CPSC 340 Assignment 3 (due Friday, Feb 9 at 9:00pm)

Instructions

Rubric: {mechanics:3}

The above points are allocated for following the general homework instructions on the course homepage.

Contents

1 Vectors, Matrices, and Quadratic Functions

The first part of this question makes you review basic operations on vectors and matrices. If you are rusty on basic vector and matrix operations, see the notes on linear algebra on the course webpage. If you have time and want a deeper refresher on linear algebra, I have heard good things about the video series Essence of Linear Algebra at `https://youtu.be/kjBOesZCoqc` and the e-book Immersive Linear Algebra at `http://immersivemath.com/ila/index.html`. We will continue to use linear algebra heavily throughout the rest of the course.

1.1 Basic Operations

Rubric: {reasoning:3}

Using the definitions below,

$$\alpha = 5, \quad x = \begin{bmatrix} 2 \\ -3 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \quad z = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & -2 \end{bmatrix},$$

evaluate the following expressions (show your work, but you may use answers from previous parts to simplify calculations):

1. x^Tx.
   Answer: $x^Tx = \begin{bmatrix} 2 & -3 \end{bmatrix} \begin{bmatrix} 2 \\ -3 \end{bmatrix} = 4 + 9 = 13$

2. $\|x\|^2$.
   Answer: $\|x\|^2 = 4 + 9 = 13$

3. $x^T(x + \alpha y)$.
   Answer:$x^T(x + \alpha y) = 13 + 5 * (2 - 12) = -37$

4. Ax
   Answer:$Ax = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & -2 \end{bmatrix} \begin{bmatrix} 2 \\ -3 \end{bmatrix} = \begin{bmatrix} -4 \\ -5 \\ 12 \end{bmatrix}$

5. z^TAx

   Answer:$z^TAx = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} -4 \\ -5 \\ 12 \end{bmatrix} = -8 + 12 = 4$

6. A^TA.

   Answer:$A^TA = \begin{bmatrix} 1+4+9 & 2+6-6 \\ 2+6-6 & 4+9+4 \end{bmatrix} = \begin{bmatrix} 14 & 2 \\ 2 & 17 \end{bmatrix}$

If $\{\alpha, \beta\}$ are scalars, $\{x, y, z\}$ are real-valued column-vectors of length d, and $\{A, B, C\}$ are real-valued $d \times d$ matrices, state whether each of the below statements is true or false in general and give a short explanation.

7. $yy^Ty = \|y\|^2y$.

   Answer: True. $yy^Ty = y\|y\|^2 = \|y\|^2y$

8. $x^TA^T(Ay + Az) = x^TA^TAy + z^TA^TAx$.

   Answer: True. $x^TA^T(Ay + Az) = x^TA^TAy + x^TA^TAZ = x^TA^TAy + z^TA^TAx$

9. $x^T(B + C) = Bx + Cx$.

   Answer:False. $x^T(B + C) = x^TB + x^TC \neq Bx + Cx$

10. $(A + BC)^T = A^T + C^TB^T$.

    Answer: True. $(A + BC)^T = A^T + C^TB^T$

11. $(x - y)^T(x - y) = \|x\|^2 - x^Ty + \|y\|^2$.

    Answer: False. $(x - y)^T(x - y) = \|x\|^2 + \|y\|^2 - y^Tx - x^Tyx \neq \|x\|^2 - x^Ty + \|y\|^2$

12. $(x - y)^T(x + y) = \|x\|^2 - \|y\|^2$.

    Answer: True. $(x - y)^T(x + y) = \|x\|^2 - \|y\|^2 + x^Ty - y^Tx = \|x\|^2 - \|y\|^2$

Hint: check the dimensions of the result, and remember that matrix multiplication is generally not commutative.

1.2 Converting to Matrix/Vector/Norm Notation

Rubric: {reasoning:2}

Using our standard supervised learning notation (X, y, w) express the following functions in terms of vectors, matrices, and norms (there should be no summations or maximums).

1. $\sum_{i=1}^n |w^Tx_i - y_i|$.

   Answer: $\sum_{i=1}^n |w^Tx_i - y_i| = \|Xw - y\|_1$

2. $\max_{i\in\{1,2,\dots,n\}} |w^Tx_i - y_i| + \frac{\lambda}{2}\sum_{j=1}^d w_j^2$.

