Homework 2

Kayla Straub
3/6/2015
ECE 5984: Introduction to Machine Learning

1. Learning with the L1 Norm, L1 Error

1. Since $\epsilon_i \sim N(0, \sigma^2)$, we can say that $(x_i - y)$ is also i.i.d. Gaussian, $\sim N(0, \sigma^2)$. Therefore, we can write that:

$$p(\epsilon_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\epsilon_i^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i - y)^2}{2\sigma^2}} = p(x_i|y)$$

And because of independence, we can write the likelihood function as:

$$p(x|y) = \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i - y)^2}{2\sigma^2}} = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{\Sigma_{i=1}^{n}(x_i - y)^2}{2\sigma^2}}$$

And the log likelihood function as:

$$\ln\big(p(x|y)\big) = -n \ln\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} \Sigma_{i=1}^{n} (x_i - y)^2$$

Therefore,

$$\hat{y}_{MLE} = \max_y \big(p(x|y)\big) = \max_y \left(-n \ln\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} \Sigma_{i=1}^{n} (x_i - y)^2\right)$$
$$= \max_y -\frac{1}{2\sigma^2} \Sigma_{i=1}^{n} (x_i - y)^2$$

Ignoring the constant shows that the maximum likelihood estimator for Y is indeed equivalent to finding the value of $\hat{y}$ which minimizes the sum of least squares error to the x's.

$$\hat{y}_{MLE} = \min_y \Sigma_{i=1}^{n} (y - x_i)^2$$

The minimum of this expression is found by calculating the partial derivative with respect to y and setting that equal to zero:

$$\frac{\partial}{\partial y} \Sigma_{i=1}^{n} (y - x_i)^2 = \Sigma_{i=1}^{n} 2y - 2x_i = 0 \rightarrow yn - \Sigma_{i=1}^{n} x_i = 0$$

$$\hat{y}_{MLE} = \frac{1}{n} \Sigma_{i=1}^{n} x_i$$

Note that $\hat{y}_{MLE}$ is simply the mean of the x samples.

2. Just as in the previous problem, we can write that:

$$p(\epsilon_i) = \frac{1}{2b} e^{-\frac{|\epsilon_i|}{b}} = \frac{1}{2b} e^{-\frac{|x_i - y|}{b}} = p(x_i|y)$$

And because of independence, we can write the likelihood function as:

$$p(x|y) = \Pi_{i=1}^{n} \frac{1}{2b} e^{-\frac{|x_i - y|}{b}} = \left(\frac{1}{2b}\right)^n e^{-\frac{\sum_{i=1}^{n}|x_i - y|}{b}}$$

And the log likelihood function as:

$$\ln\big(p(x|y)\big) = -n\ln(2b) - \frac{1}{b}\sum_{i=1}^{n}|x_i - y|$$

Therefore,

$$\hat{y}_{MLE} = \max_y \big(p(x|y)\big) = \max_y \left(-n\ln(2b) - \frac{1}{b}\sum_{i=1}^{n}|x_i - y|\right) = \max_y -\frac{1}{b}\sum_{i=1}^{n}|x_i - y|$$

Ignoring the constant shows that the MLE for Y is equivalent to finding the value $\hat{y}$ that minimizes the sum of absolute errors:

$$L(y) = \sum_{i=1}^{n}|y - x_i|$$

$$\hat{y}_{MLE} = \min_y \sum_{i=1}^{n}|y - x_i| \quad \blacksquare$$

3. Assuming $\forall i, \forall j > i, x_j > x_i$ and $x_i < y < x_{i+1}$, we can find an expression for the gradient. Say $x_j < y < x_{j+1}$, then

$$\frac{dL(y)}{dy} = \underbrace{\frac{d}{dy}\sum_{i=1}^{j} y - x_i}_{y > x_j} + \underbrace{\frac{d}{dy}\sum_{i=j+1}^{n} -(y - x_i)}_{y < x_j} = \sum_{i=1}^{j} 1 + \sum_{i=j+1}^{n} -1 = j + (j - n) = 2j - n$$

