Columbia University – Fall 2013

Machine Learning Homework #1

Joe Ellis - jge2105

September 26, 2013

1 Problem 1

In this problem we investigate fitting a polynomial model regression to a dataset consiting of 10 data points. Our d-dimensional polynomail function, f, is described below,

$$f(x;\boldsymbol{\theta}) = \boldsymbol{\theta}_0 + \boldsymbol{\theta}_1 x_1 + \boldsymbol{\theta}_2 x_2 + ... + \boldsymbol{\theta}_d x_d \tag{1}$$

where $\boldsymbol{\theta}$ is the learned parameters of our model, and x is a given data point. We choose to use squared error for our empirical loss function to gauge how well our model fits the data. The equation for empirical loss can be seen below, where x represents the given data points, y represents the output, which our function f is attempting to solve for.

$$R_{emp}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{n}(y_i - f(x_i;\boldsymbol{\theta}))^2 \tag{2}$$

We perform cross-validation to find the adequate dimensionality of an appropriate model. The cross-validation performed is completed 1000 times for each model complexity with train/test split of 9 points for training and 1 point for testing. The results of the cross-validation can be seen in Figure 1, and we can see that a model of 3 or 4 dimesions is suitable for this dataset.
Models with lower than 3-dimensions have a hard time describing the variation in the data, where as models with many more than 3-dimensions tend to overfit. Examples of chosen f of dimension 0,1,3, and 7 can be seen in Figure. 2.

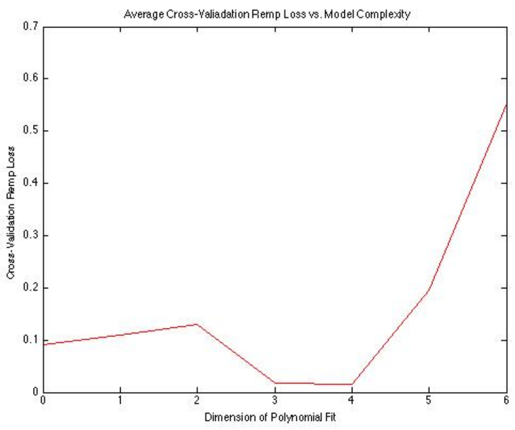

Figure 1: Cross - Validation Loss vs. Model Complextiy

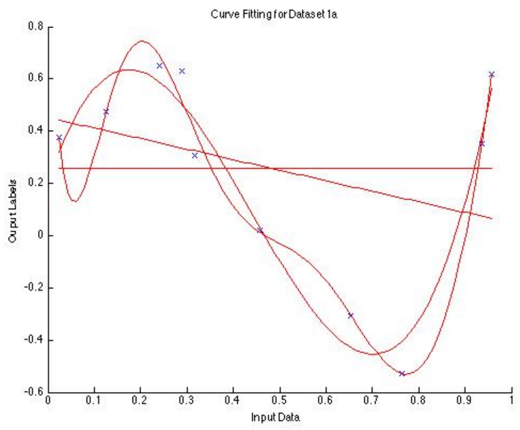

Figure 2: Polynomial Fits of dimensionality, 0, 1, 3, and 7

Table 1: Model Values for Problem 1 Polynomial Regression

d = 0	0.259	0	0	0	0	0	0	0
d = 1	0.451	-0.404	0	0	0	0	0	0
d = 3	0.195	5.448	-19.6041	14.9130	0	0	0	0
d = 7	0.744	-21.80	309.74	-1683.94	4426.87	-6093.68	4220.0	-1159.6

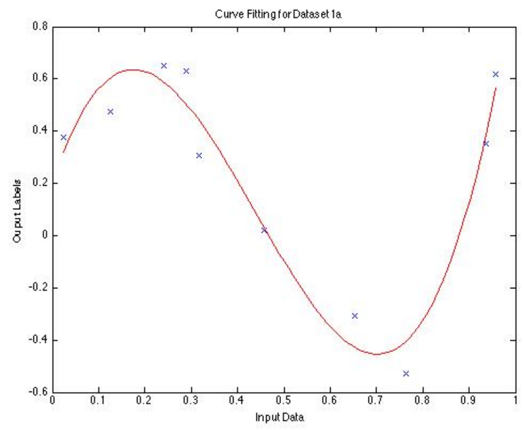

Figure 3: Polynomial Curve Fit for the Dataset1a

.