   Answer: $\max_{i\in\{1,2,\dots,n\}} |w^Tx_i - y_i| + \frac{\lambda}{2}\sum_{j=1}^d w_j^2 = \|Xw - y\|_\infty + \frac{\lambda}{2}\|w\|_2$

3. $\sum_{i=1}^n z_i(w^Tx_i - y_i)^2 + \lambda\sum_{j=1}^d |w_j|$.

   Answer: $\sum_{i=1}^n z_i(w^Tx_i - y_i)^2 + \lambda\sum_{j=1}^d |w_j| = (Xw - y)^TZ(Xw - y) + \lambda\|w\|_1$

You can use Z to denote a diagonal matrix that has the values z_i along the diagonal.

1.3 Minimizing Quadratic Functions as Linear Systems

Rubric: {reasoning:3}

Write finding a minimizer w of the functions below as a system of linear equations (using vector/matrix notation and simplifying as much as possible). Note that all the functions below are convex so finding a w with $\nabla f(w) = 0$ is sufficient to minimize the functions (but show your work in getting to this point).

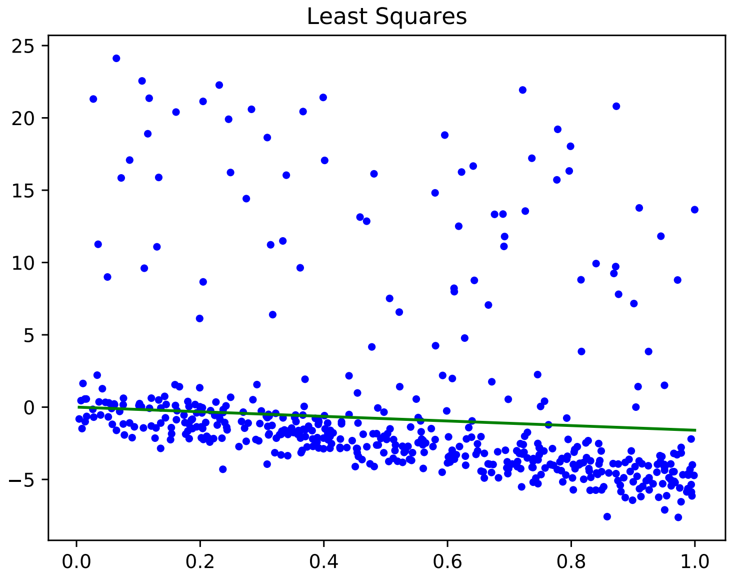
1. $f(w) = \frac{1}{2}\|w - v\|^2$.
   Answer: $f(w) = \frac{1}{2}(\|w\|^2 + \|v\|^2 - 2w^T v).\nabla f(w) = w - v$. So when $w = v$, $\nabla f(w) = 0$.
2. $f(w) = \frac{1}{2}\|w\|^2 + w^T X^T y$.
   Answer: $f(w) = \frac{1}{2}\|w\|^2 + w^T X^T y.\nabla f(w) = w + X^T y$. So when $w + X^T Y = 0$, $\nabla f(w) = 0$.
3. $f(w) = \frac{1}{2}\sum_{i=1}^n z_i(w^T x_i - y_i)^2$.
   Answer: $f(w) = \frac{1}{2}w^T X^T Z X w - w^T X^T Z y + y^T Z y.\nabla f(w) = X^T Z X w - X^T Z y$. So when $X^T Z X w - X^T Z y = 0$, $\nabla f(w) = 0$.

Above we assume that v is a $d \times 1$ vector.

Hint: Once you convert to vector/matrix notation, you can use the results from class to quickly compute these quantities term-wise. As a sanity check for your derivation, make sure that your results have the right dimensions.

2 Robust Regression and Gradient Descent

If you run `python main.py -q 2`, it will load a one-dimensional regression dataset that has a non-trivial number of 'outlier' data points. These points do not fit the general trend of the rest of the data, and pull the least squares model away from the main downward trend that most data points exhibit:



Note: we are fitting the regression without an intercept here, just for simplicity of the homework question. In reality one would rarely do this. But here it's OK because the "true" line passes through the origin (by design). In Q3.1 we'll address this explicitly.

