6.867 Problem Set 1

Anonymous Authors

September 30, 2016

1 Gradient Descent

In this section, we minimize a function f by starting at a point $\vec{x} = \vec{x_0}$ and following gradient descent:

$$\vec{x} \rightarrow \vec{x} - \lambda \cdot \nabla f(\vec{x})$$

until $||\nabla f(\vec{x})|| < \epsilon$. We examine the effects of parameters $\vec{x_0}, \lambda, \epsilon$ on convergence rate and accuracy, measured in the number of iterations needed to converge, and the distance from the final $\vec{x}$ to f's minimum. We also examine how the magnitude of the gradient changes over the course of gradient descent.

We perform experiments in which we fix two parameters and vary the third. To parametrize $\vec{x_0}$ in one real variable so we may vary it, we require that $\vec{x_0}$ be on the ray from the minimum of f to $(0,0)$, and parametrize it by its distance to the minimum. We do this for two functions f: a negative multivariate Gaussian $f_1(\vec{x})$ with mean $(10,10)$ and covariance matrix $\left(\begin{smallmatrix} 1000 & 0 \\ 0 & 1000 \end{smallmatrix}\right)$, and a quadratic bowl given by $f_2(\vec{x}) = 10(x_1^2 + x_1x_2 + x_2^2) + 400(x_1 + x_2)$. These have minima $(10,10)$ and $(\frac{80}{3}, \frac{80}{3})$, respectively.

We also perform an experiment, for both functions above, in which we fix $\vec{x_0}, \lambda, \epsilon$ and examine how $|\nabla f(\vec{x})|$ changes over the course of gradient descent. In all cases, to prevent looping, we terminate gradient descent after 5000 iterations if it fails to terminate. Our results are shown in Figures 1 and 2.

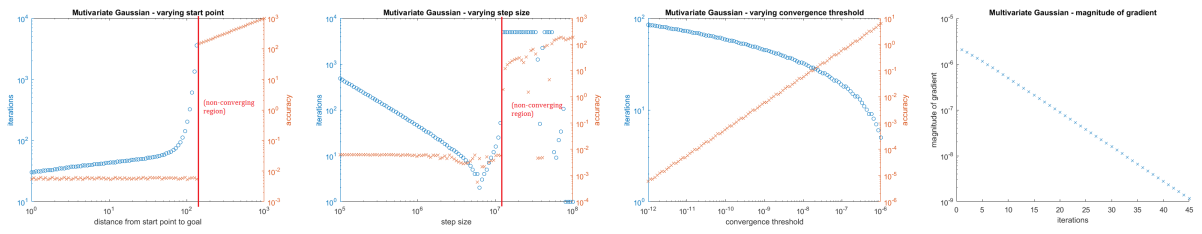
As $\vec{x_0}$ gets further from f's minimum, the runtime generally increases. This is because when $\vec{x_0}$ is farther away, gradient descent has a greater distance to walk before it converges. Exceptions are: on f_1, when $\vec{x_0}$ is far enough, $|\nabla f(\vec{x_0})|$ is smaller than ϵ, so gradient descent does not converge, and runtime is 0; on f_2, when $\vec{x_0}$ is very small, increasing $\vec{x_0}$'s distance decreases runtime because the points close the the minimum have small gradients, leading to smaller step sizes.

Since the algorithm stops when it enters some region near the minimum, which is not affected by the starting point, accuracy is independent of $\vec{x_0}$ provided the algorithm converges.

As λ increases, the runtime decreases to a minimum, then increases. The λ that minimizes runtime is the step size that takes the algorithm immediately to the minimum; smaller and larger λ cause the algorithm to undershoot and overshoot. For too-large λ, the algorithm fails to converge, making the runtime infinite. For the same reasons as for $\vec{x_0}$, accuracy is independent of λ provided the algorithm converges.

