp(0)} = \frac{1}{1+1} = 0.5 \tag{2.16}$$

3. See Figure 2.2. The probabilities range from 0.48 to 0.51, and therefore the colors in the plot are scaled to range from the minimum and maximum of the class probabilities, otherwise (ranging from 0 to 1) there is no visible difference in color for the data points. The initial cross entropy error with the initial weights was 693.14718, computed using the function in Listing 3. The cross entropy error only improves very slightly, and is now 692.96936. This is to be expected, since the class probabilities are all around 0.5 and thus the dummy basis function does not help differentiate between classes.

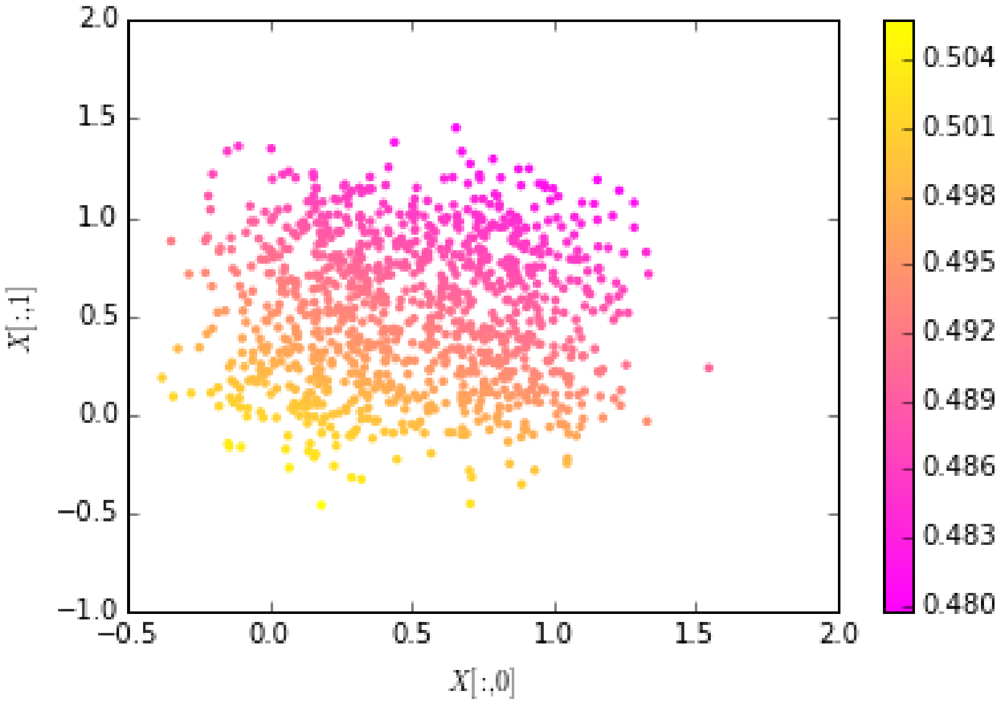

Figure 2.2: Data colored according to the class probabilities calculated using $\{x_1, x_2\}$ basis function and the dummy basis function.

Listing 3: Python code for function *cross_entropy_error(labels, y)*.

```python
def cross_entropy_error(labels,y):
    E = 0
    for label,_y in zip(labels,y):
        E += label * np.log(_y) + (1-label)*np.log(1-_y)

    return -E
```

4. We calculated ϕ_1 and ϕ_2 using the function shown in Listing 4, called with $\boldsymbol{\mu}_1 = (0, 0)$ for ϕ_1 and $\boldsymbol{\mu}_2 = (1, 1)$ for ϕ_2. The resulting plot of the data in the feature domain is shown in Figure 2.3. With these features, we think logistic regression can be a very good approach to classify the data, since the different classes are fairly linearly distinguishable in this domain.

Listing 4: Python code for function *gaussian_bf(x, mu)*.

```python
def gaussian_bf(x, mu):
    sigma = 0.2 * np.identity(2)
    return multivariate_normal.pdf(x,mu,sigma)
```

