# Hw1

Kexin Feng 5165462

October 6, 2018

## 1 Problem 1:

(a)

$$\begin{aligned} E[l(f(x),y)] &= \int (f(x)-y)^2\, p(x,y)\mathrm{d}x\mathrm{d}y \\ &= \int (f(x)-E[y|x]+E[y|x]-y)^2\, p(x,y)\mathrm{d}x\mathrm{d}y \\ &= \int (f(x)-E[y|x])^2 p(x)\mathrm{d}x + \int \left[(E[y|x]-y)^2 p(y|x)\mathrm{d}y\right] p(x)\mathrm{d}x \\ \text{where } E[y|x] &= \int \mathrm{d}y\, p(y|x)y \end{aligned}$$

Note that, in the last line, $\left[(E[y|x]-y)^2 p(y|x)\mathrm{d}y\right] p(x)\mathrm{d}x$ represents the intrinsic random error of y, which cannot be affected by the choice of $f(x)$. Therefore, to minimize the expectation of loss function, $f(x) = E[y|x]$. This makes sense, because even though y is random, given an x, we can still take its expectation as the value of the function, that is why $f(x) = E[y|x]$ is the optimal.
(b)

$$\begin{aligned} l(f(x),y) &= |f(x)-y| \\ 0 = \frac{\delta E[f(x)]}{\delta f(x')} &= \frac{\delta \int sgn(f(x)-y)(f(x)-y)p(x,y)\mathrm{d}x\mathrm{d}y}{\delta f(x')} \\ &= \int sgn(f(x'-y))p(y,x')\mathrm{d}y \\ &= \int_{f(x')}^{\infty} p(y|x')p(x')\mathrm{d}y - \int_{-\infty}^{f(x')} p(y|x')p(x')\mathrm{d}y \\ \Rightarrow \int_{f(x')}^{\infty} p(y|x')\mathrm{d}y &= \int_{-\infty}^{f(x')} p(y|x')\mathrm{d}y \end{aligned}$$

Define $\Phi(y|x) \equiv \int_{-\infty}^{y} f(y|x)\mathrm{d}y$, then optimal $f(x)$ is such that $\Phi(f(x)|x) = 0.5$ i.e.

$$f(x) = \Phi^{-1}(0.5|x)$$

, which is Gaussian's median conditional on x.

## 2 Problem 2:

Below "$y = C_j$" is shorted as "$y_j$". The error rate for class y = $C_j$ means, for all the x that is from $C_j$, the rate of these x mistakenly falling into region $\mathcal{R}_i$ other than $\mathcal{R}_j$. Here $\mathcal{R}_i$ means a region where classifier assigns x to a class i rather than class j. To put it in sybols,

$$\mathcal{R}_i : p(y_i|x) > p(y_j|x),\ \forall i \neq j,\ \text{for a fixed } j$$

Therefore,

$$\begin{aligned} err[y = C_j] \equiv err(y_j) &= \sum_{i=1:M\&i\neq j} \int_{x\in\mathcal{R}_i} p(y_j, x)\mathrm{d}x \\ &= \sum_{i=1:M\&i\neq j} \int_{x\in\mathcal{R}_i} p(y_j|x)p(x)\mathrm{d}x \\ &= \sum_{i=1:M} \int_{x\in\mathcal{R}_i} p(y_j|x)p(x)\mathrm{d}x - \int_{x\in\mathcal{R}_j} p(y_j|x)p(x)\mathrm{d}x \end{aligned}$$

which is the same as what's is asked to prove.

# 3 Problem 3:

## 3.1 Algorithm:

(a) With two classes, it can only be projected to one dimensional line, I used 1-d Fisher linear discriminant. The objective function is

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

where

$$S_w = \sum_{n\in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{n\in C_2} (x_n - m_2)(x_n - m_2)^T \tag{1}$$

where $m_k$ are just the sample mean of each class. Then the optimal w is the following

$$w \propto S_w^{-1}(m_2 - m_1) \tag{2}$$

And the discriminant function is

$$y(x) = w^T(x - m) \begin{cases} > 0 & x \in C_2 \\ < 0 & x \in C_1 \end{cases}$$

since it's $(m_2 - m_1)$.

(b) No, not really. The reason is that we have only two classes. Like the center of the two classes can form only a line, we can only extract only one-dimensional difference between two classes. Generally, to classify K classes, we can only extract K-1 dimensional difference. so we can project it onto up to K-1 dimension space or less.

Technically, for K class, $S_B$ is of rank at most K-1, so $S_W^{-1}S_B$ if of at most rank K-1, thus has maximally K-1 non-zero eigen-vectors, which forms at most K-1 dimensional projected space.

