6.867 Problem Set 1

September 29, 2016

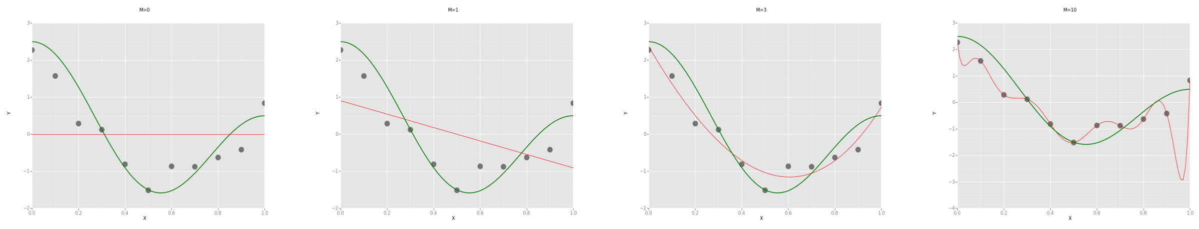
Linear Basis Function

If we assume that errors are normally distributed, $Y \sim N(X\theta, \sigma^2)$, then our maximum-likelihood estimator is equivalent to the OLS estimator. From Bishop, we know that

$$\hat{\theta}_{OLS} = \hat{\theta}_{ML} = (\Phi^T \Phi)^{-1} \Phi^T Y$$

where Φ denotes the polynomial basis expansion of X for order M.

Below, we replicate Bishop's plots for $M = 0$, 1, 3 and 10 using the aforementioned solution for the gradient and python's ggplot library.



$M = 0$ $M = 1$ $M = 3$ $M = 10$

Figure 1: Plots of fitted functions for various orders M in red. The green curve represents the function that actually generated the data.

The sum of squared errors of a linear polynomial basis model is defined by:

$$SSE = (\Phi\theta - Y)^T (\Phi\theta - Y)$$

with gradient

$$\nabla SSE = 2\Phi^T (\Phi\theta - Y)$$

which we verified with the central differences method for all values of M. We then compared the results of using Batch Gradient Descent and Stochastic Gradient Descent to find the optimal weights for different values of M. For BGD, learning rates between .02 and .06 converged for small values of M. For larger values of M, learning rates had to be between .02 and .04 to get convergence.

Since we used the Monro-Robbins learning schedule, where the k parameter impacts the steepness of the learning schedule and the τ impacts the level of the learning rate for SGD, it is difficult to directly compare learning rates between SGD and BGD. In general, SGD had to make many more iterations than BGD since it uses only one row per iteration, compared to BGD which uses the whole dataset. Additionally, SGD needed a higher average learning rate to converge than BGD for all values of M. Reducing SGD's stopping/convergence criteria reduce the probability of stopping at any point, regardless of whether the algorithm is stuck at a local minima or at the target. For this reason, reducing the stopping criterion is dangerous. Luckily, the decreasing learning schedule means that the weight does not fluctuate wildly near convergence, even if it doesn't meet the desired stability.

Table 1: 2.4: Weights learned for Cosine Basic Expansion, M=8

	Learned Weight	True Weight
w_1	0.77	1.0
w_2	1.09	1.5
w_3	0.10	0.0
w_4	0.14	0.0
w_5	-0.05	0.0
w_6	0.36	0.0
w_7	0.01	0.0
w_8	0.02	0.0

To close our beautiful adventure with gradient descent, we tried to learn the optimal θ for the actual data generation process using different values of M. The results were very similar for all values of M., so we display the results for M=8. The learned solution for all values of M has lower MSE than the true weight, so we need either more data or a different loss function (maybe with a regularization parameter) to improve our result.

Ridge Regression

Given the problems we faced for high values of M when learning the cosine basis function, we will now add a regularization term λ to our loss function to push coefficients towards 0 and avoid overfitting. The closed form solution for Ridge from Bishop equation 3.27

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{y} \tag{1}$$

where $\mathbf{X}$ is the feature matrix after the polynomial basis transformation and y are the targets, and $\hat{\theta}$ are the feature coefficients. λ is a regularization parameter – a higher lambda will penalize the model more for having large feature weights, a lower model will penalize the model less. With $\lambda = 0$ the solution is the same as that or Ordinary Least Squares.

For data we were given a training set of 13 observations, a testing set of 10 observations and a validation set of 22 observations. The algorithm for optimizing hyperparameters was to 1) search for the best combination of M and λ by training the model on the training data and testing it on the validation set. 2) given the optimal M and λ (based on validation set performance), use the corresponding $\hat{\theta}$ to attempt to predict values of $\mathbf{y}$ for the test data, and measure the Mean Squared Error of our predictions. We also experimented with training on flipping the test and train dataset to check whether our labeling was robust to the outlier in dataset B. The following table shows the results of that procedure on a search space of λ in $[0, .1, .5, 1, 3, 5, 10, 12, 100, 200, 1000, 10000]$ and M between 1 and 5. For both train/test splits, the optimal M is 2, whereas the optimal λs are in the same range.

		Train MSE	Validation MSE	Test MSE
M	L			
1	0.0	1.44	1.60	1.82
	0.5	1.44	1.52	1.71
	5.0	1.72	1.33	1.36
3	0.0	0.98	1.38	3.76
	0.5	1.00	1.28	3.16
	5.0	1.32	1.80	2.58

As we expect, higher values of M and lower values of L lead to a near perfect in-sample fit to the data. However, the validation set enforces a lower M. Interestingly, however, since dataset A and the validation set are fairly well behaved (strong relationship between X and Y, no outliers), regularization does not help on the validation set. and a model with M=2 and no regularization λ=0 performs well. All the models, including those with higher λ, perform considerably worse on the test set.

When we train on dataset B, which has a significant outlier, however, regularization reduces (i.e. improves) Validation and test MSE. In this procedure, models with $\lambda = 0.5, M = 3$ performed the best on the validation set. As the below table shows, however, more regularization would have helped. The best parameters for the test validation set were $\lambda = 0.5, M = 3$ though $\lambda = 5, M = 1$ would have performed better on the test set. Since regularization prevents over-fitting, it adds more value when there are outliers in the training data. On dataset A, where everything is very clean, there are no bad lessons that the model could learn.