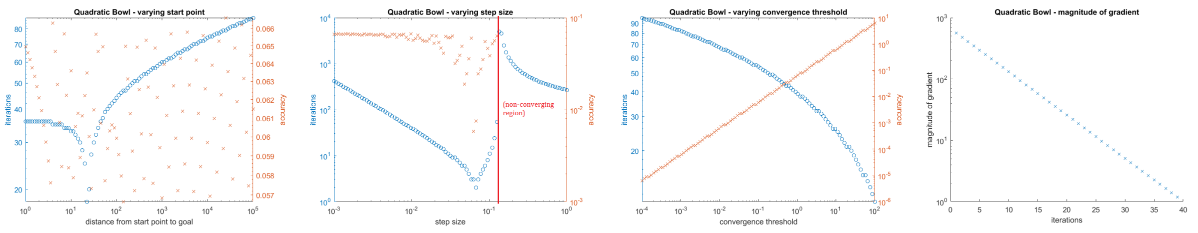
As ϵ increases, the runtime decreases because the set of stopping points is larger, making this set reachable in fewer steps; since the new stopping points are further from the minimum, accuracy decreases.

These trends are shown in Figures 1 and 2, (a-c). Moreover, Figures 1 and 2 (d) show that as gradient descent walks towards the minimum, the magnitude of the gradient decreases approximately exponentially.



(a) Vary $\vec{x_0}$; fix $\lambda = 10^6$, $\epsilon = 10^{-9}$.
(b) Vary λ; fix $\vec{x_0} = (0,0)$, $\epsilon = 10^{-9}$.
(c) Vary ϵ; fix $\vec{x_0} = (0,0)$, $\lambda = 10^6$.
(d) Fix $\vec{x_0} = (0,0)$, $\lambda = 10^6$, $\epsilon = 10^{-9}$. Plotting $|f(\vec{x})|$.

Figure 1: Experiments on f_1.



(a) Vary $\vec{x_0}$; fix $\lambda = 10^{-2}$, $\epsilon = 1$. (b) Vary λ; fix $\vec{x_0} = (0,0)$, $\epsilon = 1$. (c) Vary ϵ; fix $\vec{x_0} = (0,0)$, $\lambda = 10^{-2}$. (d) Fix $\vec{x_0} = (0,0)$, $\lambda = 10^{-2}$, $\epsilon = 1$. Plotting $|f(\vec{x})|$.

Figure 2: Experiments on f_2.

δ	Approx. ∇f ($\times 10^{-6}$)	Error ($\times 10^{-6}$)
30	(-0.932,-0.932)	0.718
20	(-1.187, 1.187)	0.358
10	(-1.372,1.372)	0.096
5	(-1.423,-1.423)	0.024
1	(-1.439 ,-1.439)	0.001

(a) Estimations for $\nabla f_1(0,0) = 10^{-6} \cdot (-1.440, 1.440)$.

δ	Approx. ∇f ($\times 10^{-6}$)	Error ($\times 10^{-6}$)
30	(400,400)	0
20	(400,400)	0
10	(400,400)	0
5	(400,400)	0
1	(400,400)	0

(b) Estimations for $\nabla f_2(0,0) = (400, 400)$.

Table 1: Approximated Gradients of f_1, f_2 at $(0,0)$.

Moreover, we examine the problem of estimating ∇f: if e_i is a unit vector in the ith coordinate, then

$$\nabla f(\vec{x})_i \approx \frac{f(\vec{x} + \delta e_i) - (\vec{x} - \delta e_i)}{2\delta}.$$

This allows us to approximate gradients when a closed form isn't available, and to verify known closed forms. By using this procedure for various δ to approximate gradients of f_1 and f_2 at $(0,0)$, we obtain Table 1.