where j represents the number of $x < y$ and $n - j$ represents the number of $x > y$. Therefore another way to interpret the gradient is:

$$\boxed{\frac{dL(y)}{dy} = (\#x < y) - (\#x > y)}$$

4. Assuming the number of $x's$ is even, the gradient is 0 when the number of x values less than y is equal to the number of x values greater than y. If there are n values of x, then the values for y such that $\frac{dL(y)}{dy} = 0$ are:

$$x_{\frac{n}{2}} < y < x_{\frac{n}{2}+1}$$

5. If the number of $x's$ is odd, then the sign of the gradient changes when you reach the median x value. If there are n values of x, you can

$$\frac{dL(y)}{dy} < 0 \; for \; y < x_{\frac{n+1}{2}} \; and \; \frac{dL(y)}{dy} > 0 \; for \; y > x_{\frac{n+1}{2}}$$

Therefore, $\boxed{y_0 = x_{\frac{n+1}{2}}}$.

6. Putting together the answers from the previous 2 sections, it is clear that $\hat{y}$ represents the median of the x values. This method is much more robust to outliers in the data because only the middle value(s) are used and therefore outliers are not included in the estimate. For example, if x is given as:

$$x = \{1, 2, 3, 4, 100\}$$

Then the sum of absolute error estimate would predict $\hat{y} = 3$. In contrast, the least squares estimator would predict $\hat{y} = 22$, which clearly shows the effect of the outlier.

2. Leave One Out Cross-Validation with Least Squares

1. If we define

$$X^T = [\boldsymbol{x_1} \; \boldsymbol{x_2} \; \boldsymbol{x_3} \; \ldots \boldsymbol{x_n}]$$

Then we can write that:

$$HY = X(X^TX)^{-1}X^TY = X\boldsymbol{w} = \; (\boldsymbol{w}^TX^T)^T = \hat{Y} \; \blacksquare$$

2. To find the complexity of computing the LOOCV error, I first found the error to compute $\hat{y}^{-i}$. From the previous problem, we learned that

$$\hat{Y}^{-i} = HY^{-i} = X^{-i}\left(X^{-i^T}X^{-i}\right)^{-1}X^{-i^T}Y^{-i}$$

Where X^{-i} is a (N-1xD) matrix and Y^{-i} is a (Nx1) matrix. I used the following two rules to calculate the complexity of this equation

1. The complexity of inverting a DxD matrix is $O(D^3)$
2. The complexity of multiplying a MxN vector with a NxP vector is O(MNP)

By using these rules and ignoring the smaller terms, I calculated the complexity of finding $\hat{Y}^{-i}$ as $O(D^3 + D^2N + DN^2)$. The complexity of the subtractions and the additions were also negligible, but what is not negligible is that $\hat{Y}^{-i}$ is computed N times, once for each term in the summation. Therefore the overall complexity of finding the LOOCV error is:

$$O\big(N(D^3 + D^2N + DN^2)\big) = O(D^3N + D^2N^2 + DN^3) \approx O(D^3N + DN^3)$$

Based on the relative sizes of N and D, we can simplify this further to say the complexity of finding the LOOCV error is:

$$\boxed{if\ D > N, O(D^3N)\ else, O(DN^3)}$$

3. We can rewrite the OLS regressor as:

$$\sum_{i=1}^{N}(p_i - z_i)^2 = \left(p_k - \hat{y}_k^{-k}\right)^2 + \sum_{i=1, i\neq k}^{N}(p_i - y_i)^2$$

If $p_i = \hat{y}_i^{-k}$, then the first term equals zero and we are left with $\sum_{i=1,i\neq k}^{N}(p_i - y_i)^2$, which is just the OLS regressor with x_k left out. We proved in a previous problem that this was a minimum over p. Therefore the given summation is minimized when $p_i = \hat{y}_i^{-k}$.