2.1 Weighted Least Squares in One Dimension

Rubric: {code:3}

One of the most common variations on least squares is *weighted* least squares. In this formulation, we have a weight z_i for every training example. To fit the model, we minimize the weighted squared error,

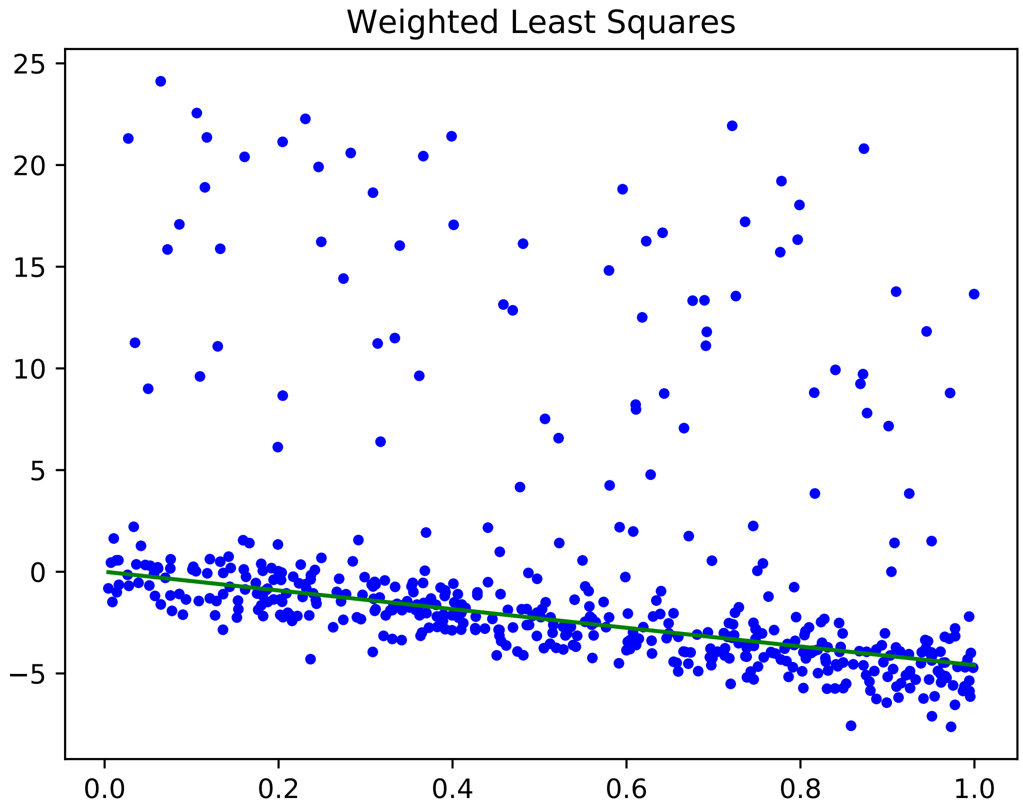
$$f(w) = \frac{1}{2}\sum_{i=1}^{n} z_i(w^T x_i - y_i)^2.$$

In this formulation, the model focuses on making the error small for examples i where z_i is high. Similarly, if z_i is low then the model allows a larger error.

Complete the model class, *WeightedLeastSquares*, that implements this model (note that Q1.3.3 asks you to show how this formulation can be solved as a linear system). Apply this model to the data containing outliers, setting $z = 1$ for the first 400 data points and $z = 0.1$ for the last 100 data points (which are the outliers). Hand in your code and the updated plot.

Answer:
Link to code: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a3/blob/master/code/linear_model.py`

Updated Plot:



2.2 Smooth Approximation to the L1-Norm

Rubric: {reasoning:3}

Unfortunately, we typically do not know the identities of the outliers. In situations where we suspect that there are outliers, but we do not know which examples are outliers, it makes sense to use a loss function that is more robust to outliers. In class, we discussed using the sum of absolute values objective,

$$f(w) = \sum_{i=1}^{n} |w^T x_i - y_i|.$$

This is less sensitive to outliers than least squares, but it is non-differentiable and harder to optimize. Nevertheless, there are various smooth approximations to the absolute value function that are easy to optimize. One possible approximation is to use the log-sum-exp approximation of the max function[1]:

$$|r| = \max\{r, -r\} \approx \log(\exp(r) + \exp(-r)).$$

Using this approximation, we obtain an objective of the form

$$f(w) = \sum_{i=1}^{n} \log\left(\exp(w^T x_i - y_i) + \exp(y_i - w^T x_i)\right).$$