Numerically, because of small numerical error, when diagonalizing $S_W^{-1}S_B$, there could still be small non-zero eigen value where it's supposed to be zero. Like projecting Boston50 onto 2d, see Fig. 1, that small eigen value could correspond to some direction, thus along with the previous non-trivial vector, form a two-dimensional plane to project the data on. However, the result shows that even in this two-dimensional plane, the projected points are still basically along a line, which verifies that with two classes there is only at most 1-d feature.

(c) With multiple classes, we need to use multi-dimensional Fisher linear discriminant. Try find the $W$ to minimize

$$J(w) = Tr\{s_w^{-1}s_B\} \tag{3}$$

We just need to find the eigen-vectors of $S_W^{-1}S_B$, $\{\vec{w}_j\}$ which form $W$ the projection coefficients. It can be shown mathematically that 1-d Fisher linear dicriminant is a special case of the derivation

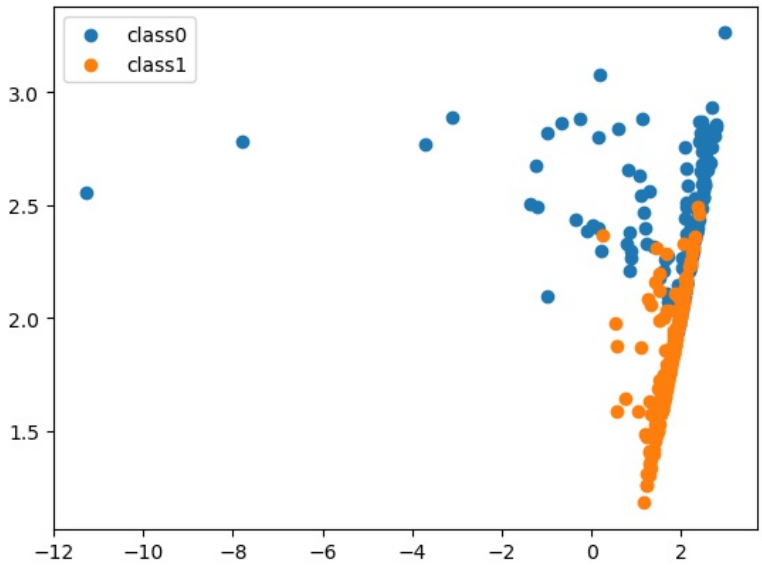


Figure 1: Boston50

here.

$$S_B = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T \tag{4}$$

$$S_W = \sum_{k=1}^{K} S_k \tag{5}$$

$$S_k = \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T \tag{6}$$

where $m_k$ are just the sample mean of each class, and $N_k$ is the number of sample in class k.

Then after projecting the data onto lower dimensional space, the following step is to use Gaussian generative model to classify them and predict for the class. This algorithm is exactly the same as Problem 4, Gaussian Naive Bayes(GNB). Please refer to there.

### 3.2 Experiment result:

I used ten-fold cross validation to test on both testing set and training set. This process is to divide the data into to ten subset, preferably with each set having similar class distribution. Then each time take one subset as testing set and the rest as training set, totally run 10 times.

(a) For Boston:
Testing set err rate:

$$0.08, 0.08, 0.06, 0.04, 0.14, 0.1, 0.02, 0.12, 0.12, 0.1$$

Standard deviation of testing set error rate:
0.03583
Mean of testing set error rate:
0.086

Training set err rate:

$$0.08, 0.08, 0.06, 0.04, 0.14, 0.1, 0.02, 0.12, 0.12, 0.1$$

Standard deviation of training set error rate:
0.035832
Mean of training set error rate:
0.086
So the error rate is pretty small which means this algorithm behaves well. The std is not big which means the algorithm is stable.

(c) For the digits set:
The result is the following:
This is used on the Digits data:
Testing set err rate:

$$0.14525, 0.1899, 0.13407, 0.26815, 0.20111, 0.19553, 0.23463, 0.050279, 0.150837, 0.11173$$

Standard deviation of testing set error rate:
0.05982
Mean of testing set error rate:
0.16815

Training set err rate:

$$0.1452513, 0.1899442, 0.13407821, 0.26815, 0.20111, 0.19553, 0.234636, 0.050279, 0.15083, 0.11173$$

Standard deviation of training set error rate:
0.059828
Mean of training set error rate:
0.168156
So the error rate is pretty small which means this algorithm behaves well. The std is not big which means the algorithm is stable.