4. We learned above that $\hat{Y} = HY$. We can also show that $\hat{Y} = H\hat{Y}$:

$$H\hat{Y} = HHY = X\underbrace{(X^TX)^{-1}X^TX}_{=I}(X^TX)^{-1}X^TY = X(X^TX)^{-1}X^TY = HY = \hat{Y}$$

Therefore, we can rewrite the prediction vector as follows:

$$\vec{p} = \left[\hat{y}_1^{-k}, \dots, \hat{y}_k^{-k}, \dots, \hat{y}_N^{-k}\right] = H[y_1^{-k}, \dots, \hat{y}_k^{-k}, \dots, y_N^{-k}]$$

Where y_i^{-k} represents a label for the original training data. Therefore, $y_i^{-k} = y_i\ \forall\ i \neq k$. This gives:

$$\vec{p} = H\left[y_1, \dots, \hat{y}_k^{-k}, \dots, y_N\right] = HZ \quad \blacksquare$$

For $p_k = \hat{y}_k^{-k}$, we can expand this using the property of $\hat{Y} = H\hat{Y} = HY$ described above:

$$p_k = \hat{y}_k^{-k} = \overrightarrow{h_k}\hat{y}^{-k} = \overrightarrow{h_k}Z$$

Where $\overrightarrow{h_k}$ is the k^{th} row of H.

5. As shown above, $\hat{y}_k^{-k} = \vec{h}_k Z$. We can expand this as follows:

$$\hat{y}_k^{-k} = \vec{h}_k Z = H_{k1}z_1 + H_{k2}z_2 + \cdots H_{kN}z_N = H_{k1}y_1 + \cdots + H_{kk}\hat{y}_k^{-k} + \cdots + H_{kN}y_N$$

$$\hat{y}_k - \hat{y}_k^{-k} = \underbrace{\left(\sum_{i=1, i\neq k}^{N} (H_{ki}y_i) + H_{kk}y_k\right)}_{\hat{y}_k} - \underbrace{\left(\sum_{i=1, i\neq k}^{N} (H_{ki}y_i) + H_{kk}\hat{y}_k^{-k}\right)}_{\hat{y}_k^{-k}} = H_{kk}y_k - H_{kk}\hat{y}_k^{-k} \blacksquare$$

6. We can rearrange the result from the previous problem as follows:

$$\hat{y}_k - \hat{y}_k^{-k} = H_{kk}y_k - H_{kk}\hat{y}_k^{-k}$$

$$\hat{y}_k - H_{kk}y_k = \hat{y}_k^{-k} - H_{kk}\hat{y}_k^{-k} = \hat{y}_k^{-k}(1 - H_{kk})$$

$$\hat{y}_k^{-k} = \frac{\hat{y}_k - H_{kk}y_k}{1 - H_{kk}}$$

We can then substitute this into our original equation for LOOCVerror:

$$LOOCVerror = \sum_{k=1}^{N}\left(y_k - \hat{y}_k^{-k}\right)^2 = \sum_{k=1}^{N}\left(y_k - \frac{\hat{y}_k - H_{kk}y_k}{1 - H_{kk}}\right)^2$$
$$= \sum_{k=1}^{N}\left(\frac{y_k - H_{kk}y_k - \hat{y}_k + H_{kk}y_k}{1 - H_{kk}}\right)^2 = \sum_{k=1}^{N}\left(\frac{y_k - \hat{y}_k}{1 - H_{kk}}\right)^2 \blacksquare$$

Note that this summation only requires $\hat{y}_k$ instead of $\hat{y}_k^{-k}$. This means that instead of finding a new prediction vector $\hat{Y}$ once for each term as we leave out, we only need to find a single $\hat{Y}$. From before, we found the complexity to calculate $\hat{Y}$ was $O(D^3 + D^2N + DN^2)$. For each of the N iterations there is also 2 subtractions, 1 division, and 1 multiplication. However, this is just 4N so it does not change the order of the complexity. Similar to before, we can simplify the complexity of finding the LOOCVerror to:

$$\boxed{if\ D > N, O(D^3)\ \ else, O(DN^2)}$$

7. Through this exercise, we found a clever way to decrease the algorithmic complexity of finding the LOOCVerror by N. For example, if we have 50 training points (N=50) and the original calculation took 25 minutes, this improved method would only take 30 seconds. Training the classifier to find the weight vector is part of the process to find the

LOOCVerror. Therefore, after the classifier has already been trained, it requires minimal additional operations to find the LOOCVerror.