[1] Other possibilities are the Huber loss, or $|r| \approx \sqrt{r^2 + \epsilon}$ for some small ϵ.

which is smooth but less sensitive to outliers than the squared error. Derive the gradient ∇f of this function with respect to w. You should show your work but you do <u>not</u> have to express the final result in matrix notation.

$$f(w)=\sum_{i=1}^{n}\log\left(\exp(w^Tx_i-y_i)+\exp(y_i-w^Tx_i)\right).$$

Answer:

$$\nabla f(w)=\sum_{i=1}^{n}\frac{1}{(\exp(w^Tx_i-y_i)+\exp(y_i-w^Tx_i))}\times\frac{df}{dw}\left(\exp(w^Tx_i-y_i)+\exp(y_i-w^Tx_i)\right).$$

$$=\sum_{i=1}^{n}\frac{1}{(\exp(w^Tx_i-y_i)+\exp(y_i-w^Tx_i))}\times x_{\mathrm{i}}(\exp(w^Tx_i-y_i)-\exp(y_i-w^Tx_i).$$

$$=\sum_{i=1}^{n}x_{\mathrm{i}}\times\frac{((\exp(w^Tx_i-y_i)-\exp(y_i-w^Tx_i))}{(\exp(w^Tx_i-y_i)+\exp(y_i-w^Tx_i))}.$$

2.3 Robust Regression

Rubric: {code:2,reasoning:1}

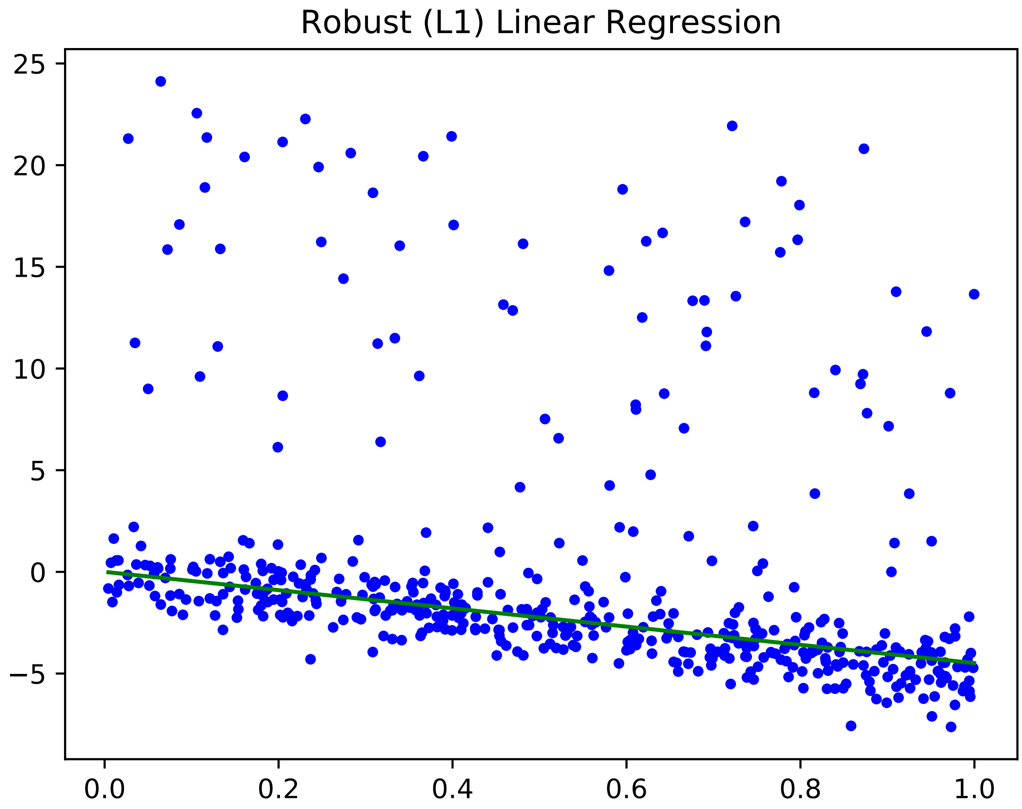
The class *LinearModelGradient* is the same as *LeastSquares*, except that it fits the least squares model using a gradient descent method. If you run `python main.py -q 2.3` you'll see it produces the same fit as we obtained using the normal equations.