Using the 3-dimensional model for f we achieve the best polynomial regression function with $R_{emp}(\boldsymbol{\theta}) = 0.0046$, and $\boldsymbol{\theta} = (0.1951, 5.4884, -19.6041, 14.9130)$. The model values for each of these curves can be seen in Table 1. The plot of the polynomial that was found to fit the data set can be seen in Figure 3.

2 Problem 2

In Problem 2 we also address the issue of finding some function to fit to a given set of data points and their real valued labels. However, instead of using a standard polynomial basis function as described in Problem 1 we instead use a set of basis functions that consist of harmonic sets of *cos* and *sin* functions,

$$f(x; \boldsymbol{\theta}) = \theta_0 + \sum_{i=1}^{k} \theta_i sin(i * x) + \sum_{i=1}^{k} \theta_{i+k} cos(i * x), \tag{3}$$

and these are known as sinusoidal basis functions. The calculation of the sinusoidal regression function shown above was completed in the "sinusoidalreg.m" program, which was handed in with this assignment. The $R_{emp}(\boldsymbol{\theta})$ was calculated in the same way as above. For this problem we once again used 1-dimensional input data with real

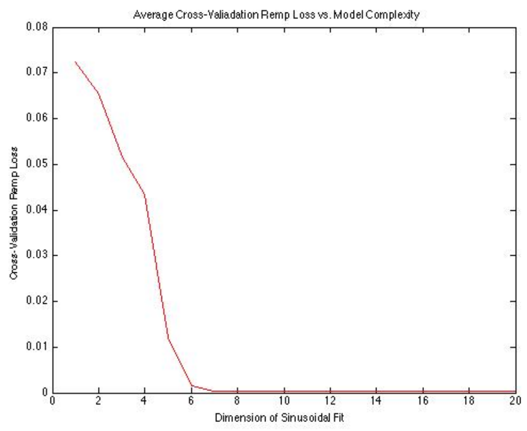

Figure 4: Cross Validation Error for Model Complexity

.

valued output variables from “dataset1b.txt” that was provided to us. For this dataset the first 100 elements from dataset1b were used for training and the second 100 were used for testing. We completed 100-fold cross validation across the training set using a training/test split of 80 points for training and 20 for test to determine what was a good complexity to model these data points. A plot of the cross-validation accuracies can be seen in Figure 4.
We can see that a good dimensionality for k that allows for a simple model but good performance, is $k = 6$. Using $k = 6$ as our model parameter we were able able to achieve a training error of $R_{emptrain} = 0.0012$ and testing error of $R_{emptest} = 0.0042$. The found model has 13 parameters, and these parameters will be outputted to the screen if “runProblem2.m” is executed in the matlab command line environment, they were ommitted here for brevity. Finally, the result of our sinusoidal regression function can be seen in Figure 5, where points marked by an “x” are training points, and those marked by an “o” are test points.

3 Problem 3

In Problem 3 we will use a linear logistic regression function to perform a binary classification task on 2-D input data. In this problem we find a suitable $\boldsymbol{\theta}$ for our logistic classification function by using batch gradient descent. The logistic classification function is defined as,

$$f(x; \boldsymbol{\theta}) = \frac{1}{1 + exp(-\boldsymbol{\theta}^t \boldsymbol{x})}. \tag{4}$$

For this problem instead of using the average squared error for our loss function instead we will use, the logistic loss cost function, which is defined as,