8. In Matlab, I implemented both the naïve LOOCVerror calculations and the faster method derived in the previous questions. Both of the functions calculated the same LOOCVerror:

$$\boxed{LOOCVerror = 3.3088\ MPG}$$

To compare the time it took each function to run, I used the `timeit` function. Timing data can be hard to measure because it has a high variance, so I averaged the time for 100 function calls for each function. The average time for the slow function was **0.0115 seconds.** The average time for the fast function was $\mathbf{1.6895\times10^{-4}}$ **seconds.** This gives an average speedup of about 68 times. This is slightly smaller than the anticipated N-times speedup (93x in this case). This is probably because calculating the H matrices is a little bit faster for the 'slow' function because it has one less training point to consider.

3. Faster Matrix Inversion

1. We know that the H matrix is defined as

$$H = X(X^TX)^{-1}X^T$$

We can begin with the inner term. The non-diagonal terms of (X^TX) are all of the form $\sum_{i=1}^{N} x_{id}x'_{id}$, so therefore they are equal to 0. The diagonal terms of this matrix are of the form $\sum_{i=1}^{N} x_{im}^2$ where m is the row number. The inverse of this matrix is therefore:

$$(X^TX)^{-1} = \begin{bmatrix} \frac{1}{\sum_{i=1}^{N} x_{i1}^2} & 0 & 0 & 0 \\ 0 & \frac{1}{\sum_{i=1}^{N} x_{i2}^2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{1}{\sum_{i=1}^{N} x_{iD}^2} \end{bmatrix}$$

If we call $\frac{1}{\sum_{i=1}^{N} x_{im}^2} = a_m$, we can then write the product of $X(X^TX)^{-1}$ as:

$$X(X^TX)^{-1} = \begin{bmatrix} x_{11} & \cdots & x_{D1} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{ND} \end{bmatrix} \begin{bmatrix} a_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_D \end{bmatrix} = \begin{bmatrix} a_1x_{11} & a_2x_{12} & \cdots & a_Dx_{1D} \\ a_1x_{21} & a_2x_{22} & \dots & a_Dx_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ a_1x_{N1} & a_2x_{N2} & \cdots & a_Dx_{ND} \end{bmatrix}$$

This is almost the full calculation of H. We can then find an expression for H_{kk} as:

$$H_{kk} = diag(X(X^TX)^{-1}X^T) = diag\left(\begin{bmatrix} a_1x_{11} & a_2x_{12} & \cdots & a_Dx_{1D} \\ a_1x_{21} & a_2x_{22} & \dots & a_Dx_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ a_1x_{N1} & a_2x_{N2} & \cdots & a_Dx_{ND} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{N1} \\ \vdots & \ddots & \vdots \\ x_{D1} & \cdots & x_{ND} \end{bmatrix}\right)$$

From this we can see that $H_{11} = \sum_{d=1}^{D} a_d x_{1d}^2$, $H_{22} = \sum_{d=1}^{D} a_d x_{2d}^2$, etc. We can generalize this to $H_{kk} = \sum_{d=1}^{D} a_d x_{kd}^2$. Substituting back in the definition for a_d for each row from 1 to D gives:

$$H_{kk} = \sum_{d=1}^{D} \frac{x_{kd}^2}{\sum_{i=1}^{N} x_{id}^2} \quad \blacksquare$$

