Pset I

*

Problem I

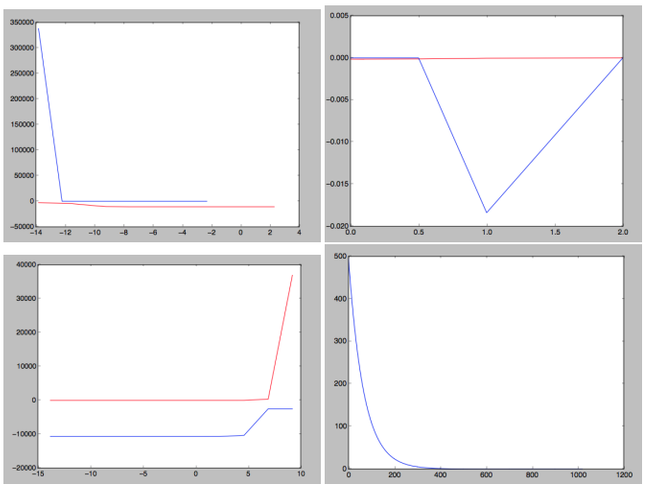

Fig. 1. 1.1: η vs loss function; 1.2: magnitude of guess vector vs loss function; 1.3: gradient norm critera for termination vs loss function; 1.4: iterations vs norm of the gradient

In this investigation, we attempt to understand the effect of hyperparameters on the solution produced by gradient descent. We measure solution quality by measuring the loss function on the solution produced by the gradient descent algorithm with various hyperparameter settings. We find that hyperparameter tuning can have significant effect on the quality of the solution; choosing a correct algorithm is only one step of solving a problem with machine learning – correct choice of hyperparameters must follow as well. The above figures represent the results of our investigation: [FIG 1.1] represents the loss function for both quadratic bowls outlined in the problem set as a function of the learning rate η. As the image makes clear, a small η means the algorithm may fail to converge, or converge on an incorrect value, whereas a large η makes the gradient blow up (the blue line terminates at a smaller value of because the gradient began to grow exponentially until it hit undefined values). It is therefore important to choose hyperparameters such as η with great care, and with respect to each specific problem encountered.

Similarly, loss may remain unaffected by the magnitude of the initial guess vector, or, with small learning rate or non-convex functions, gradient descent may end up in with a solution that is not even close to the true optimum. Some functions are more sensitive to choice of guess vector than others, as is made evident in [FIG 1.2]. In [FIG 1.3], we can see that increasing the norm of the gradient at which we terminate the algorithm produces increasingly less optimal results; this is because the smaller the norm of the gradient, the closer we are to the optimal parameter that minimizes the loss function. And finally, in [FIG 1.4], we see the way the norm of the gradient changes over the iterations the algorithm progresses through when finding the optimal parameters for the negative Gaussian. The fact that the norm of the gradient decreases reflects the fact that the gradient ought to be zero at the optimal solution, at which point the algorithm terminates.

To verify the gradients accuracy, we used the central difference approximation, which tends to be quite close to the actual gradient except in cases where the gradient increases or decreases very rapidly. As the difference step δ becomes smaller, so does the error on the gradient approximation by a factor of δ^2. Because batched gradient descent may not always be viable on large datasets (or loss functions that lack a closed-form solution), we also implemented stochastic gradient descent and compared the number of pointwise gradient computations required for convergence, as well as the optimality of the resulting solution. Stochastic gradient descent converged in 3195 iterations through the entire dataset (so 3195 * len(X) pointwise gradient computations) and had a loss of 623590.175265. Batch gradient descent converged in 3280 iterations (3280 * len(X) pointwise gradient computations) and had a loss of 623499.341758, slightly less than that of SGD.

Problem 2

We are here concerned with approximating a polynomial fit to data given a noisy dataset. The dataset is based on a cosine function (specifically y(x) = cos(πx) + cos(2πx), and we attempt to find a polynomial approximation using both the analytic (maximum-likelihood vector) and batched gradient descent on the sum of squared error (here abbreviated as SSE) loss function. For various degrees of M, we attempt to find the analytic fit: [2.1]: M = 5; [2.2]: M = 2. As might be expected, loss decreases as model complexity increases.

