Classification: Naive Bayes vs Logistic Regression

John Halloran
University of Hawaii at Manoa
EE 645
Fall 2009

December 21, 2009

---

1 Introduction

Logistic regression and the naive Bayes classifier are both very popular and widely used classification techniques. They are simple, easy to implement, and provide good performance on a wide variety of problems. However, it was not too long ago that many people preferred logistic regression(a discriminative model) to naive Bayes(a generative model). In fact, many people preferred discriminative to generative models in general, since discriminative models achieve lower asymptotic errors. These preferences were tested by Ng and Jordan[1], and it was shown that a decision between which classifier is better is not always a simple answer; it was shown that naive Bayes reaches its asmyptotic error very quickly with regards to the number of training examples. Thus, if training data is scarce, one can expect naive Bayes to outperform logistic regression, but as the number of training examples grows, logistic regression will outperform naive Bayes and achieve a lower asymptotic error rate. In this paper, we will explore both logistic regression and naive Bayes classifiers by giving a background on both, describing how to implement them, and attempting to recreate several of the learning curves found in [1].

2 Notation and Terms

Random variables and vectors are all upper case. We will consider the problem of classification, i.e. when the data label Y takes on a discrete set of values. Further, we restrict our attention to the case when Y is binary valued. In this case, 0 is called the negative class and 1 is called the positive class. We let n denote the number of features and m be the number of samples present in a dataset. The input features are represented as a random vector $X = [X_1\, X_2\, \ldots\, X_n]^T$ and, as consistent with [1], each element of X is a discrete random variable. When discussing a particular element of X, we will address it with its subscript index, and when discussing a particular value that a random variable takes on, say the j^{th} value in its feature space, we denote it as lower case

with two subscripts, the first subscript denoting which random variable in X we are addressing and the second denoting which value it is taking from the feature space. So the i^{th} random variable in X taking on its j^{th} value is $X_i = x_{ij}$. We use superscripts to index training examples, so that X_i^j refers to the value of the random variable X_i in the j^{th} training example. Thus, (Y^j, X^j) corresponds to the j^{th} sample in a dataset. Lastly, $1\{\}$ denotes the indicator function, which is 1 if its argument is true and 0 otherwise.

A generative model is a model wherein we attempt to learn $P(Y|X)$ by first learning $P(X|Y)$ and $P(Y)$. We may then calculate $P(Y|X)$ using Bayes' theorem:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{P(X|Y)P(Y)}{\sum_y P(X|Y=y)} \tag{1}$$

In stark contrast to generative models, a discriminative model attempts to learn $P(Y|X)$ directly.

3 Naive Bayes

The naive Bayes classifier is a generative model wherein we make the oversimplifying assumption that our inputs are independent given the label. Under this assumption, equation (1) becomes

$$P(Y|X) = \frac{P(Y)\prod_{i=1}^{n} P(X_i|Y)}{\sum_y \prod_{i=1}^{n} P(X_i|Y)}$$

Thus, we have

$$P(Y=1|X) = \frac{P(Y=1)\prod_{i=1}^{n} P(X_i|Y=1)}{\sum_y \prod_{i=1}^{n} P(X_i|Y=y)} \tag{2}$$

$$P(Y=0|X) = \frac{P(Y=0)\prod_{i=1}^{n} P(X_i|Y=0)}{\sum_y \prod_{i=1}^{n} P(X_i|Y=y)} \tag{3}$$

Given a set of training data, the maximum likelihood estimates(MLE) for the parameters $P(X_i = x_{ij}|Y = 0), P(X_i = x_{ij}|Y = 1), P(Y = 1)$, and $P(Y = 0)$ are then

$$\begin{aligned}
\hat{P}(X_i = x_{ij}|Y = 0) &= \frac{\sum_{k=1}^{m} 1\{X_i^k = x_{ij} \quad \& \quad Y^k = 0\}}{\sum_{k=1}^{m} 1\{Y^k = 0\}} \\
\hat{P}(X_i = x_{ij}|Y = 1) &= \frac{\sum_{k=1}^{m} 1\{X_i^k = x_{ij} \quad \& \quad Y^k = 1\}}{\sum_{k=1}^{m} 1\{Y^k = 1\}} \\
\hat{P}(Y = 0) &= \frac{\sum_{k=1}^{m} 1\{Y^k = 0\}}{m} \\
\hat{P}(Y = 1) &= \frac{\sum_{k=1}^{m} 1\{Y^k = 1\}}{m}.
\end{aligned}$$

These estimators also make intuitive sense, since they are simply the relative frequency of the occurences of a particular feature given that Y was either 0 or 1.

Thus, after calculating these relative frequencies of the features in the training set, we assign test data to the negative class based on whether the probability of a test sample given $Y = 0$ (given by the MLE) is greater than the probability of the test sample given $Y = 1$. If it is not, we assign it to the positive class. This rule may be expressed using the indicator function:

$$1\{\hat{P}(Y = 1)\prod_{i=1}^{n} \hat{P}(X_i|Y = 1) > \hat{P}(Y = 0)\prod_{i=1}^{n} \hat{P}(X_i|Y = 0)\}$$

However, consider an unlikely scenario wherein $Y^i = 0$ for all training examples, then for any feature we have $\hat{P}(X_i = x_{ij}|Y = 0) = \frac{0}{0}$. We do not know how to assign a probability in this scenario. To guard against such occurences, as well as features which are assigned 0 probability since they were not present in the training data but occur in the test data (0 probability should not be assigned to any event in the feature space), we employ Laplace smoothing, wherein our

maximum likelihood estimates for $P(X_i = x_{ij}|Y = 0)$ and $P(X_i = x_{ij}|Y = 1)$ are then

$$\hat{P}(X_i = x_{ij}|Y = 0) = \frac{\sum_{k=1}^{m} 1\{X_i^k = x_{ij} \quad \& \quad Y^k = 0\} + 1}{\sum_{k=1}^{m} 1\{Y^k = 0\} + J}$$

$$\hat{P}(X_i = x_{ij}|Y = 1) = \frac{\sum_{k=1}^{m} 1\{X_i^k = x_{ij} \quad \& \quad Y^k = 1\} + 1}{\sum_{k=1}^{m} 1\{Y^k = 1\} + J}$$

where we have added one to the numerators and J to the denominators of our estimates (J is the number of discrete values X_i may take).

4 Logistic Regression

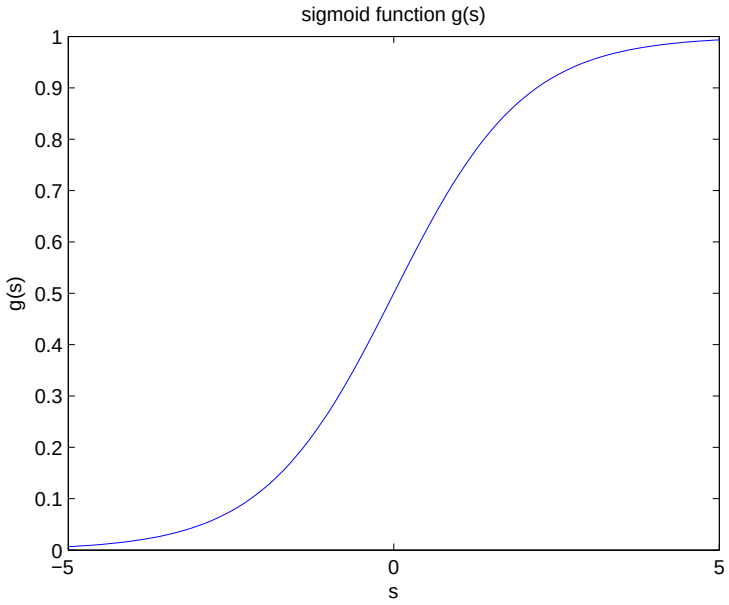

Figure 1: The sigmoid function, $g(s) = \frac{1}{1+e^{-s}}$

Logistic regression is a discriminative model which relies heavily on the logistic function, $g(s) = \frac{1}{1+e^{-s}}$, plotted in figure 1. In logistic regression, we take a linear combination of our input features,

$$s = w_0 + w_1 X_1 + w_2 X_2 + \ldots + w_n X_n = w_0 + \sum_{i=1}^{n} w_i X_i \tag{4}$$

and assign the following probabilities:

$$\begin{aligned} P(Y=1|s) &= \frac{1}{1+e^{-s}} && (5)\\ P(Y=0|s) &= 1-P(Y=1|s) && (6)\\ &= \frac{e^{-s}}{1+e^{-s}}. && (7) \end{aligned}$$