2. Since in this case D = 1, we can rewrite $(X^TX)^{-1}$ using the result above as follows (as long as the sum of the $x's \neq 0$) :

$$(X^TX)^{-1} = \frac{1}{\sum_{i=1}^{N} x_i^2}$$

And we can use this result to rewrite w^{OLS} as:

$$w^{OLS} = (X^TX)^{-1}X^TY = \frac{1}{\sum_{m=1}^{N} x_i^2} X^TY = \frac{\sum_{j=1}^{N} x_j y_j}{\sum_{m=1}^{N} x_m^2}$$

We can then substitute the residual error equation into the given summation:

$$\sum_{i=1}^{N} e_i x_i = \sum_{i=1}^{N} (y_i - w^{OLS}x_i)x_i = \sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} w^{OLS} x_i^2$$

And then substitute the equation for w^{OLS}:

$$= \sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i^2 \frac{\sum_{j=1}^{N} x_j y_j}{\sum_{m=1}^{N} x_m^2}$$

We can see that $\sum_{i=1}^{N} x_i^2 = \sum_{m=1}^{N} x_m^2$ and $\sum_{i=1}^{N} x_i y_i = \sum_{j=1}^{N} x_j y_j$, which leads to:

$$\sum_{i=1}^{N} e_i x_i = \sum_{i=1}^{N} x_i y_i - \sum_{j=1}^{N} x_j y_j = 0 \quad \blacksquare$$

3. We can transform each column of X such that it is uncorrelated with all other columns of X so that we can apply the above calculation simplification. This process is similar to the Gram-Schmidt process for finding unique basis functions for a signal set.

This problem is analogous to the previous derivation. We want to predict $u_2$using z_1 using the following formulas:

$$\gamma_{21} = (z_1^T z_1)^{-1} z_1^T u_2 \rightarrow z_2 = u_2 - \gamma_{21} z_1$$

We can rewrite γ_{21} just as we did for w^{OLS} in the previous problem:

$$\gamma_{21} = \frac{\sum_{j=1}^{N} z_{1j} u_{2j}}{\sum_{m=1}^{N} z_{1m}^2}$$

We can then generalize this to:

$$\gamma_{ba} = \frac{\sum_{j=1}^{N} z_{aj} u_{bj}}{\sum_{m=1}^{N} z_{am}^2}$$

Using these equations, we can rewrite z_2 and z_3:

$$z_2 = u_2 - \gamma_{21} z_1 = u_2 - \frac{\sum_{j=1}^{N} z_{1j} u_{2j}}{\sum_{m=1}^{N} z_{1m}^2} z_1$$

$$z_3 = u_3 - \gamma_{31} z_1 - \gamma_{32} z_2 = u_3 - \underbrace{\frac{\sum_{j=1}^{N} z_{1j} u_{3j}}{\sum_{m=1}^{N} z_{1m}^2}}_{\gamma_{31}} z_1 - \underbrace{\frac{\sum_{j=1}^{N} z_{2j} u_{3j}}{\sum_{m=1}^{N} z_{2m}^2}}_{\gamma_{32}} z_2$$

Now, let us check to see if z_3 and z_1 are uncorrelated just as we did before with x and e.

$$\sum_{i=1}^{N} z_{3i} z_{1i} = \sum_{i=1}^{N} u_3 z_{1i} - \sum_{i=1}^{N} z_{1i}^2 \frac{\sum_{j=1}^{N} z_{1j} u_{3j}}{\sum_{m=1}^{N} z_{1m}^2} - \underbrace{\sum_{i=1}^{N} z_{1i} z_{2i} \gamma_{32}}_{=0}$$

