# CSci 5525 Homework 1

Luis Guzman

Thursday October 1, 2020

1. a The optimal $f(x)$ will minimize our expected loss.

$$\begin{aligned} E_{(x,y)}[\ell(f(x),y)] &= \int_x \int_y \ell(f(x),y)p(x,y)dydx \\ &= \int_x \int_y (f(x)-y)^2 p(x,y)dydx \end{aligned}$$

Because the optimal $f(x)$ will minimize expected loss for every x, it is sufficient to minimize only the inside integral.

$$\arg\min_{f(x)} \int_y (f(x)-y)^2 p(x,y)dy$$

Since our loss function is convex, we can find this optimal function by setting the derivative of the expected loss equal to zero.

$$\begin{aligned} \frac{\partial E[\ell]}{\partial f(x)} &= 2\int (f(x)-y)p(x,y)dy = 0 \\ \int f(x)p(x,y)dy &= \int yp(x,y)dy \\ f(x)p(x) &= \int yp(x,y)dy \\ f(x) &= \frac{\int yp(y|x)p(x)dy}{p(x)} \\ &= \int yp(y|x)dy \\ &\boxed{f(x) = E_y[y|x]} \end{aligned}$$

This show that the $f(x)$ that minimizes our expected loss also gives us the expected y value for a given x. This is also known as the regression function, since it will be the best line fit for our data and this loss function.

b Similarly, for the loss function $\ell(f(x),y) = |f(x)-y|$,

$$\begin{aligned} E_{(x,y)}[\ell(f(x),y)] &= \int_x \int_y \ell(f(x),y)p(x,y)dydx \\ &= \int_x \int_y |f(x)-y|p(x,y)dydx \end{aligned}$$

Because the optimal $f(x)$ will minimize expected loss for every x, it is sufficient to minimize only the inside integral.

$$\arg\min_{f(x)} \int_y |f(x) - y| p(x, y) dy$$

We are considering only linear $f(x)$, so there will be only one zero-crossing of $f(x) - y$. We can split the absolute value into two parts.

$$\int_y |f(x) - y| p(x, y) dy = \int_{-\infty}^{f(x)} (f(x) - y) p(x, y) dy dx + \int_{f(x)}^{\infty} (y - f(x)) p(x, y) dy dx$$

Solving for $f(x)$ that minimizes loss,

$$\begin{aligned} \frac{\partial E[\ell]}{\partial f(x)} &= \int_{-\infty}^{f(x)} p(x, y) dy - \int_{f(x)}^{\infty} p(x, y) dy = 0 \\ \int_{-\infty}^{f(x)} p(y|x) dy &= \int_{f(x)}^{\infty} p(y|x) dy \\ &\boxed{f(x) = \text{Median}(y|x)} \end{aligned}$$

By definition, $f(x)$ must be the median of $(y|x)$ since the probability of having a data point above or below $f(x)$ is equal.

2. Claim: The least squares problem

$$\min_w ||\boldsymbol{Aw} - \boldsymbol{b}||_2^2 + \boldsymbol{c}^T \boldsymbol{w} + d$$

can be written as

$$\min_{\boldsymbol{w}} ||\boldsymbol{Aw} - \boldsymbol{b} + \boldsymbol{f}||_2^2 + g$$

Proof:

$$\begin{aligned} ||\boldsymbol{Aw} - \boldsymbol{b} + \boldsymbol{f}||_2^2 + g &= (\boldsymbol{Aw} - \boldsymbol{b} + \boldsymbol{f})^T (\boldsymbol{Aw} - \boldsymbol{b} + \boldsymbol{f}) + g \\ &= ((\boldsymbol{Aw} - \boldsymbol{b})^T + \boldsymbol{f}^T)(\boldsymbol{Aw} - \boldsymbol{b} + \boldsymbol{f}) + g \\ &= (\boldsymbol{Aw} - \boldsymbol{b})^T (\boldsymbol{Aw} - \boldsymbol{b}) + (\boldsymbol{Aw} - \boldsymbol{b})^T \boldsymbol{f} + \boldsymbol{f}^T (\boldsymbol{Aw} - \boldsymbol{b}) + \boldsymbol{f}^T \boldsymbol{f} + g \\ &= (\boldsymbol{Aw} - \boldsymbol{b})^T (\boldsymbol{Aw} - \boldsymbol{b}) + \boldsymbol{c}^T \boldsymbol{w} + d \end{aligned}$$