We may write equation (4) more compactly by augmenting our feature vector such that $X_0 = 1, X = [X_0\, X_1 \ldots X_n]^T$, and defining $w = [w_0\, w_1 \ldots w_n]^T$. Then $s = w^T X$, the inner product between our feature X and weight vector w. Assuming our data samples are independent, we obtain the likelihood function, $L(w)$, such that

$$L(w) = P(Y|X,W) = \prod_{i=1}^{m} P(Y_i|X^i, w) = \prod_{i=1}^{m} \frac{(e^{-w^T X^i})^{(1-Y_i)}}{1+e^{(-w^T X^i)}}.$$

The log likelihood function is then

$$\begin{aligned} \ell(w) &= \log L(w) && (8)\\ &= \sum_{i=1}^{m} (1-Y_i) \log(\frac{e^{-w^T X^i}}{1+e^{-w^T X^i}}) && (9)\\ &= \sum_{i=1}^{m} (1-Y^i)(-w^T X^i) - \log(1+e^{-w^T X^i}) && (10) \end{aligned}$$

We proceed by calculating the maximum likelihood estimator $\hat{w}$, such that $\arg\max_{\hat{w}}\{\ell(w)\}$.

Unfortunately, no closed form solution exists to calculate $\hat{w}$. However, $\ell(w)$ is convex, so a global maximum exists and we may find it using gradient ascent. In doing so, we must move in the direction of the gradient of the function. Thus, we take the partial derivatives of $\ell(w)$ with respect to the components of w (note that $\frac{\partial(-w^T X^i)}{\partial w_j} = -X_j^i$)

$$\begin{aligned} \frac{\partial \ell(w)}{\partial w_j} &= \sum_{i=1}^{m} -X_j^i(1-Y^i) - X_j^i \frac{e^{-w^T X^i}}{1+e^{-w^T X^i}}\\ &= \sum_{i=1}^{m} -X_j^i(1-Y^i) + X_j^i - X_j^i \frac{1}{1+e^{-w^T X^i}}\\ &= \sum_{i=1}^{m} X_j^i(Y^i - \frac{1}{1+e^{-w^T X^i}})\\ &= \sum_{i=1}^{m} X_j^i(Y^i - g(w^T X^i)) \end{aligned}$$

which gives us the following gradient ascent update rule,

$$w_{new} = w_{previous} + \alpha \sum_{i=1}^{m} X^i(Y^i - g(w^T X^i)) \quad (11)$$

where α is the stepsize, or learning rate, with which we approach the maximum. Notice that our update simply implies a correction term which is proportional to the error in our prediction on the class labels based on the current weights and training examples. Interestingly, this is the same rule as the least mean squares(LMS) update rule, save for the fact that the function $g(s)$ is different (in the case of LMS, the function pertaining to the weights and features is simply a linear combination of the features, whereas the sigmoid function is nonlinear). This is due to the fact that logistic regression is a member of a broad class of models called Generalized Linear Models [2]. Also of note are the two different implementations of gradient ascent: batch gradient ascent wherein we update based on the total error between our labels and hypotheses, and stochastic gradient wherein we update our weights based on the error between each hypothesis and its label.

Proceeding, we use gradient ascent to calculate our weights, and we encounter some new test data, how do we determine whether we label the samples in this data belong to the negative or postive class? To answer this, we look at likelihood ratio of the probabilites in equations (5), (6), $\frac{P(Y=0|s)}{P(Y=1|s)}$. If the likelihood ratio is greater than unity, then we assign a sample to the negative class. Otherwise, we assign the sample to the positive class. This decision simplifies:

$$\begin{aligned} \frac{P(Y=0|s)}{P(Y=1|s)} &> 1 \\ e^{-s} &> 1 \\ s &< 0. \end{aligned}$$

Thus, if the inner product of our weight vector and a test sample is less than 0, we assign it to the negative class, otherwise we assign it to the possitive class.

5 Testing

Both naive Bayes and logistic regression were implemented in Matlab. The datasets used in testing were, as far as could be deciphered, those used in [1], borrowed from the UCI machine learning database. These datasets are: breast-cancer-wisconsin.data, house-votes-84.data, (sick.data, sick.test), and promoters.data. Only discrete attributes were considered. The method of assigning a numerical label to the attributes was to take the convention of assigning an attribute an integer value based on the appearance of that value in the accompanying .name file with the dataset. The parsers for each dataset are also available in the appendices.

The generalization error for each dataset was calculated by using n-fold cross-validation in the following manner:

1. If test data is not present with dataset, randomly split dataset into $\frac{2}{3}$'s training data, $\frac{1}{3}$ test data.
2. Choose a random subset of the training data to train, with training sizes from 2 to desired maximum training size(for all results, an increment of 1 was used to iterate through test sizes; this is not always necessary, but the closer to one the increment the smoother the learning curve will be).

3. After training each subset, test against the held-out $\frac{1}{3}$ test set. Calculate ratio of incorrectly classified to total testing set size.

4. Repeat m times, averaging classification error over m runs.

The great length of time necessary for batch logistic regression to converge lead to inquiries into both stochasic logistic regression (also called on-line logistic regression) and locally weigted logistic regression(lwlr). Further, a cooling schedule was necessary for stochastic logistic regression to converge on many data sets within a reasonable amount of time. Prior to a cooling schedule, stochastic logistic regression's run time was worse than batch logistic regression. However, this is due to the nature of stochastic logistic regression and the fact that it "bounces" around the maximum, never converging. However, by decreasing the stepsize, we allow stochastic gradient ascent to further close the gap to the global maxium. Locally weighted logistic regression's run-time was significantly shorter than the other two implementations of logistic regression, however its performance was often poorer than batch logistic regression.

5.1 Voting Records

The voting records dataset consisted of 435 votes for each of the U.S. House of Representatives Congressman on the 16 key votes identified by the Congressional Quarterly Almanac in 1984. The data set labels denote whether a Congressman was republican(0) or democrat(1), and the 16 attributes were the key votes and how each congressman voted on the issue. The feature space for each attribute consisted of 3 elements; yea, nay, ?, where ? question mark denoted a value that was simply not yea or nay.

We see that the plot of logistic regression(any of the 3) vs naive Bayes for the voting records data set found in figure 2 matches closely to the plot in [1]. For a small amount of training examples, naive Bayes outperforms logistic regression. Further, it approaches its asymptotic error rate very quickly. We see that as the number of training examples increases, the performance of logistic regression surpasses that of naive Bayes. Also worth noting is the long run-time of stochastic logistic regression displayed in figure 3. This is due to the nature of stochastic logistic regression discussed above; that logistic regression "bounces" around the global maximum. However, by decreasing the stepsize of stochastic logistic regression as it runs, we can achieve much faster run-times without a huge sacrifice in performance.