The typical input to a gradient method is a function that, given w, returns $f(w)$ and $\nabla f(w)$. See *funObj* in *LinearModelGradient* for an example. Note that the *fit* function of *LinearModelGradient* also has a numerical check that the gradient code is approximately correct, since implementing gradients is often error-prone.[2]

An advantage of gradient-based strategies is that they are able to solve problems that do not have closed-form solutions, such as the formulation from the previous section. The class *LinearModelGradient* has most of the implementation of a gradient-based strategy for fitting the robust regression model under the log-sum-exp approximation. The only part missing is the function and gradient calculation inside the *funObj* code. Modify *funObj* to implement the objective function and gradient based on the smooth approximation to the absolute value function (from the previous section). Hand in your code, as well as the plot obtained using this robust regression approach.

Answer:
Link to code: https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a3/blob/master/code/linear_model.py
Updated Plot:

[2] Sometimes the numerical gradient checker itself can be wrong. See CPSC 303 for a lot more on numerical differentiation.



3 Linear Regression and Nonlinear Bases

In class we discussed fitting a linear regression model by minimizing the squared error. In this question, you will start with a data set where least squares performs poorly. You will then explore how adding a bias variable and using nonlinear (polynomial) bases can drastically improve the performance. You will also explore how the complexity of a basis affects both the training error and the test error. In the final part of the question, it will be up to you to design a basis with better performance than polynomial bases.

3.1 Adding a Bias Variable

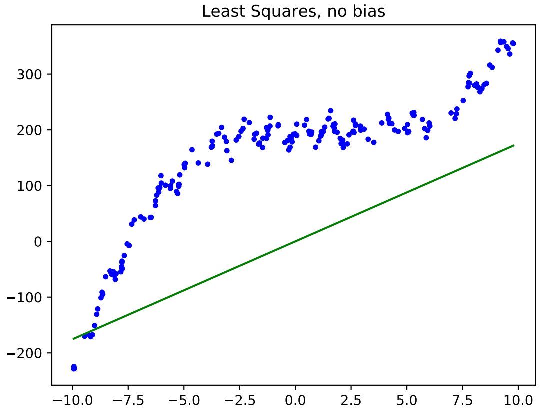
Rubric: {code:3,reasoning:1}

If you run `python main.py -q 3`, it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Report the training error.
4. Report the test error (on a dataset not used for training).

5. Draw a figure showing the training data and what the linear model looks like.

Unfortunately, this is an awful model of the data. The average squared training error on the data set is over 28000 (as is the test error), and the figure produced by the demo confirms that the predictions are usually nowhere near the training data:



The y-intercept of this data is clearly not zero (it looks like it's closer to 200), so we should expect to improve performance by adding a *bias* (a.k.a. intercept) variable, so that our model is