The last four terms are linear in w, so it can be written as $\boldsymbol{c}^T \boldsymbol{w} + d$. To find out these constants, we must expand the sum further. The last four terms are repeated below:

$$\begin{aligned} \boldsymbol{c}^T \boldsymbol{w} + d &= (\boldsymbol{Aw} - \boldsymbol{b})^T \boldsymbol{f} + \boldsymbol{f}^T (\boldsymbol{Aw} - \boldsymbol{b}) + \boldsymbol{f}^T \boldsymbol{f} + g \\ &= (\boldsymbol{Aw})^T \boldsymbol{f} - \boldsymbol{b}^T \boldsymbol{f} + \boldsymbol{f}^T \boldsymbol{Aw} - \boldsymbol{f}^T \boldsymbol{b} + \boldsymbol{f}^T \boldsymbol{f} + g \\ &= \boldsymbol{w}^T \boldsymbol{A}^T \boldsymbol{f} - \boldsymbol{b}^T \boldsymbol{f} + \boldsymbol{f}^T \boldsymbol{Aw} - \boldsymbol{f}^T \boldsymbol{b} + \boldsymbol{f}^T \boldsymbol{f} + g \\ &= \boldsymbol{w}^T (\boldsymbol{A}^T \boldsymbol{f}) + (\boldsymbol{A}^T \boldsymbol{f})^T \boldsymbol{w} - \boldsymbol{b}^T \boldsymbol{f} - \boldsymbol{f}^T \boldsymbol{b} + \boldsymbol{f}^T \boldsymbol{f} + g \\ &= 2(\boldsymbol{A}^T \boldsymbol{f})^T \boldsymbol{w} - 2\boldsymbol{b}^T \boldsymbol{f} + \boldsymbol{f}^T \boldsymbol{f} + g \\ &\boxed{\boldsymbol{c} = 2(\boldsymbol{A}^T \boldsymbol{f})} \\ &\boxed{g = -2\boldsymbol{b}^T \boldsymbol{f} + \boldsymbol{f}^T \boldsymbol{f} + g} \end{aligned}$$

Therefore $||\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b} + \boldsymbol{f}||_2^2 + g$ can be written as $||\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}||_2^2 + \boldsymbol{c}^T\boldsymbol{w} + d$ and we can solve for $\boldsymbol{w}$ by finding a stationary point.

$$\begin{aligned}
\boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} ||\boldsymbol{A}\boldsymbol{w} - (\boldsymbol{b} - \boldsymbol{f})||_2^2 \\
0 &= \nabla||\boldsymbol{A}\boldsymbol{w} - (\boldsymbol{b} - \boldsymbol{f})||_2^2 \\
&= \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{w} - (\boldsymbol{b} - \boldsymbol{f})) \\
\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{w} &= \boldsymbol{A}^T(\boldsymbol{b} - \boldsymbol{f}) \\
&\boxed{\boldsymbol{w}^* = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T(\boldsymbol{b} - \boldsymbol{f})}
\end{aligned}$$

This agrees with what we would expect from the typical least squares solution.