The third term above is equal to 0 because we already proved that z_1 and z_2 are uncorrelated. Similar to the previous problem, the summations cancel to give:

$$\sum_{i=1}^{N} z_{3i} z_{1i} = 0 \rightarrow z_3 and\ z_1 are\ uncorrelated$$

Using the same method, we can test to see if z_3 and z_2 are uncorrelated:

$$\sum_{i=1}^{N} z_{3i} z_{2i} = \sum_{i=1}^{N} u_3 z_{2i} - \underbrace{\sum_{i=1}^{N} z_{1i} z_{2i} \gamma_{31}}_{=0} - \sum_{i=1}^{N} z_{2i}^2 \frac{\sum_{j=1}^{N} z_{2j} u_{3j}}{\sum_{m=1}^{N} z_{2m}^2} = 0$$

The $z_1 z_2$ term is equal to 0 again because we proved that z_1 and z_2 are uncorrelated. And again all of the summations cancel to prove that z_3 and z_2 are also uncorrelated. Likewise, any z_d is uncorrelated with any other $z_{d'}$, $\forall d \neq d'$.

If we transform all of the x values into z values using this method, we will transform the feature matrix X into a matrix $Z \in \mathbb{R}^{N \times D}$ such that each column in Z is uncorrelated with all other columns in Z. As long as $D > 1$, this meets the criteria we stated for part 2 of this problem and we can apply the simplification we derived as follows:

$$H_{kk} = \sum_{d=1}^{D} \frac{z_{kd}^2}{\sum_{i=1}^{N} z_{id}^2} \quad \blacksquare$$

4. Discriminative vs Generative Classifiers: Implementing Naïve Bayes and Logistic Regression

1. Implementation

(a) Naïve Bayes

I began by implementing a Naïve Bayes classifier following the pseudocode below.

```
%create probability tables
for vote=1:num_votes
      for senator=1:num_train_senators
            %fill the table for each vote and smooth
            table(vote) = [(count(X=-1, Y=0)+1)/(count(Y=0)+1),
                           (count(X=-1, Y=1)+1)/(count(Y=1)+1;
                           (count(X=1, Y=0)+1)/(count(Y=0)+1),
                           (count(X=1, Y=1)+1)/(count(Y=1)+1)]
      end
end

%make predictions on test data
for senator=1:num_test_senators
      prior = [p(Y=0), p(Y=1)]
      for vote=1:num_votes
            prior = prior * table(vote, senator(vote))
      end
      if prior(1) > prior(2)
            prediction(senator) = 0
      else
            prediction(senator) = 1
      end
end
```

I trained the classifier by constructing probability tables like we did in class. I smoothed my data by adding 1 to each bin of the probability tables, as shown in the pseudocode.

Then I generated predictions for test data using the following equation to maximize the likelihood assuming conditional independence as follows:

$$y^* = \max_y P(Y = y|X = x) = \max_y P(X = x|Y = y)P(Y) = \max_y P(Y = y)\Pi_i P(X_i = x_i|Y = y)$$

Where $P(Y = y)$ is found simply by finding the probability of occurrence for each category in the training data. $P(X_i = x|Y = y)$ is found from the generated probability tables.

I evaluated my NB classifier by training with $2/3$ of the given training data and testing the remaining $1/3$. I found the average classification error over 5 such random splits, but the variance was very large. Instead I averaged the classification error over 1000 runs and found that on average 1.7080 errors were made out of 17 test points. This is an error rate of 10.05%.

(b) Logistic Regression

Next, I implemented a logistic regression classifier based on the pseudocode below:

```
%add bias to the training data
X = [bias TrainData]
w = 0

%run gradient ascent to find optimal w
while change < ϵ
        change = (learning_constant)*(gradient) – (regularization_constant)*w
        w = w + change
end

%predict labels using optimal w
predictions = round(sigmoid(TestData * w)) %rounds up to 1 or down to 0 to predict
```

I calculated the gradient using:

$$\frac{\partial LL(\overline{w})}{\partial w} = \Sigma_{i=1}^{N}[y_i - P(y_i = 1|x_i, w)]x_i^T = \Sigma_{i=1}^{N}[y_i - \theta_x]x_i^T$$

where

$$\theta_x = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

I tried several values for all of the available parameters, but I found the following combination to work best:

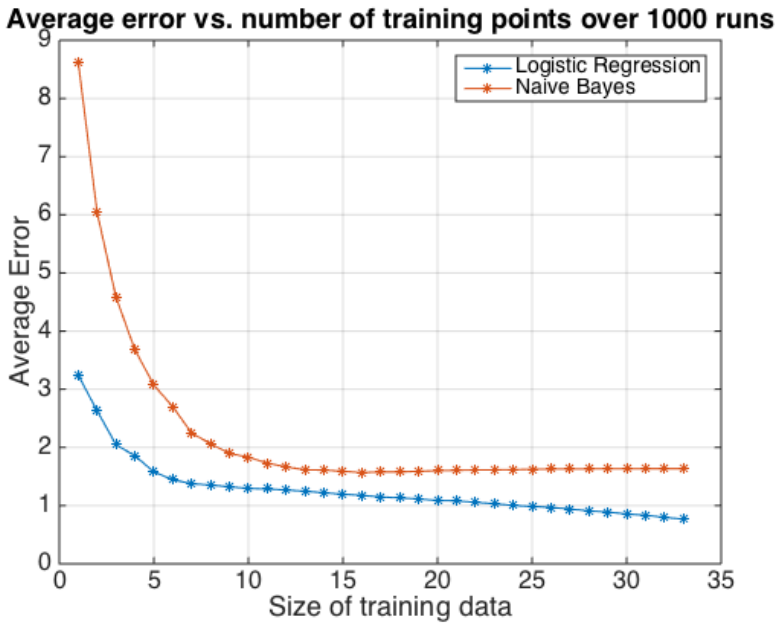
$$Minimum\ Change, \epsilon = 0.0001$$
$$Learning\ Constant, \eta = 0.01$$
$$Regularization\ Constant, \lambda = 0.15$$

I also had the gradient ascent quit if it had not converged in 500 iterations. I evaluated the results by training with 2/3 of the training data and testing with the remaining 1/3. Just as with the NB classifier, I found that averaging 5 such runs had a very high variance. The average over 1000 runs was 0.7970 errors, or about 4.69%. This error was much lower than the error for the NB classifier.

2. Learning Curve

I randomly divided the training data into 2/3 for training (33 points) and 1/3 for testing (17 points) I trained both a NB and a LR classifier using the first 2, then 4, then 8, then 16, then 33 (all) of these randomly selected training points. Then for each of the training sets I averaged the results. These are shown plotted in the graph below.

Again, I found a lot of variance in generating this graph, and I wanted to be more confident about the trends I was seeing. Therefore, I expanded the test so that I trained using each possible number of training points from 1-33 and I averaged the results over 1000 runs. This plot is shown below.



These plots are consistent with the results I saw in the previous problem: the LR classifier performed better than the NB for all sizes of training data. For the NB curve in these plots, I calculated the prior by finding the proportions of Republicans and Democrats in the given training data. I smoothed the data by adding 1 to each bin of the probability tables. For the LR curves, I used the constants mentioned in the previous problem.

3. Conclusions

From the plots above, it can be seen that the NB classifier seems to converge more rapidly compared the LR curve. The NB curve has a consistent average error for any training set larger than about 15 samples, whereas the LR curve does not appear to have converged even with the full 33 training points. And, using the full training set, the average error for NB is higher than the average error for LR. These results are consistent with the results of the paper written by Ng and Jordan. However, LR performs better than NB for every size of training data, which is inconsistent with the Ng/Jordan paper.

Based on these experiments, I conclude that both types of classifiers work for this problem but the LR outperforms the NB. I think this is because the data is so clearly split – some of the bill votes had 100% of one party vote the same way. This type of data makes it easy to train a LR classifier to perform well. However, this type of data requires smoothing in order for the NB to be successful. Smoothing the data alters the original information by adding pseudo votes, and this is likely the reason why the NB does not perform as well as the LR classifier.