CIS 520, Machine Learning, Fall 2019
Homework 3

Matthew Scharf

September 30, 2019

1 Regularization Penalties

1. These are the conditions for OLS so:
   $w* = (X^T X)^{-1} X^T y = [0.889, -0.826, 4.190]$
2. This is L_2 regularized regression so:
   $w* = (X^T X + I)^{-1} X^T y = [0.865, -0.821, 4.122]$
3. Using sklearn's Lasso function, we get:
   $w* = [0.87491626, -0.81824434, 4.18288764]$
4. We'll split this up into 8 cases:

X_1	X_2	X_3	Error
x	x	x	3107.790
x	x		4136.005
	x	x	3131.135
x		x	3138.074
x			4225.236
	x		4160.100
		x	3171.074
			12800.818

   The best error is found when using all 3 features:
   $w* = (X^T X)^{-1} X^T y = [0.889, -0.826, 4.190]$
   Error = 3107.790
5. The OLS gives us our baseline $w*$. Then, the L_1 and L_2 regularized regression shrink all the values of $w*$ a little bit by adding a penalty for the magnitude of the weights. L_2 especially shrank the largest weight. The L_1 norm especially shrank the 2 smaller weights. For L_0 we end up getting the OLS $w*$ simply because the penalty here for the weights is so small in comparison to the regression penalty that it ends up being optimal to just do OLS regression.
6. (a) 0.00613
   (b) i. I expect the training error to increase with the number of samples. As N gets very large, it will hit some asymptote when the linear model is fitting the underlying distribution as best as it can. At that point, the training error will no longer depend on N.
   ii. I expect $||w_{mle}||_2^2$ to begin high for low N as the weights will behave erratically, and then as N grows, $||w_{mle}||_2^2$ will shrink to some asymptotic optimal as $w*$ fits the underlying distribution better and better. At that point, the $||w_{mle}||_2^2$ will no longer depend on N.
   (c) $\lambda = 5$
   error = 0.8522798449717197
   (d) $\lambda = 30$
   error = 0.44762258653852666

2 Feature Selection

1. Streamwise regression.
   (a) Forward:
      i. starting error: 98.0 coeff: [0, 0, 0]
      ii. error: 10.033 coeff: [3.833, 0, 0]
      iii. error: 0.764 coeff: [5.727, -2.273, 0]
      iv. error: 0.6 coeff: [50, -18, -10,]
   (b) Backward:
      i. starting error: 98.0 coeff: [0, 0, 0]
      ii. error: 0.855 coeff: [0, 0, 1.245]
2. Stepwise regression.
   (a) $Err_{old} = 98$
   (b) i. error: 10.033 coeff: [3.833, 0, 0]
      ii. error: 60.7 coeff: [0, 2.5, 0]
      iii. error: 0.855 coeff: [0, 0, 1.245]
   (c) Pick x_3: $Err_{old} = 0.855$
   (d) i. error: 0.876 coeff: [-0.642, 0, 1.436]
      ii. error: 0.864 coeff: [0, -0.236, 1.292]
      Both of these errors are higher than the Err_{old} so we should halt here.
   (e) FINAL: error: 0.855 coeff: [0, 0, 1.245]
3. Findings:
   Streamwise regression is greedy and dependent on the ordering of the features you add, which is not a desirable attribute for an algorithm. Stepwise is slightly better because it searches across all features equally, however is still greedy and therefore misses out on global minimums (as seen in the above example). That being said, where p is the number of features in the data set, streamwise has complexity p while stepwise has complexity p^2. Furthermore, the less data points you have, the more noise you have, and the higher the possibility that these greedy searches will fail to find the global minimum.