3. (i) For this question, I implemented Fisher's linear discriminant analysis (LDA) on the Boston50 dataset. The algorithm calculates the within-class ($S_W$) and between-class ($S_B$) variance, and then projects the data to a lower dimension for classification. The dimension we choose to project onto is chosen to maximize the between-class variance and minimize the within class variance. $S_B$ and $S_W$ are defined as:

$$\begin{aligned}
S_B &= (m_2 - m_1)(m_2 - m_1)^T \\
S_W &= \sum_{x_n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{x_n \in C_2} (x_n - m_2)(x_n - m_2)^T
\end{aligned}$$

where $C_1, C_2$ are the two class labels and $m_1, m_2$ are the mean feature vectors of each class. Fisher's criterion aims to maximize

$$\begin{aligned}
J\,|w\rangle &= \frac{||w^T(m_2 - m_1)||^2}{w^T(\sum_{x_n \in C_1}(x_n - m_1)(x_n - m_1)^T + \sum_{x_n \in C_2}(x_n - m_2)(x_n - m_2)^T)w} \\
&= \frac{w^T S_B w}{w^T S_W w}
\end{aligned}$$

So we can find the optimal $w$ as

$$w = S_W^{-1}(m_2 - m_1)$$

I ran 10-fold cross-validation, and calculated the mean error rate of each fold. The threshold for class separation was chosen by testing 10 values between the minimum values of the orange class and the maximum value of the blue class. The value corresponding to the lowest training error was then chosen as the threshold. Below are my results and a few selected histograms that show the 13-dimensional features projected down to one dimension.

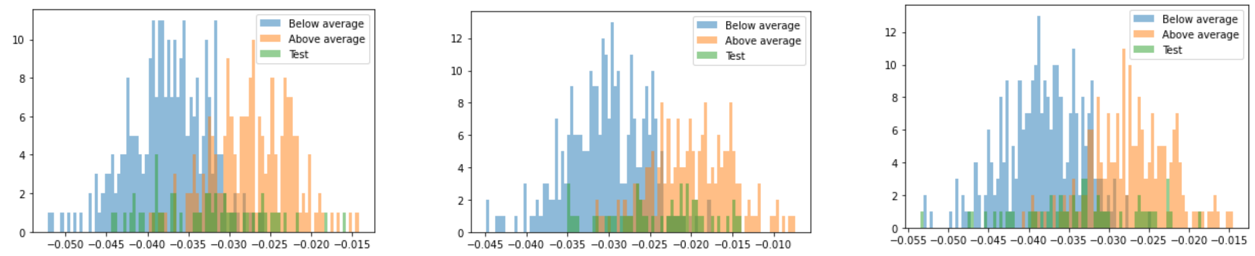

Figure 1: Images showing the separation of the two classes for three of the cross-validation stages.

| Fold | Training Error rate | Testing Error rate |
|---|---|---|
| 0 | 13.15 | 14.00 |
| 1 | 11.84 | 12.00 |
| 2 | 11.84 | 14.00 |
| 3 | 12.15 | 16.00 |
| 4 | 17.32 | 18.00 |
| 5 | 14.25 | 12.00 |
| 6 | 15.13 | 10.00 |
| 7 | 12.06 | 14.00 |
| 8 | 13.59 | 10.00 |
| 9 | 12.72 | 10.00 |
| Average | 13.41 | 13.00 |
| St. Dev. | 1.76 | 2.57 |

The average testing error rate was 13.00% and the standard deviation was 2.57%

(ii) Part two of this question consisted of applying LDA to the Digits dataset, and projecting 64-dimensional data onto two dimensions. In this case, the equation $w = S_W^{-1}(m_2 - m_1)$ is not sufficient because we have more than two classes and $w$ will be a 2D vector. For the 2D case, I redefine $S_B$ and $S_W$ as

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

$$S_W = \sum_{k=1}^{K} \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T$$

The $w$ that we want to project onto will then be a matrix consisting of the eigenvectors corresponding to the two largest eigenvalues of $S_W^{-1}S_B$. We can create our $w$ matrix by stacking these eigenvectors side-by-side. The data, projected down to two dimensions, is then

$$X_{\text{proj}} = Xw$$

Although not perfectly separated (due to possible bugs/mistakes in the code), the resulting image shows some striping pattern in the data, which should allow us to perform some classification through Gaussian generative modeling.