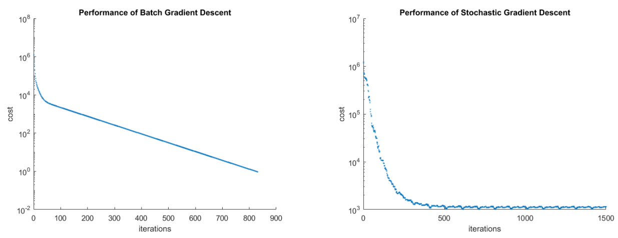
The low errors in Table 1 verify our closed forms for ∇f_1 and ∇f_2.

As shown by Table 1(a), for most functions f the accuracy of the estimated ∇f increases as δ gets smaller, because the partial derivatives of f are the limit of the above expression as $\delta \to 0$. By special properties of quadratic functions, (namely, that the slope of a secant equals the slope of the tangent at the secant's midpoint). our estimated gradients are actually exact for f_2, as shown in Table 1(b).

Finally, we run batch and stochastic gradient descent on the same data set to obtain Figure 3. BGD is much more accurate; the BGD's solution has squared error ≈ 1, while the SGD's solution has squared error $\approx 10^3$. However, SGD is much faster; SGD terminates in 1500 iterations, while BGD terminates in about 800 iterations, each of which takes 100 times the work of an iteration of SGD.

2 Linear Basis Function Regression

In this section, we fit data $(x^{(i)}, y^{(i)})$ with linear models $\hat{y} = \vec{\phi} \cdot \vec{w}$, for some function ϕ and a parameter $\vec{w}$. For now, let $\vec{\phi}(x) = (1, x, \ldots, x^M)$, for some parameter M. By Bishop Equation 3.15, if Φ is the matrix whose



(a) Batch GD. (b) Stochastic GD.

Figure 3: Comparison of Performance of Batch and Stochastic Gradient Descent.

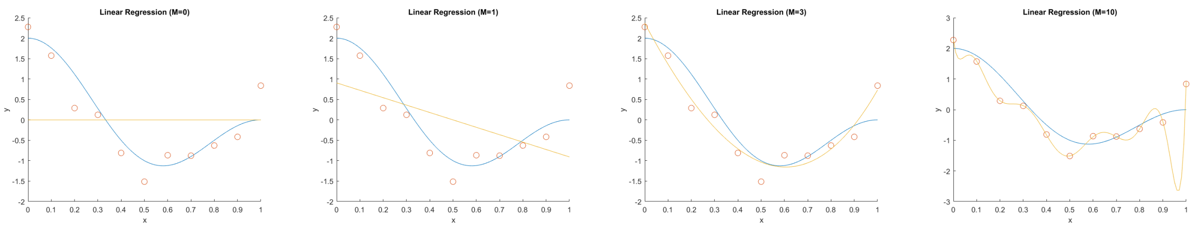
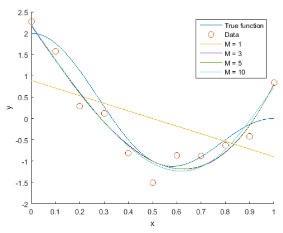
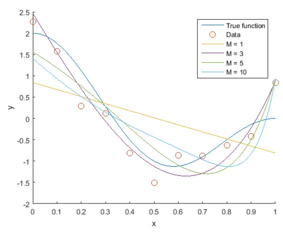


Figure 4: Polynomial fits for various M. Blue curve is the model; yellow curve is the generative function.



(a) Batch gradient descent



(b) Stochastic gradient descent

Figure 5: Polynomial fits given by batch and stochastic gradient descent for various M. The true curve has y-intercept about 2.

rows are $\Phi_j = \phi(x_j)$ ($j \in [1, N]$), then the maximum-likelihood $\vec{w}$, which minimizes the squared error, is

$$\vec{w} = \left(\Phi^T \Phi\right)^{-1} \Phi^T \vec{y}.$$

Using this procedure to find the optimal $\vec{w}$ on the given data, for $M = 0, 1, 3, 10$, yields Figure 4. Note that as our basis of functions gets larger, our models become more expressive. So, the $M = 3$ model more closely follows the data than the $M = 0$ and $M = 1$ models; however when M is too large, our models become too expressive and overfit the data, as shown by the $M = 10$ model.