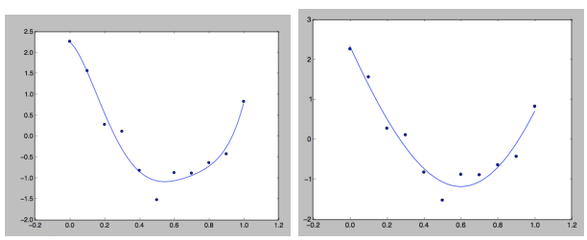

Fig. 2. 2.1: M = 5, analytic fit to points; 2.2: M = 2, analytic fit to points.

Once again, verifying the gradient of SSE with the central difference approximation succeeded; the smaller the δ, the

Reserved for Publication Footnotes

closer the approximation. As we discussed previously, the error of the approximation decreases with δ as δ^2 .

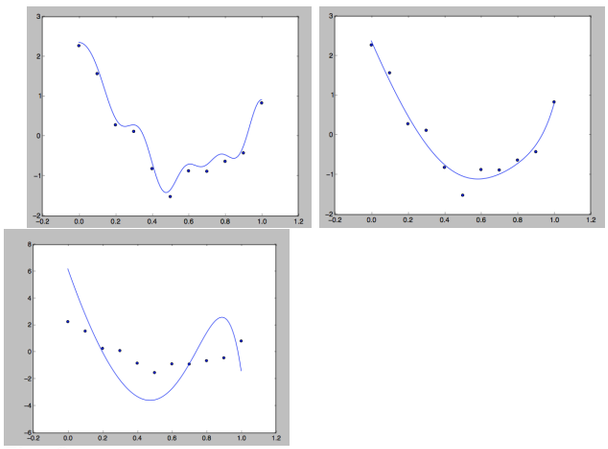

Fig. 3. 2.3: Analytic fit to polynomial, M = 10; 2.4: batch gradient descent fit to polynomial, M = 10; 2.5: stochastic gradient descent fit to polynomial, M = 10.

Next, we attempt gradient descent fit to the polynomial. Illustrated from left to right we see analytic fit, batch gradient descent fit, and stochastic gradient descent fit for M = 10. For batch gradient descent, the algorithm is relatively forgiving of choice of original parameters, and using a small step size is effective for a finely-tuned fit. Too large of a step size causes the gradient to blow up and overshoot the minimum. The convergence threshold need not be too small; even with a relatively large convergence threshold, the algorithm tends to converge on an optimal solution that provides a good fit to the points. However, it can be quite small if needed, because convergence on an optimal value is guaranteed eventually. For stochastic gradient descent, however, the algorithm is fairly forgiving of choice of original parameters, but not as much so as batch gradient descent. Moreover, a large step size can cause the algorithm to take too much of a step in a very wrong direction, while a too small step size may prevent the algorithm from converging in a reasonable timeframe. Stochastic gradient descent cannot be run with too small of a convergence value, because it may not always reach said optimal value but may instead oscillate around the optimal value, but probabilistically with high likelihood it will arrive at a value very close to the optimal value.

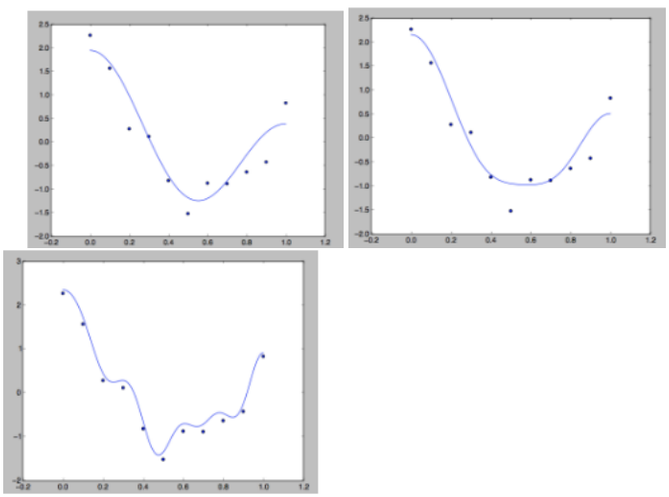

Fig. 4. 2.6: Cosine basis function, M = 2; 2.7: Cosine basis function, M = 5; 2.8: cosine basis function, M = 10