# 4 Problem 4:

## 4.1 Algorithm:

**Native Bayes Gaussian generative model:** The key algorithm is based on the Bayes formula:

$$p(C_k|x) = \frac{exp(a_k)}{\sum_j exp(a_j)} \tag{7}$$

$$a_k = \ln p(x|C_k)p(C_k) \tag{8}$$

$p(C_k)$ are prior probability, which is estimated by $\pi_k = \frac{N_k}{N}$ in the code, where $N_k$ is the amount of data that is in class K.
$p(x|C_k)$ is the Gaussian dist. function that belongs to class k, i.e.

$$p(C_k) = \mathcal{N}(x|\mu_k, \Sigma_k)$$

, where $\mu_k$ and $\Sigma_k$ are estimated from the sample as below:

$$S_k = \frac{1}{N_k - 1} \sum_{n \in C_k} (x_n - \mu_k)(x_n - \mu_k)^T$$

$\mu_k$ are just the sample means.
The objective function is the minus log likelihood:

$$\sum_n^N t_n \ln \mathcal{N}(x_n|\mu, \Sigma) \tag{9}$$

**Logistic Regression:** The idea of LR is still based Bayes

$$p(C_k|x) = \frac{exp(a_k)}{\sum_j exp(a_j)} \tag{10}$$

but now $a_j$ instead of being a specific dist. function, like Gaussian, is modeled as $w^T x$, that is

$$a_k = \mathbf{w}_k^T \mathbf{x} \tag{11}$$

So the likelihood function is

$$p(\mathbf{T}|w1, \ldots, \mathbf{w_k}) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}} \tag{12}$$

where y is the the probability calculated from equation (10)
The objective function $E(w)$ is minus the log likelihood function which is the one to minimize.

$$E(w) = -\log(p(\mathbf{T}|w1, \ldots, \mathbf{w_k}))$$

I used iterative reweighted least squres,

$$w^{(new)} = w^{(old)} - \mathbf{H}^{-1} \nabla E(W) \tag{13}$$

$$\tag{14}$$

where $\mathbf{H}$ is the Hessian matrix of $E(w)$.

$$\nabla_{\mathbf{w}_k} \nabla_{\mathbf{w}_j} E(w) = \sum_{n=1}^{N} y_{nk}(\delta_{kj} - y_{nj}) \mathbf{x}_n \mathbf{x}_n^T \tag{15}$$

For multi-dimension this is going to be a 4th order tensor. To deal with this 4th order tensor product, I first obtain a 4th order matrix then based on this a direct producted version of this matrix can be obtained using the function like 'np.reshape' and high dimensional transpose 'np.transpose'. Correspondingly, $\nabla E(W)$ has to be represented in the direct product space as well.

## 4.2 Ng Jordan experiment:

First do 10 times 80-20 train-test split on the data. For each time, an increasing percentage of training set is used to train and then to calculate the error on the testing set. The increasing percentages are [10 25 50 75 100]. By averaging over the 10 times of split, we can get the mean error and the standard deviation of the error.
So we can see as training percentage increases the error rate decreases as expected.
GNB behaves better than LR on Digits meaning it's good at dealing with multi-class problem. LR is better than GNB at Boston, better at two-class problem. On digits set the result is contrary to the widely-held belief that discriminative classifier is preferable to generative classifier. Usually, discriminative model has lower error rate due to low bias ( large accommodation) yet it converges slower. It's also relevant to the converging condition, which will change the error rate. So the consistency analysis is required. For example, for the digits set, a longer time is taken, the LR method may converge to a more precise result, thus lower the error rate.

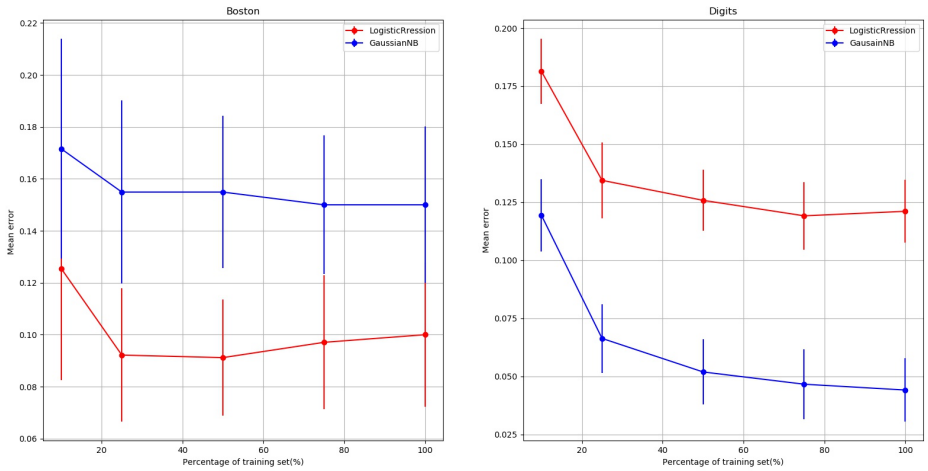


Figure 2: NgJordan experiment on Boston and Digits