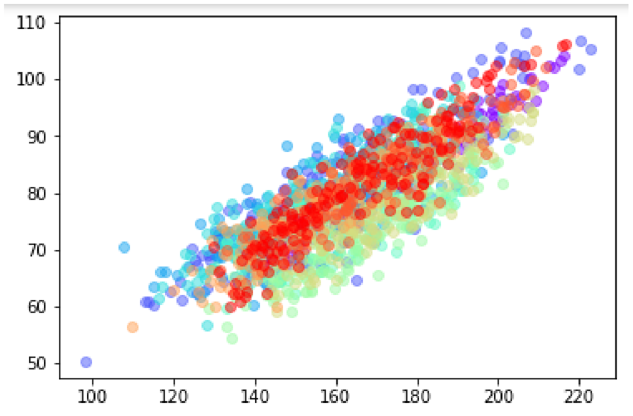

Figure 2: Image showing the Digits dataset after it has been projected to 2 dimensions from 64

To perform the classification, I calculate a multivariate Gaussian PDF for each class using the class mean and covariance matrix:

$$p(x|y) = \frac{1}{2\pi\sqrt{|\Sigma_y|}} \exp\Big( -\frac{1}{2}(x-\mu_y)^T \Sigma_y^{-1} (x-\mu_y) \Big)$$

where $\mu_y$ and $\Sigma_y$ are the mean and covariance of the class $y$. The $2\pi$ term is also not inside the square root since our distribution has two dimensions. We can then predict the class label probabilities with Bayes rule (explained more in-depth in question 4(ii)):

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

Although I believe the method I followed is correct, I am not getting error rate that much better than random guessing, so I suspect some issues with the code.

| Fold | Training Error rate |
|---|---|
| 0 | 92.73 |
| 1 | 89.38 |
| 2 | 87.71 |
| 3 | 88.27 |
| 4 | 91.06 |
| 5 | 91.62 |
| 6 | 89.39 |
| 7 | 90.50 |
| 8 | 89.94 |
| 9 | 89.94 |
| Average | 90.01 |
| St. Dev. | 1.51 |

4. In this problem, I aim to reproduce the results of the paper "On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes" by A. Ng and M. Jordan. I implemented and trained the Logistic Regression and Naive Bayes classifiers on the Boston50, Boston75 and Digits datasets.

   (i) Logistic Regression

   Logistic regression uses the logistic loss function and assumes that the log-odds rule is linear in x:

$$\ell(f(x), y) = \log(1 + \exp(-y_i w^T x))$$
$$\log\Big(\frac{P(1|x)}{P(0|x)}\Big) = w^T x$$

   From the above equation, we can calcuate the probabilities that $x$ belongs to a given class.

$$P(1|x) = \frac{\exp(w^T x)}{1+\exp(w^T x)} = \sigma(w^T x)$$
$$P(0|x) = \frac{1}{1+\exp(w^T x)} = 1 - \sigma(w^T x)$$

where $\sigma$ is defined as the sigmoid function. The $w$ that we want to use for our classifier minimizes the loss function.

$$w^* = \arg\min_w \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i w^T x))$$

$$\nabla_w E(w^T) = \nabla_w \sum_{i=1}^{n} \log(1+\exp(-y_i w^T x)) = 0$$

This has no closed-form solution, so we can solve with gradient descent.

$$w_{t+1} = w_t - \alpha_t \nabla E(w_t)$$

where $\alpha_t$ is our learning rate or "step size".

The paper of interest compares how testing error rates vary with increased size of the training set. First, I split off 20% of the data for testing. I then trained logistic regression on five training sets, consisting of 10, 25, 50, 75, and 100 percent of the regaining data. I did this 10 times on randomly shuffled data and averaged the scores across each fold. Figure 3 shows the result.