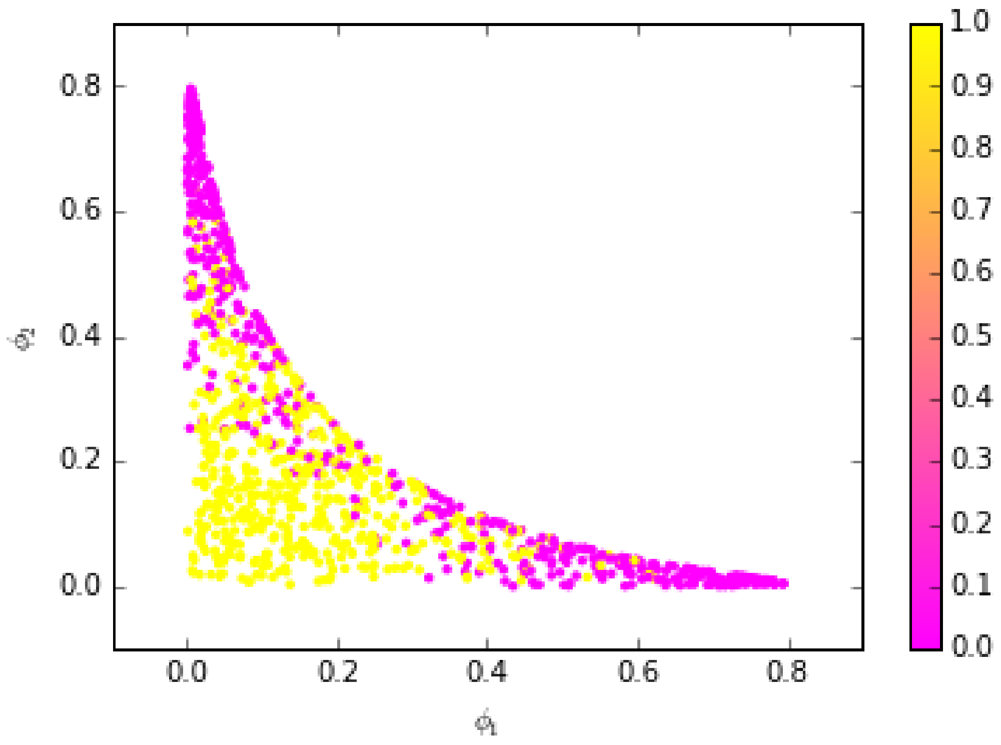

Figure 2.3: Data shown in the feature domain.

5. The cross entropy error improved to 346.50408, it became much smaller than before (692.96936). The error halfed more or less. By mapping the features to a different feature space (by using the two Gaussian basis functions as features) the data became much better linearly separable, although some values near the boundaries still have fairly uncertain probabilities of belonging to either class (see Figure 2.4).

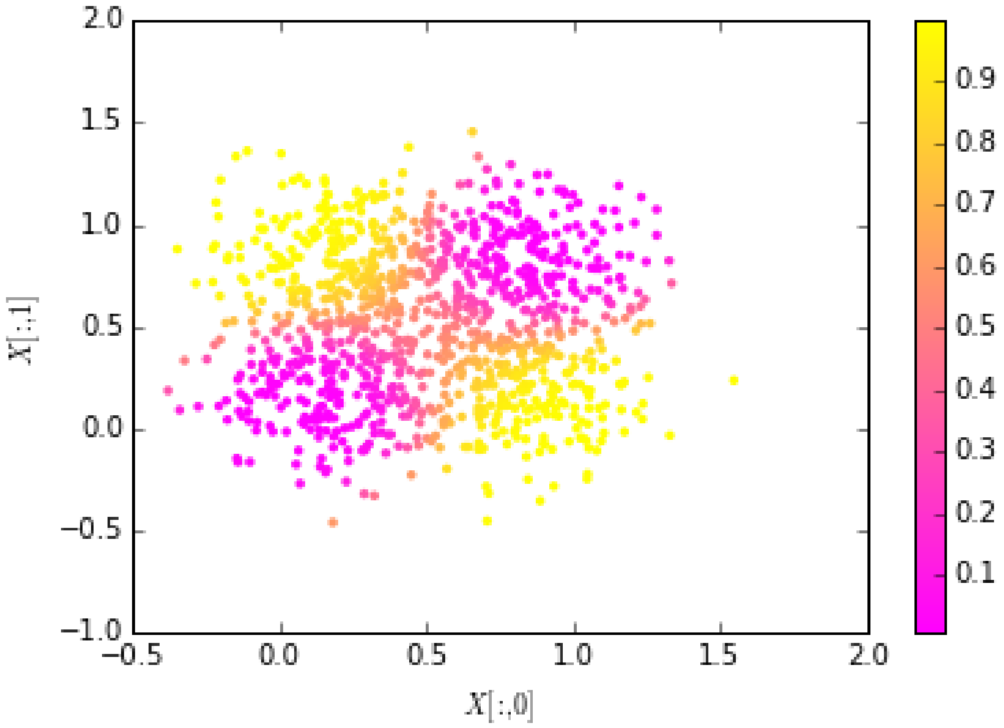

Figure 2.4: Data colored according to the class probabilities calculated using the Gaussian basis functions and the dummy basis function.