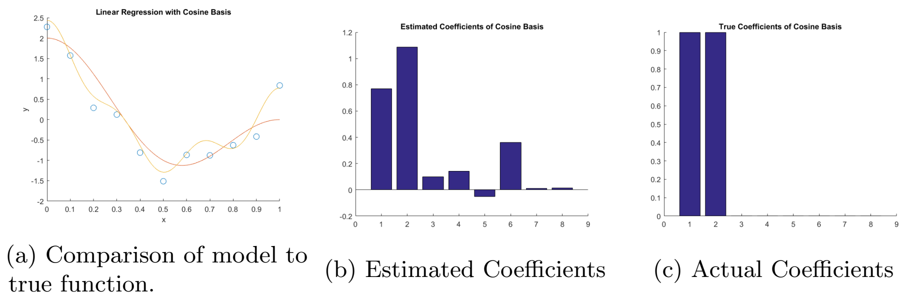
Building off the last section, we can also try using gradient descent to minimize the squared error as a function of $\vec{w}$. Using batch gradient descent with the given data, for $M = 1, 3, 5, 10$ gives the curves in Figure 5(a). These curves were generated when the start vector consisted of entirely 1's, the step size was 4.1×10^{-3}, and the convergence threshold was 10^{-5}. Experimenting with different step sizes and convergence thresholds around these values did not change the results appreciably.

Note that the derived curves for $M = 3, 5, 10$ are nearly identical in this range, unlike in Figure 4, where the curve changes shape and better fits some points as M gets larger. We can attribute this to the gradient being small on the coefficients of $\vec{w}$ corresponding to higher order terms. More specifically, we are trying to minimize

$$\sum_i \left(\vec{\phi}(x^{(i)}) \cdot \vec{w} - y^{(i)}\right)^2 = \sum_i \left(\vec{w_1} + \vec{w_2}(x^{(i)}) + \cdots + w_{\vec{M}+1}(x^{(i)})^M - y^{(i)}\right)^2$$

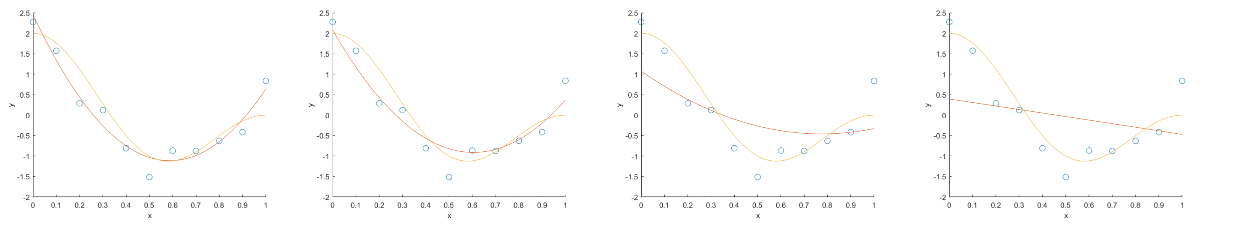
and the $(M+1)$th coordinate of its gradient is of the form $(x^{(1)})^M(\dots) + (x^{(2)})^M(\dots) + \dots$. Since the training data all have $x^{(i)} \in [0, 1]$ and we start with $w_{M+1} = 1$, the Mth coordinate of the gradient will be extremely small, and so over the course of the gradient descent, the $(M+1)$th and other higher order coordinates in $\vec{w}$ will not change appreciably. Furthermore, the higher order terms do not contribute much to the value of $\vec{\phi} \cdot \vec{w}$, and so doing gradient descent when $M = 10$ closely resembles doing gradient descent with $M + 3$, since the effects of having higher order terms in the objective function are negligible because of this training set.