In figure 4, we see that stochastic logistic regression performs similarly over the data set with a simple cooling schedule. The cooling schedule implemented was linear such that stochastic logistic regression begins with a large step size and a fairly large error(around $1e-2$), and upon achieving this error, the stepsize and error are decreased by a factor of 10. For the results obtained in figure 4, the rate(number of times the stepsize and error are to be adjusted upon a convergence) was 3. Further, figure 5 displays the strength of the cooling schedule. The decrease in run-time for stochastic logistic regression is dramatic when compared to its run-time in figure 3. Such a drastic decrease in run-time warrants some investigation for batch logistic regression. From here on, we will implement stochastic logistic regression with this same cooling schedule. Thus, for the rest of

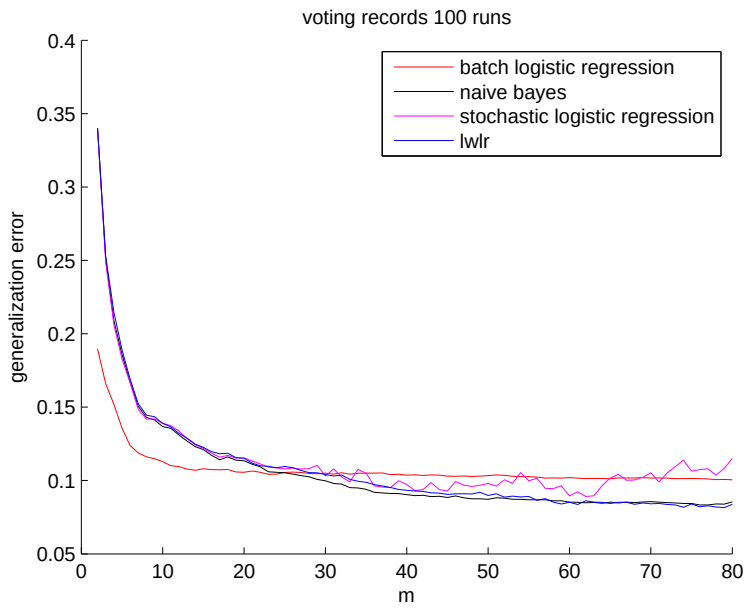

Figure 2: Batch, stochastic, and locally weighted logistic regression and naive Bayes for the voting results data set. Generalization error averaged over 100 runs with a maximum of 80 training examples.

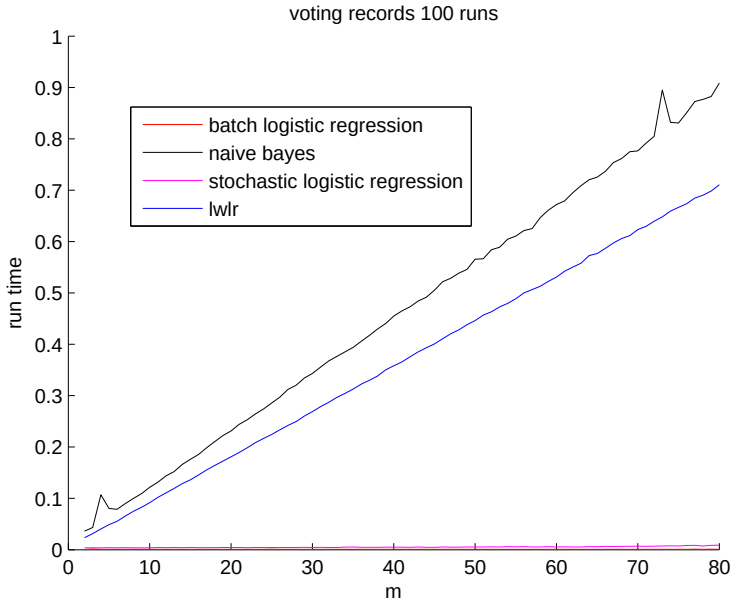

Figure 3: Average run times for the voting records data set over 100 runs with a maximum of 80 training examples.

the paper, when referring to stochastic logistic regression, we mean with cooling.

In figure 6 we have the generalization error associated with batch logistic regression, naive Bayes,

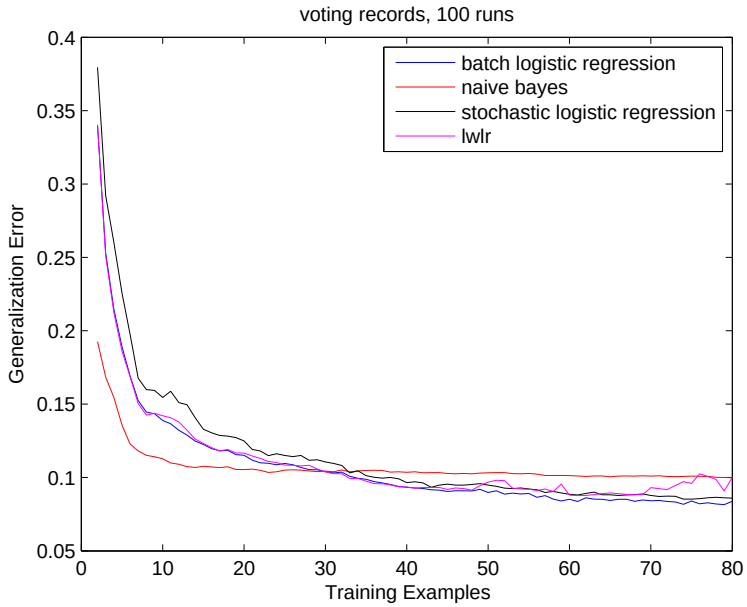

Figure 4: Generalization error for the voting records data set, averaged over 100 runs with a maximum of 80 training examples.

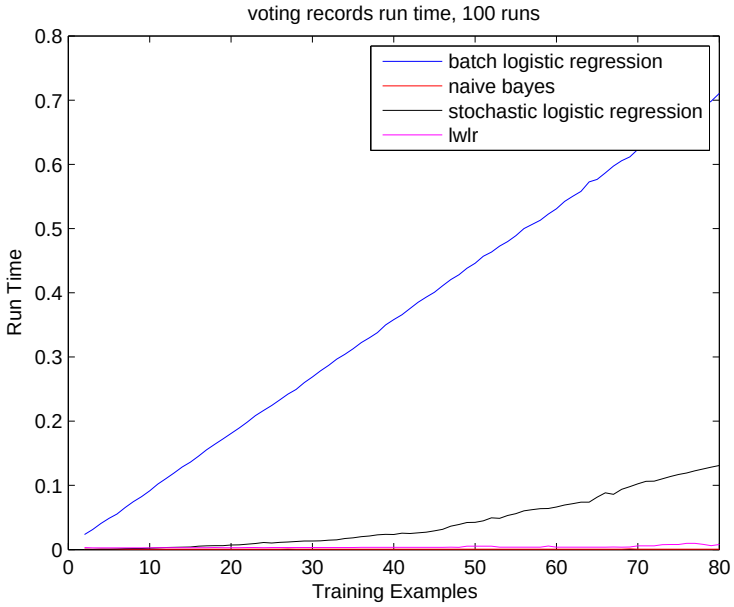

Figure 5: Average run times for the voting records data set over 100 runs with a maximum of 80 training examples.