3 Kernel Regression

1. Build the model. (auto-graded only)
2. Analysis of the model.
   Figures:

Figure 1: $\sigma = 0.01$

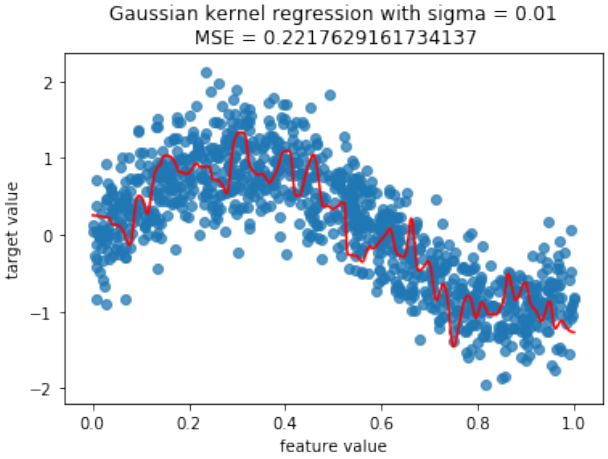


Figure 2: $\sigma = 0.05$

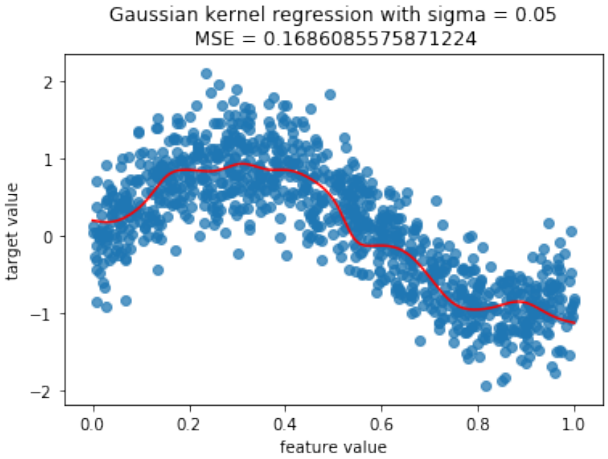


Figure 3: $\sigma = 0.1$

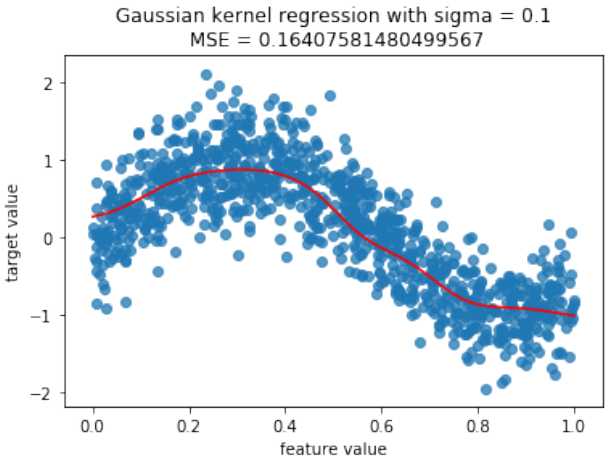


Figure 4: $\sigma = 0.15$

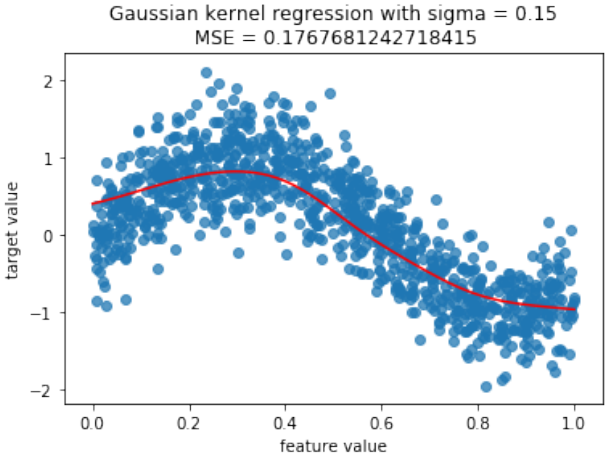


Figure 5: $\sigma = 0.2$

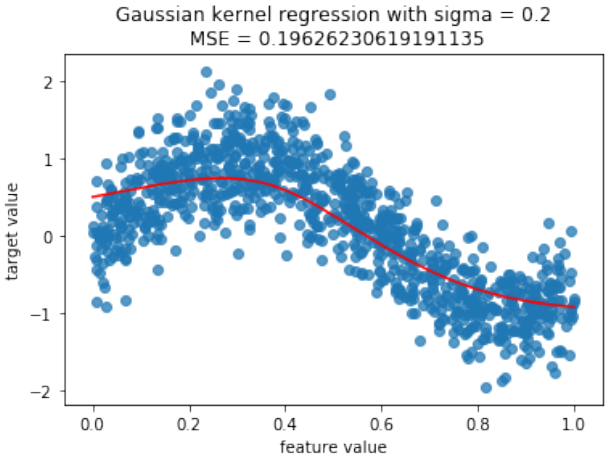


Figure 6: $\sigma = 0.5$

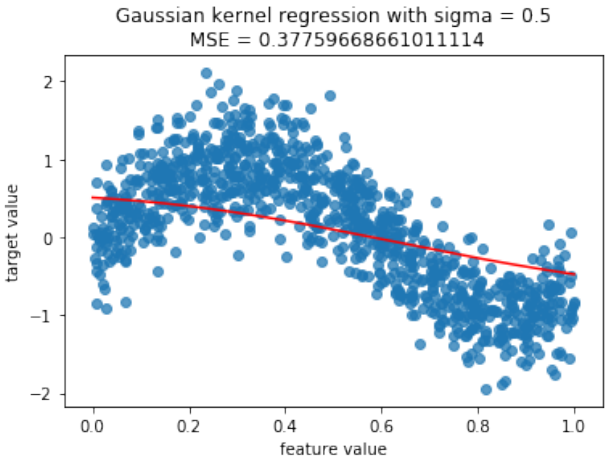
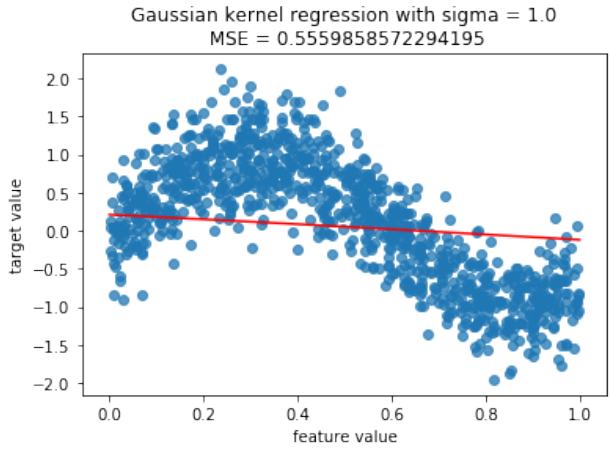


Figure 7: $\sigma = 1.0$



$\sigma = .1$ has the lowest error: $MSE = 0.1641$

Like λ for linear regression, σ is a regularization hyperparameter (inverse measure of complexity) for the kernel regression. Very small sigma values cause the model to overfit while high sigma values are underfitting the data. The model when $\sigma = .05$ actually appears to be a very good fit for the data. This is in contrast to linear model we fit in HW 2 which very clearly underfit the data.

4 Gradient Descent on Logistic Regression

1. Logistic regression coefficient: [8.77265821e-05, 3.21052485e-05, -8.33510775e-06, -6.45718076e-04, -1.03204479e-03, 3.75017388e-03, 9.91131009e-06, -7.39096539e-06, -3.97844092e-06, -1.54729123e-03, -7.71034389e-07]

   Test Accuracy: 0.81

2. (a) Likelihood: $\prod_{i=1}^{n} f(x_i)$ where

$$f(x_i) = \begin{cases} h_w(x_i), & \text{if } y_i = 1 \\ 1 - h_w(x_i), & \text{if } y_i = 0 \end{cases}$$

   (b) Log Likelihood: $\sum_{i=1}^{n} log(f(x_i))$ where

$$f(x_i) = \begin{cases} h_w(x_i), & \text{if } y_i = 1 \\ 1 - h_w(x_i), & \text{if } y_i = 0 \end{cases}$$

   $= \sum_{i=1}^{n} g(x_i)$ where