We now extend our linear basis function regression to the cosine basis that forms the true parameters from which the dataset was drawn. From left to right, FIG 2.6: M = 2, FIG 2.7: M = 5, FIG 2.8: M = 10. By and large the values of these models produce are fairly close to the dataset we are given, especially given the small n of the dataset and the noise that was introduced. For M = 2, the parameters are [0.7789928 1.17413213]; for M = 5, the parameters are [0.770386 1.14128366 0.10082616 0.19709081 -0.04918536]; and when M = 10, the parameters are [0.79114136 1.08116276 0.12158152 0.13696992 -0.02843 0.35524566 0.03454684 0.0087985 -0.20058971 0.05680211]. As may be evident, all three parameter vectors are fairly consistent for the cos(πx) and cos(2πx) components, falling fairly close to the actual parameters ([1, 1]). For parameters c in cos(cπx)beyond 1 and 2, they are fairly close to 0 in both the M = 5 and M = 10 result. Given the noise and the small n of the dataset, these results are quite good.

Problem 3

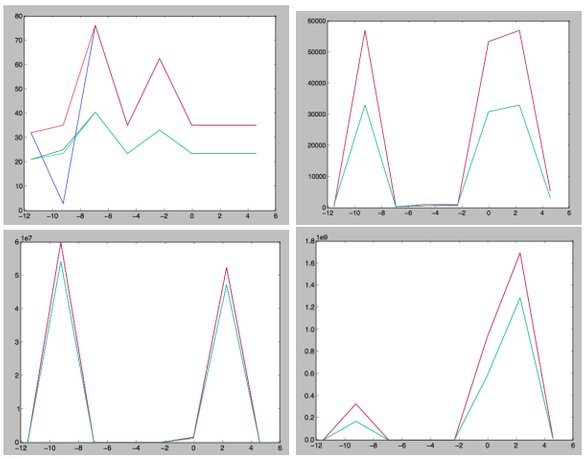

Fig. 5. log(λ) vs ridge loss for: 3.1: M = 1; 3.2: M = 8; 3.3: M = 9; 3.4: M = 10

In this next section, we implement ridge regression for polynomial curve fitting. We also introduce a training/test/validation split in the dataset for further experimental rigor. Because ridge regression accuracy is affected by the hyperparameter λ, as well as the dimensionality of the basis function ϕ, we examine results for various hyperparameters. The above are plots of loss (meaning ridge loss, which penalizes large magnitude of θ, as well as inaccuracy) versus log(λ) for various values of M: 1, 8, 9, and 10. As may be evident, there are a few values of λ and M for which the loss is very low; where M = 1, all the losses are quite small, and for larger values of M there are some intervals of for which the loss is also quite small, and some where the loss is in fact quite large. We posit here that very small λ creates larger losses when it fails to reach a global minimum but instead ends up in a local minimum, and of course large λ may overshoot the global minimum or alternatively cause the gradient to blow up.

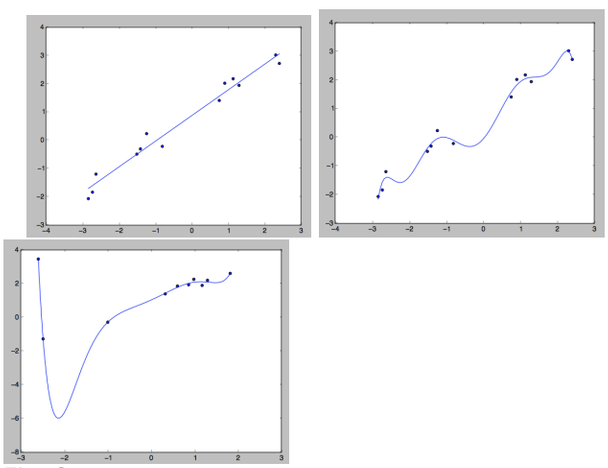

Fig. 6. ridge regression fits on training data for: 3.5: M =1, λ = 1; 3.6: M = 8; λ = 0.1; 3.7: M = 9, λ = 0.1