and batch logistic regression implemented with a cooling schedule(called annealed batch logistic regression due to the similar idea of a cooling schedule used in simulated annealing, although the end goal is different since simulated annealing guards against converging to a local maxima/minima by using a cooling schedule, which is a problem we do not have in logistic regression since there is only a global maximum). Notice that the difference in performance for "normal" batch versus annealed

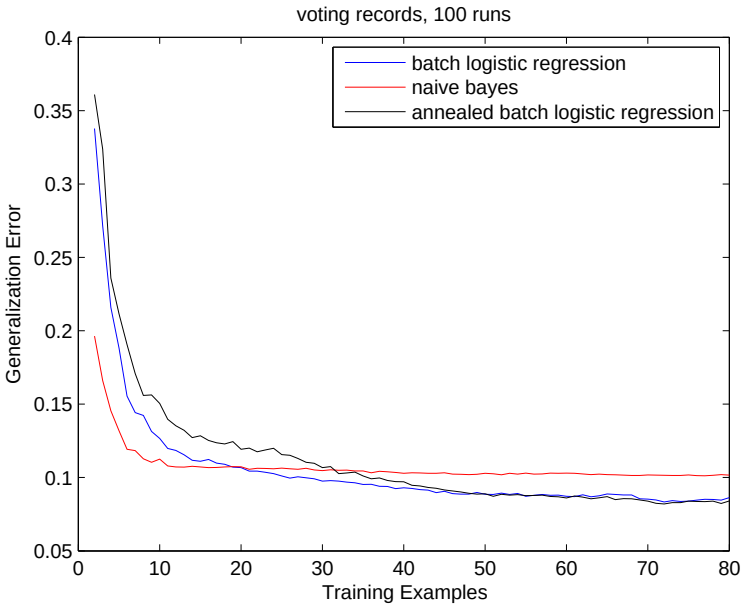

Figure 6: Generalization error for the voting records data set over 100 runs with a maximum of 80 training examples.

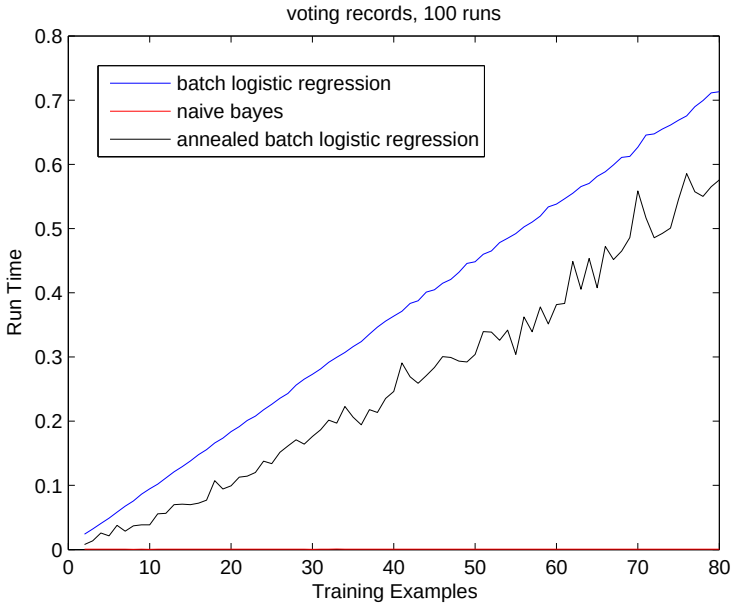

Figure 7: Average run time for the voting records data set over 100 runs with a maximum of 80 training examples.

batch is significant for a smaller number of training examples. However, both perform similarly as the number of training examples increases, reaching the same asymptotic error. Figure 7 shows the difference in run-time between the two different batches. Annealed batch logistic regression has a smaller run-time than that without annealing, although it is not as dramatic as annealing in stochastic logistic regression. Further, annealed batch logistic regression did not perform as well

on the other data sets, and batch without annealing was preferred.

5.2 Sick

The sick dataset pertained to thryoid disease, wherein the class labels were sick(1, patient tested positive for thyroid disease) and negative(0, patient's test came back negative). There were 22 class attributes, 21 of which were medical questions consisting of true or false answers (save for sex, which was male or female), and the 22^{cnd} being the referral source with six possible values. There were 2800 training examples provided, and 972 test examples. Note that in all simulations, the trained data is always tested against the entire 972 test examples.

We initially start out over a small number of training examples to get a feel for the performance as well as the run-time for the different algorithms. Figure 8 shows that of the logistic regression programs and as with the voting results dataset, batch yields the lowest generalization error. Also, the number of training examples is far to small for logistic regression to perform competitively against naive Bayes, as naive Bayes achieves a much lower generalization error. Figure 9 displays the average run times for 100 iterations and 20 training examples. It is clear that while batch logistic regression performs well, its average run-time was already quite large for such a small number of training examples.

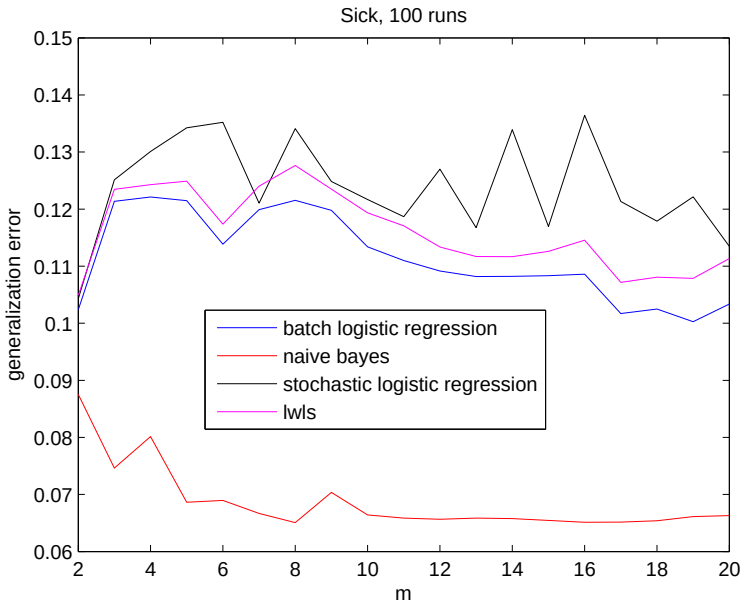

Figure 8: Batch, stochastic, and locally weighted logistic regression and naive Bayes for the sick dataset. Generalization error averaged over 100 runs with a maximum of 20 training examples.

In figure 10, the generalization error is displayed as we increase the maximum number of training examples to 80. Batch logistic regression became infeasible to run with this number of training examples, and in order to run stochastic logistic regression within several hours, the number of iterations had to be slimmed to 50 and the maximal number of iterations allowed for the program

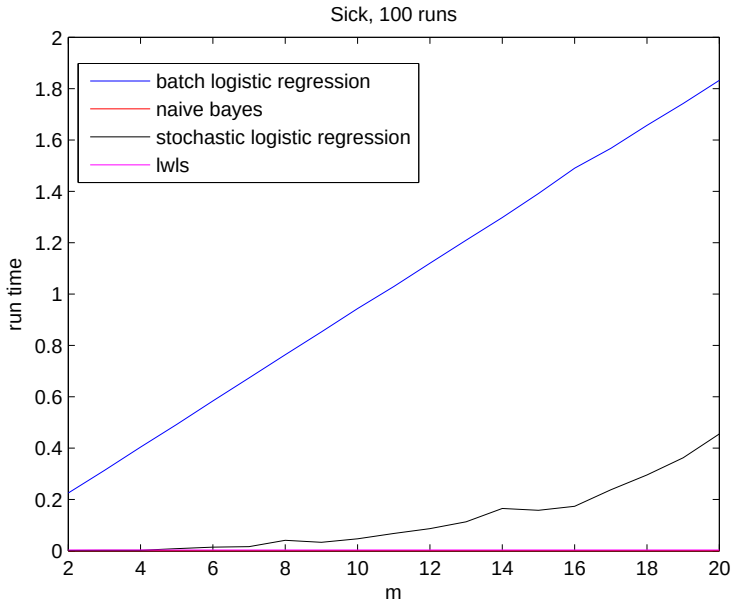

Figure 9: Average run times for the sick data set over 100 runs with a maximum of 20 training examples.