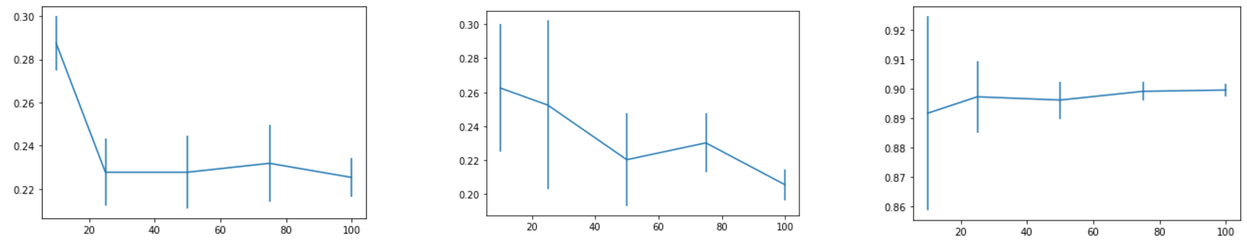

Figure 3: Images showing the logistic training error vs. number of samples for the Boston50, Boston75, and Digits datasets (from left to right)

I was unfortunately not able to complete this algorithm for the Digits dataset because my method would require significant changes to work for multiclass classification. The error rate for this dataset is around 90% (equal to random guessing). An alternative method could be one-vs-one or one-vs-all classification, which would not require changing the math, but I did not have time to implement these.

(ii) Naive-Bayes with marginal Gaussian distributions (GNB)

The Naive-Bayes classifier assumes conditional independence to simplify classification calculations. Conditional independence states that

$$p(\boldsymbol{x}|C_k) = \prod_{i=1}^{p} p(x_i|C_K)$$

As long as we know what $p(x_i|C_K)$ is for every data point and class, we can use Bayes rule to calculate the probability that a new data point belongs to a given class.

$$\begin{aligned} p(y|x) &= \frac{p(x|y)p(y)}{p(x)} \\ &\propto p(x|y)p(y) \end{aligned}$$

For this assignment, I assume that the data follows a Gaussian distribution. In the algorithm, I split the data by class label and calculate the mean and standard deviation of each column (feature) independently. Then the probability of a given $x$ value is

$$p(x|y) = \mathcal{N}(x, \mu_y, \sigma_y)$$

where $\mu_y$ and $\sigma_y$ are the values mean and standard deviation values for a given class. The prior probability $p(y)$ is simply the number of instances of that class divided by the total number of samples. The predicted value is then just the label with the highest probability according to Bayes rule

$$\hat{y} = \arg\max_y p(x|y)p(y)$$

I ran the Naive Bayes classifier on the three datasets, averaging over 10 random 80-20 splits to calculate the error values. Five different training set sizes were compared: 10, 25, 50, 75, and 100 percent of the total available training data. The following graphs compare the error rate as more data is used to train the model:

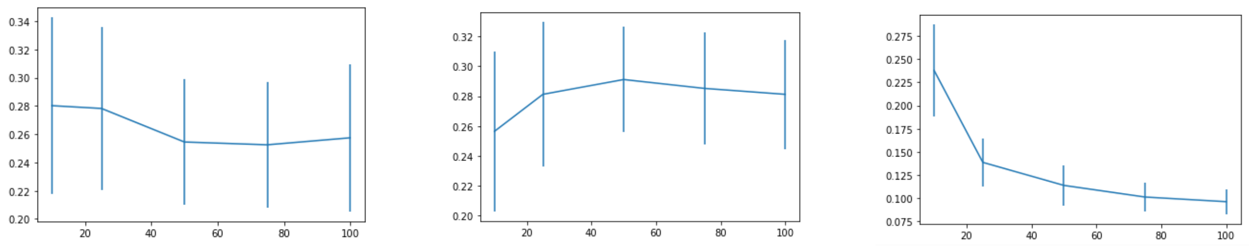

Figure 4: Images showing the bayes training error vs. number of samples for the Boston50, Boston75, and Digits datasets (from left to right)

My result for the Boston datasets surprised me since they did not match what was in the paper (more similar to the Digits graph). I'm not sure whether this was an issue with my code or if the dataset is not as sensitive to the size of the training set. The Digits dataset, on the other hand, matches quite well with what is presented in the paper.