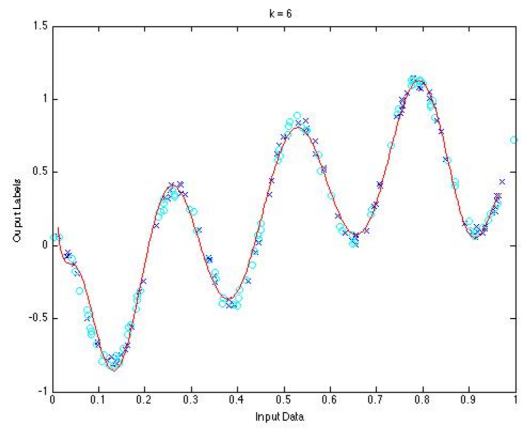

Figure 5: Sinusoidal Regression Function

$$R_{emp}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)log(1 - f(x;\boldsymbol{\theta})) - y_i log(f(x;\boldsymbol{\theta})). \tag{5}$$

To perform gradient descent to find the global minimum of $R_{emp}(\boldsymbol{\theta})$ we need to find the partial derivatives of $R_{emp}(\boldsymbol{\theta})$ with respect to each different parameter within the $\boldsymbol{\theta}$ vector. The partial with respect to each element of $\boldsymbol{\theta}$ is,

$$\frac{\partial R_{emp}}{\partial \theta_j} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x;\boldsymbol{\theta}))x_{ij}. \tag{6}$$

Once we have the partial derivative for each element we can find the $\nabla R_{emp}(\boldsymbol{\theta})$, and then update our theta using this found gradient. We define a stepsize as $\eta = 100$, and then update our $\boldsymbol{\theta}$ after every iteration of gradient descent according to the equation,

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} + \eta * \nabla R_{emp}(\boldsymbol{\theta}) \tag{7}$$

This algorithm completes once $|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t}| < \epsilon$, where we have chosen $\epsilon = 0.1$. Using the gradient descent algorithm we were able to achieve a perfect binary classification error on the given dataset, and our algorithm converges to a solution after approximately 80 iterations. The η chosen allowed for very aggresive steps, and with this our Loss and binary error had some level of oscillation throughout the process. Our final found normalized parameters give $\boldsymbol{\theta} = (-0.319, 0.836, 0.444)$, with a $R_{emp}(\boldsymbol{x};\boldsymbol{\theta}) = 0.0146$ In Figure 6, we can see the final linear logistic classifier imposed on top of our dataset, which shows the classification scheme.

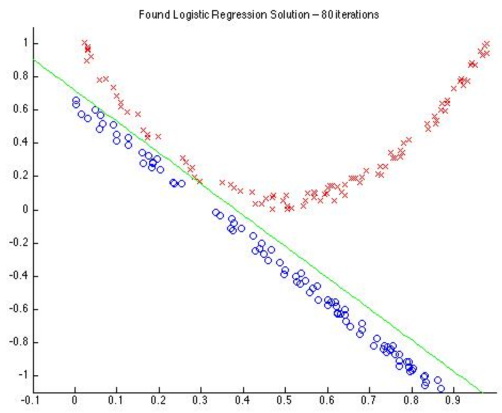

Figure 6: Found Logistic Classifier

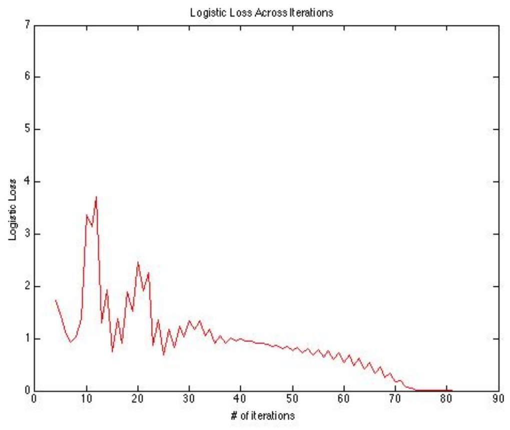

Figure 7: Empirical Loss throughout the Gradient Descent

Figure 7 demonstrates how the $R_{emp}(\boldsymbol{\theta})$ varies over the iteration process, notice the oscillations due to the large step size. Figure 8 shows how the Binary Classification error changes throughout the process. The $|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t}|$ measure can be seen for every step in the iteration in Figure 9.