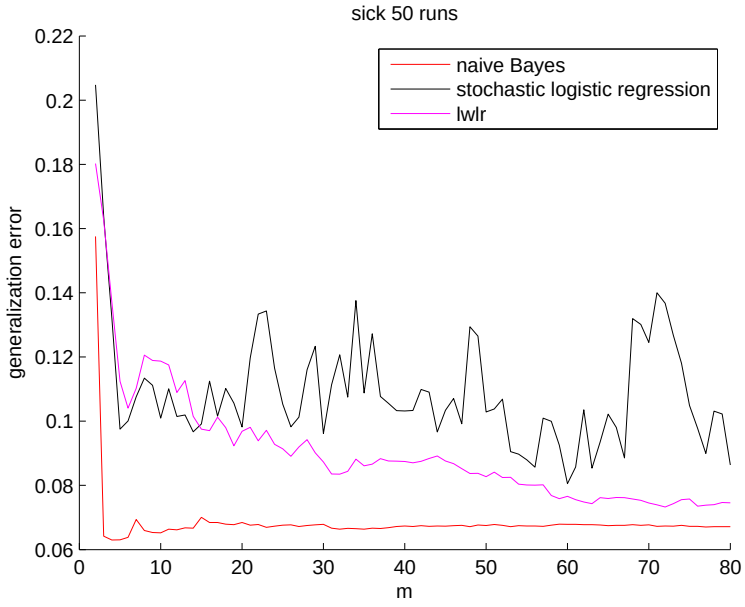

Figure 10: Generalization error for the sick data set averaged over 50 runs with a maximum of 80 training examples.

to run was also slimmed. We see that naive Bayes has reached its asymptotic error rate (in fact long before 80 training examples were considered), and that logistic regression(lwlr) is converging towards its asympotic error rate. Based on the behavior of batch logistic regression in figure 10, we could expect that it would have achieved the lowest generalization error of the logistic regressions if not for its long run-time. Figure 11 displays the average run times, showing that batch logistic

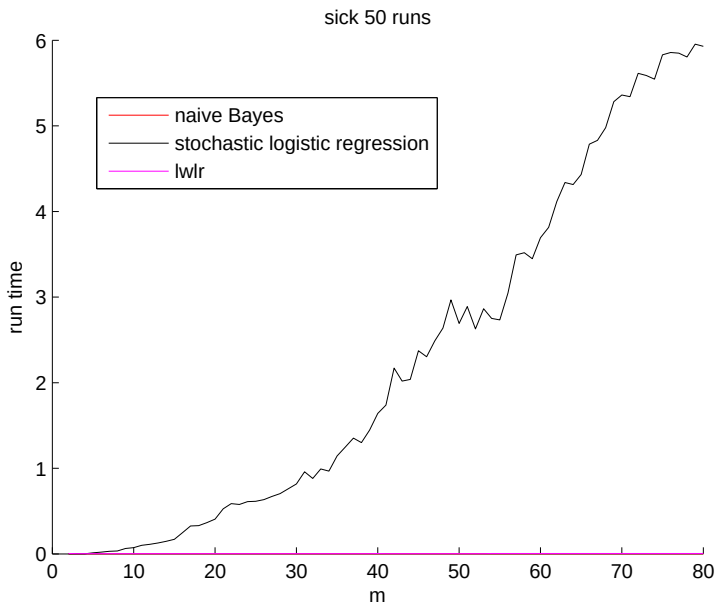

Figure 11: Average run times for the sick data set over 50 runs with a maximum of 80 training examples

regression's run-time is growing exponentially in the number of training examples. With its run-time growing immensely and lack of performance, stochastic logistic regression also became infeasible to run as we increased the number of training examples several more.

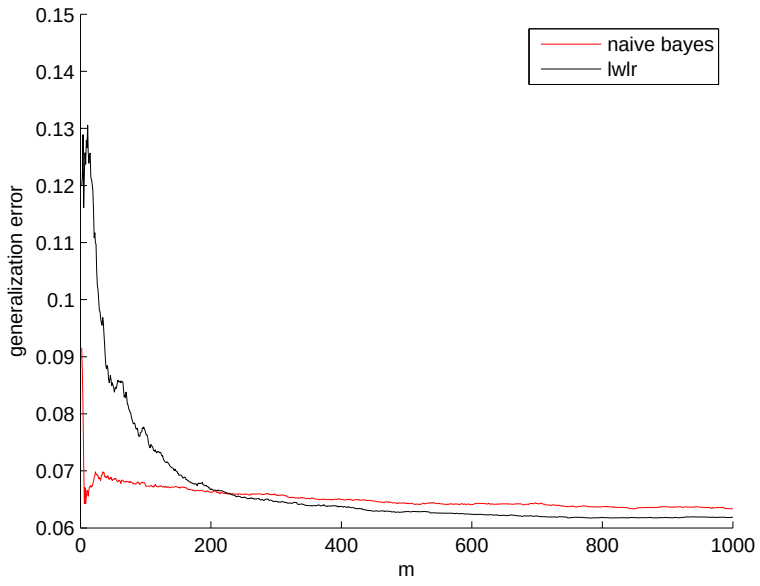

Figure 12: Generalization error for the sick data set averaged over 100 runs with a maximum of 1000 training examples

In figure 12 we have the generalization error after we increasing the maximum number of training

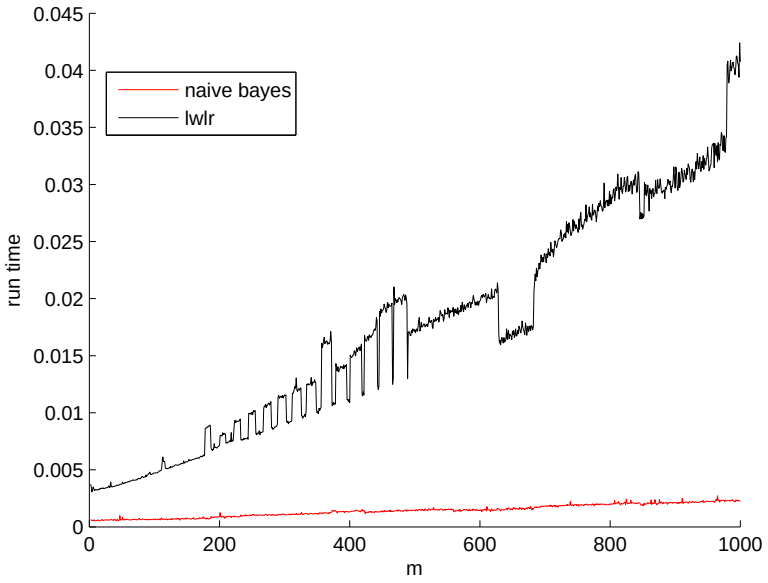

Figure 13: Average run times for the sick data over 100 runs with a maximum of 1000 training examples

examples to 1000. While the run-times of both batch and stochastic logistic regression made them infeasible to run, lwlr looks to have reached the asymptotic error rate for logistic regression. As in the voting data set, we see that logistic regression does indeed reach a lower asymptotic error rate than naive Bayes as the number of training examples grows. However, the number of training examples necessary to achieve this low error is much higher than naive Bayes. Naive Bayes achieved its asymptotic error rate requiring less than 40 training examples, whereas logistic regression required around 250 training examples to outperform naive Bayes and around 800 training examples to reach its asymptotic error rate. Also of note is the difference in run-time between naive Bayes and lwlr in figure 13; the run-time of naive Bayes looks to grow linearly with a very small slope as the number of training examples increases, whereas lwlr's run-time increases dramatically in contrast, which looks to grow quadratically as the number of examples incrases. Note that figure 12 also coincides with the plot found in [1] pertaining to the sick data set, although we have increased the number of training examples enough for logistic regression to its asymptotic error rate.