$$g(x_i) = \begin{cases} -log(1 + e^{-w^T x_i}), & \text{if } y_i = 1 \\ -w^T x_i - log(1 + e^{-w^T x_i}), & \text{if } y_i = 0 \end{cases}$$

   (c) Gradient: $\frac{dLL}{dw} = \sum_{i=1}^{n} h(x_i)$ where

$$h(x_i) = \begin{cases} x_i \frac{e^{-w^T x_i}}{1+e^{-w^T x_i}}, & \text{if } y_i = 1 \\ -x_i \frac{(1}{1+e^{-w^T x_i}}, & \text{if } y_i = 0 \end{cases}$$

Then,

$$h(x_i) = \begin{cases} x_i(1 - s(w^T x_i)), & \text{if } y_i = 1 \\ x_i(-s(w^T x_i)), & \text{if } y_i = 0 \end{cases}$$

where $s(x) = \frac{1}{1+e^{-x}}$.

Note that in matrix form, this is is equivalent to saying:

$$\frac{dLL}{dw} = (Y - s(Xw))^T X$$

where I am applying $s(x)$ element-wise to the vector, Xw.

So, the weight update formula is as follows:

$$w_{t+1} = w_t + \eta((Y - s(Xw))^T X)$$

3. Logistic SGD coefficients: [-0.03225568 0.05192955 -0.01114289 -0.32907494 -0.97632477 3.46687457 0.01545103 -0.01559378 -0.00672916 -1.82143744]

   Intercept: -0.000908

   Training Accuracy: 0.666

   Test Accuracy: 0.67

4. Logistic AdaGrad coefficients: [8.56662095e-05 7.32612582e-03 -4.06501636e-03 -1.84267336e-04 -1.05933126e-03 4.09573947e-03 7.04321246e-03 -7.02779082e-03 -6.96839213e-03 -2.22061425e-03]

   Intercept: -0.00222

   Training Accuracy: 0.82375

   Test Accuracy: 0.815

5. Adagrad clearly has the better accuracy on the test set at the end. This is due to the fact that the gradient is a bit unstable and so in order to have a nice convergence, you must reduce the learning rate (as is done in Adagrad). The SGD does at some points hit the same accuracy as Adagrad but then continues to oscillate away from the optimal.

Figure 8: Accuracy vs. iteration for SGD and AdaGrad (2 points)