It should be noted that if a smaller η was chosen the plots 7-9 would have less oscillation within them. We experimented with other values of η and ϵ, and although the other values allowed for smoother descent they also took longer to converge. This trade-off can be changed and altered based on the situation or use of the algorithm. It is also true that we could possibly find a "better" solution if we allowed for smaller ϵ, but it would take longer to converge. A trade-off between speed and finding the best possible solution does exist with gradient descent, and in these examples we have tried to minimize time to convergence, although the graphs could looks lightly different if other elements of the

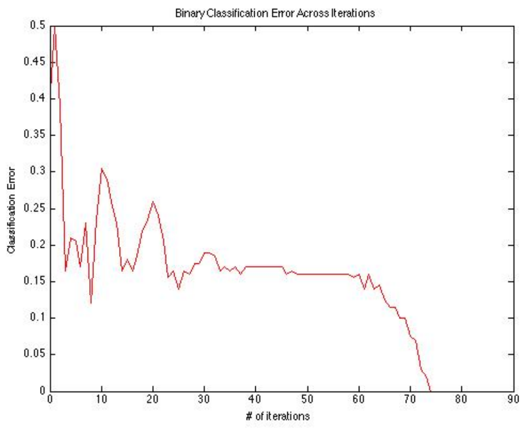

Figure 8: Binary Classification Error throughout the Gradient Descent

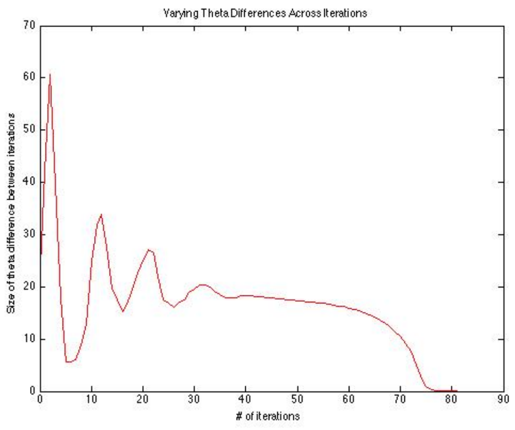

Figure 9: Theta Difference throughout the Gradient Descent

process were prioritized.

4 Problem 4

In Problem 4 we will discuss and prove some properties of the logistic squashing function. This function is denoted as $g(z) = \frac{1}{1+e^{-z}}$. We will show that $g(-z) = 1 - g(z)$. The proof is as follows.

$$\begin{aligned}
1 - g(z) &= 1 - \frac{1}{1+e^{-z}} \\
&= \frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}} \\
&= \frac{e^{-z}}{1+e^{-z}} \\
&= \frac{e^{-z}}{1+e^{-z}}(\frac{e^{z}}{e^{z}}) \\
&= \frac{1}{1+e^{z}} \\
&= g(-z).
\end{aligned}$$

Based on the proof shown above we will show that the inverse of the logistic squashing function is, $g^{-1}(y) = ln(\frac{y}{1-y})$. Let $g(z) = y$.

$$\begin{aligned}
g^{-1}(y) &= g^{-1}(g(z)) \\
&= ln(\frac{g(z)}{1-g(z)}) \\
&= ln(\frac{g(z)}{g(-z)}) \\
&= ln(\frac{\frac{e^{z}}{e^{z}+1}}{\frac{e^{-z}}{e^{-z}+1}}) \\
&= ln(\frac{e^{z}+1}{e^{-z}+1}) \\
&= ln(\frac{e^{z}(e^{-z}+1)}{e^{-z}+1}) \\
&= ln(e^{z}) \\
&= z.
\end{aligned}$$

Therefore, we have shown that $g^{-1}(y) = ln(\frac{y}{1-y})$.