5.3 Promoters

This data set contains 106 samples pertaining to detecting a promoter based on interactions with proteins and DNA. The class labels consist of +(1) denoting a promoter and -(0) denoting not a promoter. The data consisted of 57 attributes characterizing the sequence, with each attribute taking 1 of 4 possible values.

The run-time of batch gradient descent was very large, even for small training examples, and was thus excluded from testing on this data set. The plots in figures 14 and 16 almost exactly match

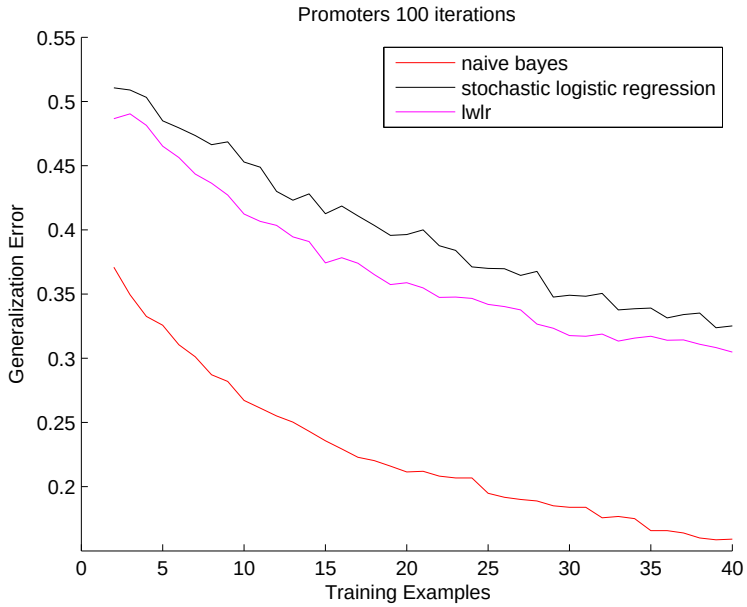

Figure 14: Stochastic and locally weighted logistic regression and naive Bayes generalization error for the promoters data set over 100 runs with a maximum of 40 training examples.

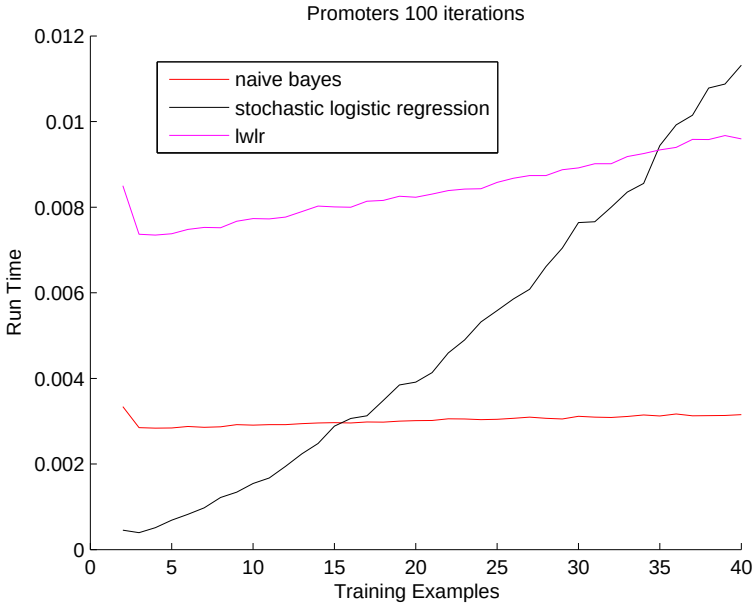

Figure 15: Average run time for the promoters data over 100 runs and a maximum of 40 training examples.

the plot pertaining to the promoters data set found in [1], except that our plots look a bit noisier(to be expected, considering that the plots found in [1] were averaged over 1000 runs, and we are only working with 100) and the maximum number of training examples we consider is 70. Niave Bayes far outperforms logistic regression on this particular data set. Thus, one large drawback of logistic regression is that if the number of training examples is not sufficiently large, logistic regression

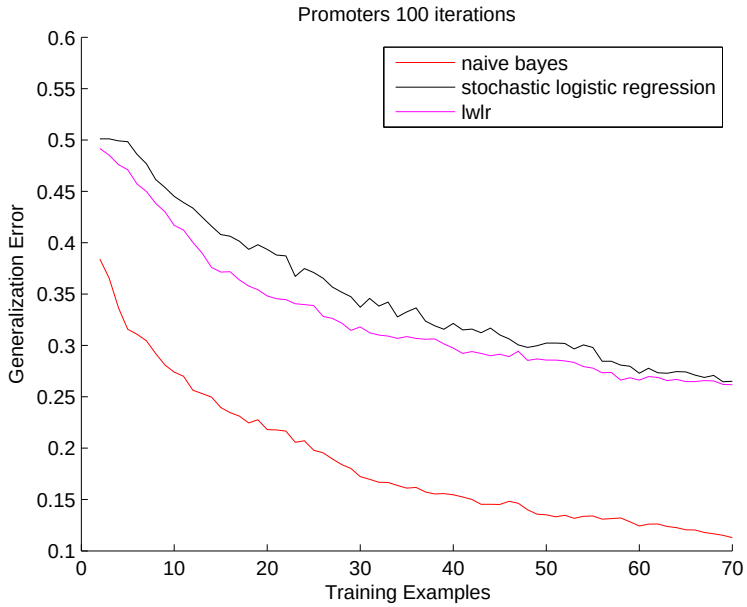

Figure 16: Generalization error for the promoters data set over 100 runs with a maximum of 70 training examples.