Using the formula in Bishop 3.15, we compute that for $M = 10$, the maximum-likelihood weight vector has higher order coefficients that are on the order of 10^3, which indicates that to get this weight vector with



(a) Comparison of model to true function. (b) Estimated Coefficients (c) Actual Coefficients

Figure 6: Linear basis regression with basis $\{\cos(\pi x), \cos(2\pi x), \ldots, \cos(8\pi x)\}$. Orange curve is the model; yellow curve is the generative function.



(a) $\lambda = 0$ (b) $\lambda = 10^{-2}$ (c) $\lambda = 10^{-1}$ (d) $\lambda = 1$

Figure 7: Models with $M = 2$. The red curve is the model; the yellow curve is the true generative function.

batch gradient descent, we should start with larger weights for the higher order coefficients.

Using stochastic gradient descent with the data for $M = 1, 3, 5, 10$ gives similar results, as in Figure 5(b). The curves in 5(b) were generated when the start vector consisted of entirely 1's, the learning rate was $0.8(\tau + t)^{-\kappa}$ that satisfies the Robbins-Monro condition, and the convergence threshold was 10^{-2}. we As noted in the previous section, SGD yields less accurate estimates than BGD. By the same argument as before, with the same start vectors and range for training data, there is not much appreciable difference between lower M and higher M.

Now, suppose we take $\vec{\phi}(x) = (\cos(\pi x), \cos(2\pi x), \ldots, \cos(M\pi x))$. By applying Bishop Equation 3.15, we get the model in Figure 5(a). Like the high-degree polynomial model, this model is also overfit. As shown in Figure 5(b-c), our maximum-likelihood weight vector's first and second coefficients are close to 1, which is the true vector's coefficient. But, our maximum-likelihood vector also has other coefficients, most notably a sixth coefficient of approximately 0.4. These coefficients, which attempt to compensate for the training data's noise, lead to overfitting.

3 Ridge Regression

As before, let $\vec{\phi}(x) = (1, x, \ldots, x^M)$, for some parameter M. We fit a model $\hat{y} = \vec{\phi} \cdot \vec{w}$, for a parameter $\vec{w}$. We use ridge regression; given parameter λ, we minimize the cost

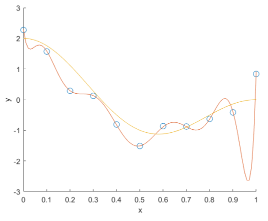
$$\sum_i \left(\vec{\phi}(x^{(i)}) \cdot \vec{w} - y^{(i)}\right)^2 + \lambda ||\vec{w}||^2.$$

By Bishop equation 3.28, if Φ is defined as before, the optimal $\vec{w}$ is

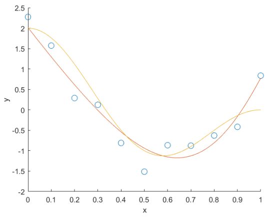
$$\vec{w} = \left(\lambda I + \Phi^T \Phi\right)^{-1} \Phi^T \vec{y}.$$

To examine the effects of the parameters M and λ on the perforamance of this model, we use the above formula to model the data from the previous section. To study the effects of λ on both small and large M, we produce models for $\lambda \in \{0, 10^{-2}, 10^{-1}, 1\}, M \in \{2, 10\}$. The resulting plots are shown in Figures 7 and 8.

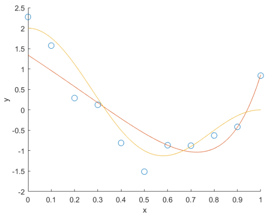
When $M = 2$, the quality of the models monotonically decreases as a function of λ. From this, we conlude that when the polynomial basis is small, our polynomial model is not expressive enough for the data.