Displayed above are some fits on for some of the more optimal values of λ and M on training data [FIG. 3.5]: M = 1, λ = 1; [FIG. 3.6]: M = 8, λ = 0.1; [FIG. 3.7]: M = 9; λ = 0.1. (note how good the fits appear to be on the data the model was trained on). However, when we test on the validation data, we find that more complex models quickly begin to fail, especially those with smaller λ.

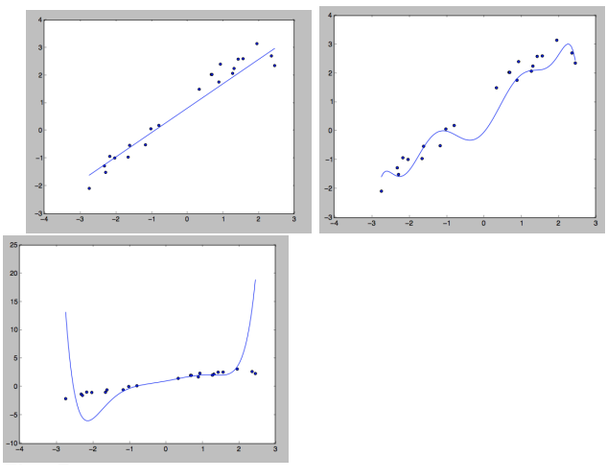

Fig. 7. same models' fits as above, done on training data. 3.8: M =1, λ = 1; 3.9: M = 8; λ = 0.1; 3.10: M = 9, λ = 0.1

We see above the same parameters' fits shown on the validation data; as may be evident from earlier in this discussion, the training set is small, so it is not surprising that overfitting is an issue that tends to emerge with smaller λ (i.e. less penalty for larger θ) and larger M. As we examine the figures, we can see that this is indeed the case; the validation data makes this point especially clear.

Problem 4

Here we compare the accuracy of the LASSO estimator, with various hyperparameters, to the ridge and least-squares estimators. LASSO tends to create sparse estimators, minimizing the equation $\frac{1}{n}\Sigma_{i=1}^{n}(y^{(i)} - w^T\phi(x^{(i)}))^2 + \lambda\Sigma_{j=1}^{M}|w_j|$. Ridge regression also penalizes norm (in this case l^2 norm) of θ, yielding a somewhat sparse estimation. The larger the parameters in either case, the sparser the estimator, because vector norm, of whatever degree, is more heavily penalized in the objective function. As we can see from [4.1], the estimators come quite close to the dataset; the LASSO estimator comes closest; the ridge estimator with large is similarly close; and the ridge estimator with λ = 0 (which is, in essence, a least squares estimator) is less accurate, and there was a great deal of noise added to the data, if the ground truth (true parameters) is to be believed.

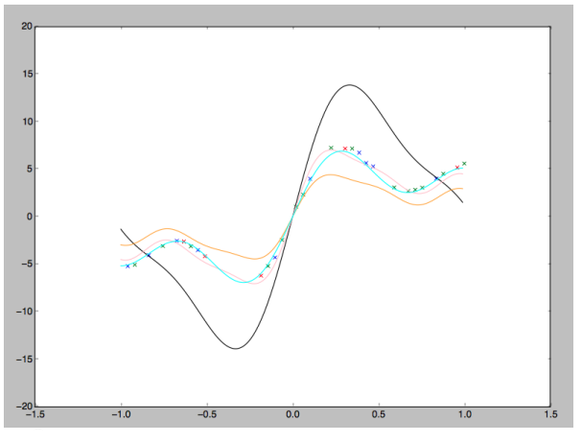

Fig. 8. fits to training data using various estimators. Black: ground truth for θ. Aqua: least squares estimator. Orange: ridge regression. Pink: LASSO