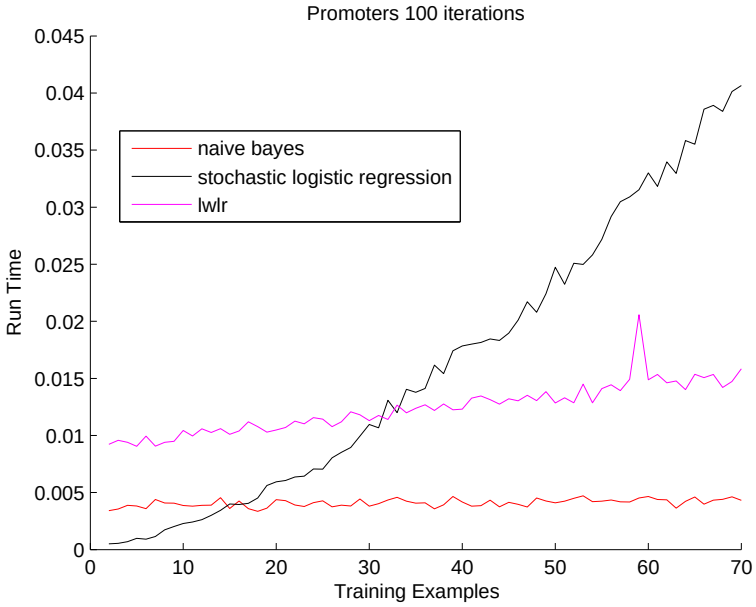

Figure 17: Average run time for the promoters data set over 100 runs with a maximum of 70 training examples.

cannot reach its asymptotic error rate. Looking at figure 15 and 17, it is interesting to see that annealed stochastic gradient ascent's run-time was in the same regime as that of naive Bayes and lwlr. This would imply that the cooling schedule used(the same cooling schedule used and described on the voting data set) worked very well for this particular data set.

5.4 Breast Cancer

This data set contains 699 samples consisting of 10 attributes pertaining to various calculations made on patient breast tissue, each taking on an integer value between 1 and 10. The class labels denote whether a patient's breast tissue was benign(0) or malignant(1).

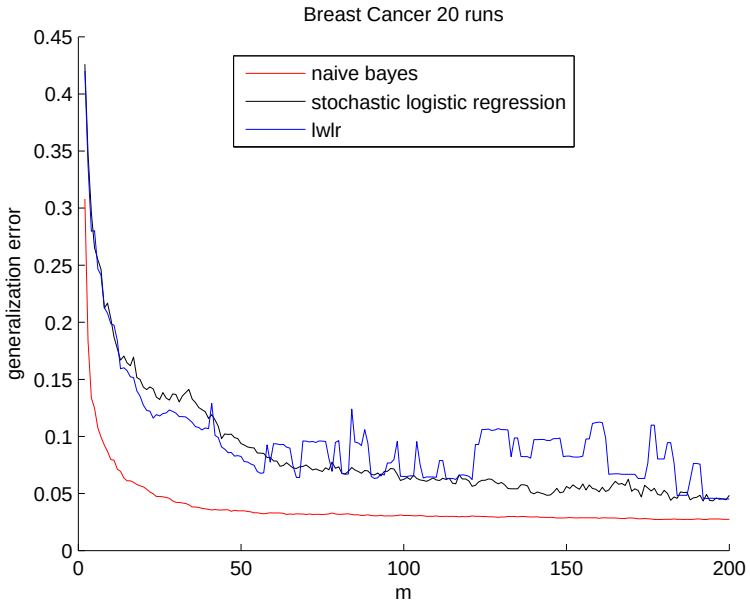

Figure 18: Stochastic and locally weighted logistic regression and naive Bayes generalization error for the breast cancer data set over 20 runs with a maximum of 200 training examples.

For this particular data set, stochastic logistic regression outperforms lwlr and appears to be approaching naive Bayes(fig 18). However, in figure 19 we see the price for this performance; even with a cooling schedule, the run-time of stochastic logistic regression grows exponentially as the number of training examples, and the run-times of naive Bayes and lwlr are essentially 0 in comparison. Due to this extreme overhead, we further increase the number of training examples but only consider the performance of lwlr and naive Bayes.

We see that lwlr is unable to reach its asymptotic error rate within the training example sizes(fig 20). Also, note that the behavior of the curves is similar to those pertaining to the breast cancer data set in [1], but the numerical values differ.

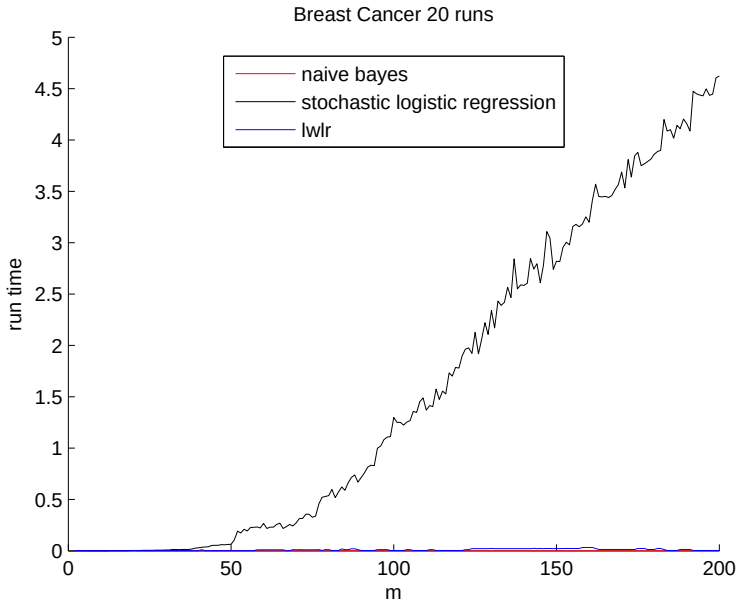

Figure 19: Average run times for the breast cancer data set over 20 runs with a maximum of 200 training examples.

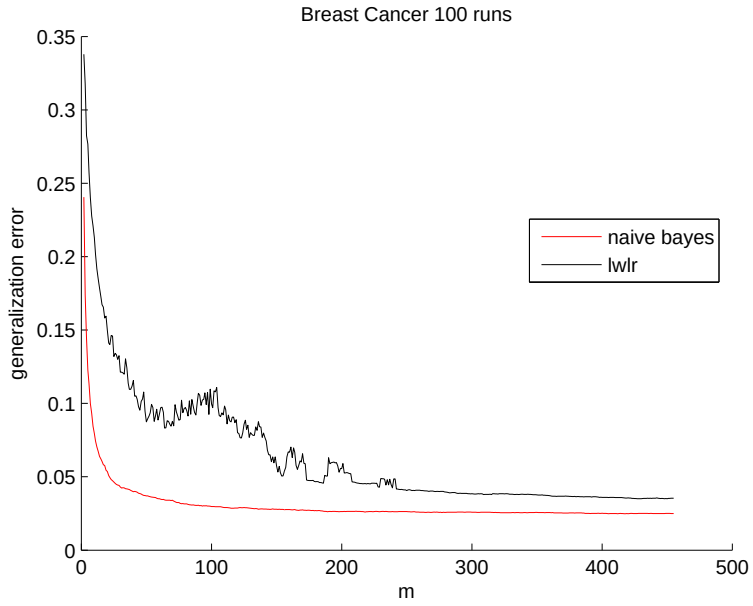

Figure 20: Generalization error for the breast cancer data set over 20 runs with a maximum of 200 training examples.

6 Conclusions

We were able to successfully implement both logistic regression and naive Bayes, as well as recreate and observe some of the results in [1]. Naive Bayes performed very well on the data sets considered.