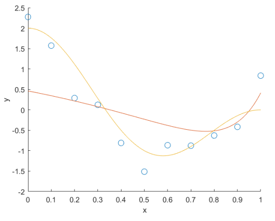
(a) $\lambda = 0$



(b) $\lambda = 10^{-2}$



(c) $\lambda = 10^{-1}$



(d) $\lambda = 1$

Figure 8: Models with $M = 10$. The red curve is the model; the yellow curve is the true generative function.

	$M = 2$		$M = 6$		$M = 10$	
λ	Train	Val	Train	Val	Train	Val
10^{-3}	0.875	2.372	0.557	4.062	0.032	36.80
10^{-2}	0.875	2.388	0.557	4.062	0.057	38.55
10^{-1}	0.878	2.551	0.579	3.890	0.234	15.33
1	1.121	4.359	0.804	4.801	0.690	7.990

(a) Training set A.

	$M = 2$		$M = 6$		$M = 10$	
λ	Train	Val	Train	Val	Train	Val
10^{-3}	9.826	30.29	0.092	316.8	0.069	970400
10^{-2}	9.827	30.20	0.100	394.0	0.077	228500
10^{-1}	9.838	29.45	0.130	338.2	0.119	876.4
1	10.41	28.68	0.631	576.0	0.466	6186

(b) Training set B.

Table 2: Training and validation scores for various M, λ.

Increasing λ penalizes large $\vec{w}$ and makes the space of viable models even more restricted, making the model's performance worse.

When $M = 10$, the quality of the models increases for small λ until $\lambda = 10^{-2}$, and then decreases for larger λ. When the polynomial basis is large, our model is too expressive. Unless λ is large enough, $\vec{w}$ is allowed to become large and overfit the data. Increasing λ from 0 to 10^{-2} improves the model's performance by punishes large $\vec{w}$. However, when λ becomes too large, it restricts the space of models too strongly.

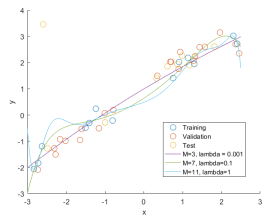
We run ridge regression with training sets A and B, validating with the given validation set, yielding Table 2. When we train on Training Set A, the model with the best validation score is $(M, \lambda) = (3, 10^{-3})$; when we train on Training Set B, the model with the best validation score is $(M, \lambda) = (3, 1)$. When we test these models on the other training set, we get squared errors of 25.76 and 37.38.

Figure 9 shows graphs of optimal models for pairs (M, λ). For both training sets, as M gets larger the model performs worse, because all but one data point lie on a fairly simple curve. Moreover, when the regularization term λ is not large enough, the curve overfits the data – this is especially true on Training Set B, when the upper-left outlier strongly influences the model.

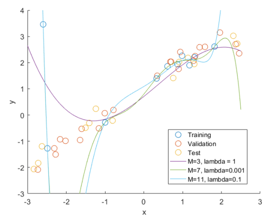
4 Sparsity and LASSO

In this section, we continue exploring fitting a model $\vec{y} = \vec{\phi} \cdot \vec{w}$ for some parameter $\vec{w}$. More specifically, we will try fitting when we believe the true $\vec{w}$ underlying our observations are sparse.

In the previous section's ridge regression, we use a quadratic regularizer to regularize the coefficients of the weight vector. However, when faced with weight vectors that achieve approximately the same value on

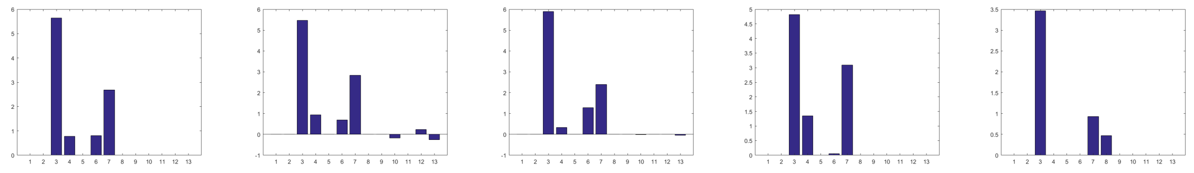


(a) Models using Training Set A.