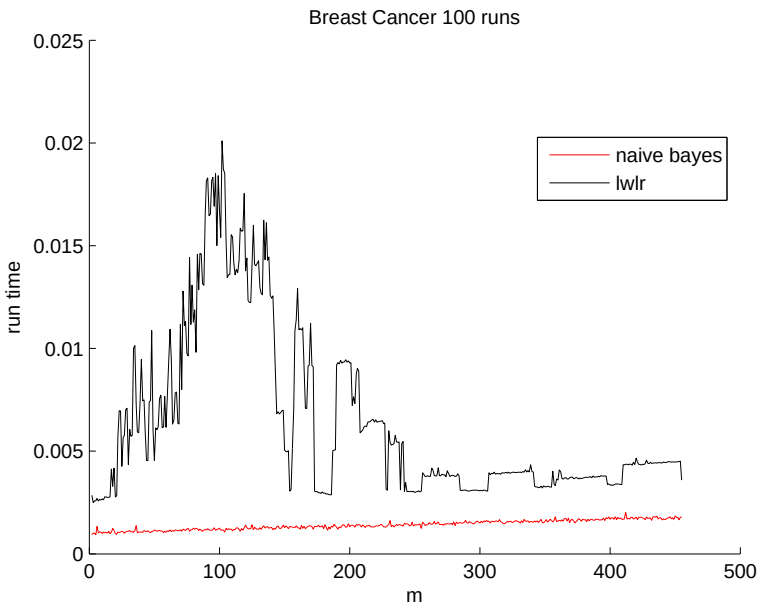

Figure 21: Average run times for the breast cancer over 20 runs with a maximum of 200 training examples.

It approaches its asymptotic error without the need for a large number of training examples, and it does so very quickly. Logistic regression, on the other hand, is capable of outperforming naive Bayes, given the number of training examples is large enough. Also, locally weighted logistic regression is much faster than both batch and stochastic gradient ascent, although, to be fair, both programs would run much faster if programmed in C or C++. In that regard, it is difficult to truly compare run-times between the three programs, although one can expect lwlr to converge much faster than batch and stochastic logistic regression(it is an implementation of the Newton-Raphson method geared towards logistic regression). However, if the number of training examples is very large, then using lwlr is very costly, since it involves the inverse of a Hessian matrix. In such settings, stochastic gradient ascent (with a cooling schedule which works well on the data) would be preferable, due to the simple and uncostly operations involved in it.

As a rule of thumb, naive Bayes will almost surely outperform logistic regression if the number of training examples is small. Further, it will reach its asymptotic error very quickly, making it much more desirable in such a scenario than logistic regression. Thus, if we run both algorithms on a training set and notice that naive Bayes is far outperforming logistic regression, than it is a safe bet to stick with naive Bayes until a much larger amount of training examples are available. However, if we run both programs on a training set and see that naive Bayes is not outperforming logistic regression by much, then we can assume that logistic regression will outperform naive Bayes within a relatively small amount of new training examples.

7 Appendix A. Matlab Programs

7.1 Dataset Parsers

7.1.1 Voting Records

First read text into cell array using Matlab's import wizard, then run:

```
%republican = 0, democrat = 1

n1 = 435;
n2 = 17;
voting_records = zeros(n1, n2);
for i = 1:n1
    if(voting{i}(1)=='r')
        voting_records(i,1) = 0;
    else
        voting_records(i,1) = 1;
    end
    run = 2;
    while(voting{i}(run)~=',') run = run+1; end
    run = run+1;
    for j = 2:(n2-1)
        if(voting{i}(run)=='n')
            voting_records(i,j) = 1;
        elseif(voting{i}(run)=='y')
            voting_records(i,j) = 2;
        else
            voting_records(i,j) = 3;
        end
        run = run+2;
    end
    if(voting{i}(run)=='n')
        voting_records(i,n2) = 1;
    elseif(voting{i}(run)=='y')
        voting_records(i,n2) = 2;
    else
        voting_records(i,n2) = 3;
    end
end

save voting voting_records
```

7.1.2 Sick

```
clear all; clc; close all
%%
%attributes:
%sex = 1 (M), 2 (F)
%on thyroxin = 1 (f), 2 (t), 3(?)
%all other cases where the attribute set is {t, f} are the same
%referral source = 1(WEST), 2(STMW), 3(SVHC), 4(SVI), 5(SVHD), 6(other),
%7(?)
%label = 1(sick), 0(negative), 3(?)

n1 = size(sick,1);
n2 = 23;
s = zeros(n1, n2);
for i = 1:n1
    run = 1;
    while(sick{i}(run)~=',') run = run+1; end
    run = run+1;
    if(sick{i}(run) == 'M')
        s(i,1) = 1;
    elseif(sick{i}(run) == 'F')
        s(i,1) = 2;
    else
        s(i, 1) = 3;
    end
    run = run+2;
    for j = 2:16
        if(sick{i}(run)=='f')
            s(i,j) = 1;
        elseif(sick{i}(run)=='t')
            s(i,j) = 2;
        else
            s(i,j) = 3;
        end
        run = run+2;
    end
    for j = 17:21
        while(sick{i}(run) ~=',') run = run+1; end
        run = run+1;
        if(sick{i}(run) =='f')
            s(i,j) = 1;
        elseif(sick{i}(run)=='t')
            s(i,j) = 2;
        else
            s(i,j) = 3;
        end
```

```
        run = run+2;
    end
    while(sick{i}(run)~=',') run = run+1; end
    run = run+1;
    ind = 22;
    switch sick{i}(run)
        case 'W'
            s(i, ind) = 1;
        case 'o'
            s(i, ind) = 6;
        case 'S'
            if(sick{i}(run+1)=='T')
                s(i,ind) = 2;
            else
                if(sick{i}(run+2)=='I')
                    s(i, ind) = 4;
                else
                    if(sick{i}(run+3)=='C')
                        s(i, ind) = 3;
                    else
                        s(i, ind) = 5;
                    end
                end
            end
        otherwise
            s(i, ind) = 7;
    end
    while(sick{i}(run)~=',') run = run+1; end
    run = run+1;
    ind = 23;
    if(sick{i}(run)=='s')
        s(i, ind) = 1;
    elseif(sick{i}(run)=='n')
        s(i, ind) = 0;
    else
        s(i, ind) = 3;
    end
end
```

7.1.3 Promoters

First read text into cell array using Matlab's import wizard, then run:

```
%attributes
%label = 1(+), 0(-)
```

```
%2-58 = 1(a), 2(g), 3(t), 4(c)

n1 = size(promoters,1);
n2 = 58;
p = zeros(n1, n2);
for i = 1:n1
    run = 1;
    if(promoters{i}(run)=='+')
        p(i,1) =  1;
    else
        p(i,1) = 0;
    end
    run = run+1;
    l = find(isspace(promoters{i}));
    run = l(end)+1;
    for j = 2:n2
        switch promoters{i}(run)
            case 'a'
                p(i,j) = 1;
            case 'g'
                p(i,j) = 2;
            case 't'
                p(i,j) = 3;
            case 'c'
                p(i,j) = 4;
            otherwise
                p(i,j) = 5;
        end
        run = run+1;
    end
end
```

7.1.4 Breast Cancer

On the very last dataset, I found a very handy tool in Matlab called textscan which would have made life much easier from the start for parsing.

```
fid = fopen('breast_cancer/breast-cancer-wisconsin.data', 'r');
D = textscan(fid, '%d %d %d %d %d %d %d %d %d %d %d', 'delimiter', ',', ...
    'treatAsEmpty', '?', 'EmptyValue', 11);
fclose(fid);

n1 = length(D{1});
n2 = size(D, 2);
b = zeros(n1,n2-1);
```

```
for i = 2:n2
    b(:,i-1) = D{i};
end
%label classes 0 (benign, 2 in data), 1 (malignant, 4 in data)
x = find(b(:,end)==2);
b(x,end) = 0;
x = find(b(:,end)==4);
b(x,end) = 1;

save breast_cancer b
```

References

[1] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Neural Information Processing Systems*, 2002.

[2] Andrew Ng. Cs229 lecture notes part III: Generalized linear models.