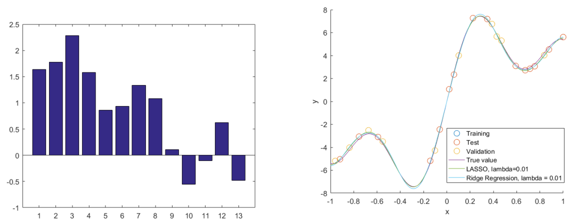
(b) Models using Training Set B.

Figure 9: Graphs of Optimal Models for each M.



(a) True weights (b) $\lambda = 0.001$ (c) $\lambda = 0.01$ (d) $\lambda = 0.1$ (e) $\lambda = 1$

Figure 10: Experiments with λ during LASSO. Plotting components of resulting weight vectors



(a) $\lambda = 0.01$

(b) Comparing results of LASSO and ridge regression

Figure 11: Experiments with ridge regression.

the sum of squared error term, the ℓ_2 norm in the regularizer favors those with weights that are smaller and close in absolute value over those with a few large weights and a few small weights.

Instead, suppose we use LASSO, which uses a linear regularizer, based on the ℓ_1 norm to regularize the weight vector. We minimize instead

$$\sum_i \left(\vec{\phi}(x^{(i)}) \cdot \vec{w} - y^{(i)}\right)^2 + \lambda||\vec{w}||_1.$$

With the ℓ_1 norm, large weights are not disfavored as much and so we have more liberty to pick some large weights as our estimate, as long as they are offset by smaller terms. These observations suggest that LASSO will be better for sparse estimation than ridge regression.

To test this concretely, we will use LASSO to do our own sparse estimation on some training data and compare it to ridge regression. In particular, our data is drawn from the distribution $y = \vec{w}_{true}^T \phi(x)$ for $x, y \in \mathbb{R}$, $\vec{w}_{true} \in \mathbb{R}^{13}$ and $\phi(x) = x, \sin(0.4\pi x), \sin(0.4\pi x \times 2), \sin(0.4\pi x \times 3), \ldots, \sin(0.4\pi x \times 12)$. The true weights in $\vec{w}_{true}$ are shown in Figure 10(a).

To pick the best λ for running LASSO, we produce models for $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 1\}$, as shown in Figure 10.

We can see that LASSO gives reasonably sparse and accurate estimates of the true weights. When λ is small (e.g. 10^{-3}), our LASSO model still has many nonzero terms, since the ℓ_1 norm term is weighted less by smaller λ. The model then becomes sparser and closer to the true weights as λ increases to 10^{-2} and 10^{-1}.

However, once λ is large (e.g. 1) the ℓ_1 norm term dominates the objective function, and is too restrictive on the resulting model. In this case, the weights are sparse, but much smaller than the true weights, indicating that LASSO favored weights with a low ℓ_1 norm overall over weights that reduce the squared error.

Using $\lambda = 0.01$, which yields good sparse estimates with LASSO, we plot $\vec{\phi} \cdot \vec{w}_{LASSO}$ and $\vec{\phi} \cdot \vec{w}_{ridge}$ in Figure 11(b). The LASSO curve estimates the true curve slightly better than the ridge regression curve, particularly around the extrema.

Furthermore, in Figure 11(a), we see that since ridge regression favors weights that are more uniform in absolute value, the large weight for the third basis function $\sin(0.4\pi x \times 2)$ in ϕ is interpreted approximately as some linear cominbation of $x, \sin(0.4\pi x), \sin(0.4\pi x \times 2), \sin(0.4\pi x \times 3)$, with smaller weights in the resulting graph.

We conclude that LASSO provides a more accurate fit of both the individual weights and the overall curve, at least when the true weights are sparse.