$$y_i = w^T x_i + w_0.$$

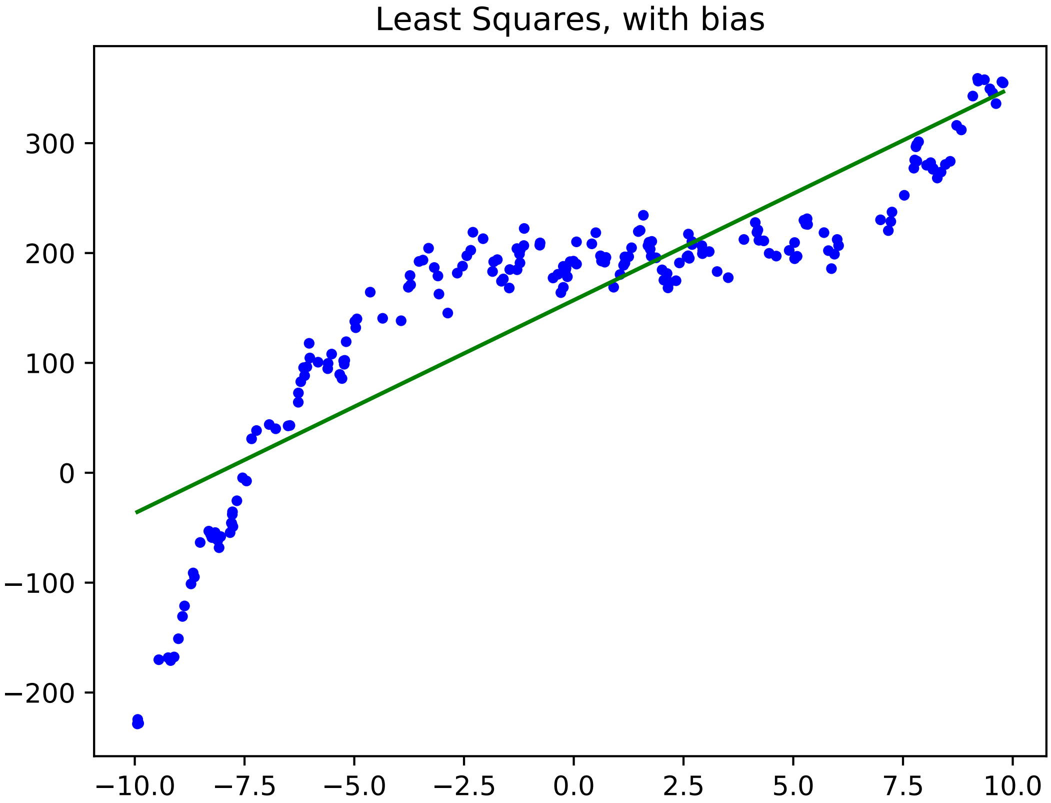
instead of

$$y_i = w^T x_i.$$

In file *linear_model.py*, complete the class, *LeastSquaresBias*, that has the same input/model/predict format as the *LeastSquares* class, but that adds a *bias* variable (also called an intercept) w_0 (also called β in lecture). Hand in your new class, the updated plot, and the updated training/test error.

Hint: recall that adding a bias w_0 is equivalent to adding a column of ones to the matrix X. Don't forget that you need to do the same transformation in the *predict* function.

Answer:
Link to code: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a3/blob/master/code/linear_model.py`
Updated Plot:



Updated Training Error: 3551.3
Updated Test Error: 3393.9

3.2 Polynomial Basis

Rubric: {code:4,reasoning:1}

Adding a bias variable improves the prediction substantially, but the model is still problematic because the target seems to be a *non-linear* function of the input. Complete *LeastSquarePoly* class, that takes a data vector x (i.e., assuming we only have one feature) and the polynomial order p. The function should perform a least squares fit based on a matrix Z where each of its rows contains the values $(x_i)^j$ for $j = 0$ up to p. E.g., *LeastSquaresPoly.fit(x,y)* with $p = 3$ should form the matrix

$$Z = \begin{bmatrix} 1 & x_1 & (x_1)^2 & (x_1)^3 \\ 1 & x_2 & (x_2)^2 & (x_2)^3 \\ \vdots & & & \\ 1 & x_n & (x_n)^2 & (x_N)^3 \end{bmatrix},$$

and fit a least squares model based on it. Hand in the new class, and report the training and test error for $p = 0$ through $p = 10$. Explain the effect of p on the training error and on the test error.

Note: you should write the code yourself; don't use a library like sklearn's `PolynomialFeatures`.

Answer:
Link to code: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a3/blob/master/code/linear_model.py`
Training and test error for p=0 through p-10:

Table 1: Training And Test Error For P's 0-10

Value of P	Training Error	Test Error
0	15480.5	14390.8
1	3551.3	3393.9
2	2168.0	2480.7
3	252.0	242.8
4	251.5	242.1
5	251.1	239.5
6	248.6	246.0
7	247.0	242.9
8	241.3	246.0
9	235.8	259.3
10	235.1	256.3

As the value of P increases, the training error decreases but the approximation error increases. From p=0 to p=5 the test error increases, but from p=5 to p=9 the test error increases

4 Non-Parametric Bases and Cross-Validation

Unfortunately, in practice we often don't know what basis to use. However, if we have enough data then we can make up for this by using a basis that is flexible enough to model any reasonable function. These may perform poorly if we don't have much data, but can perform almost as well as the optimal basis as the size of the dataset grows. Using the same data set as in the previous question, in this question you will explore using Gaussian radial basis functions (RBFs), which have this property. These RBFs depend on a hyperparameter σ, which (like p in the polynomial basis) can be chosen using a validation set. In this question, you will also see how cross-validation allows you to tune parameters of the model on a larger dataset than a strict training/validation split would allow.

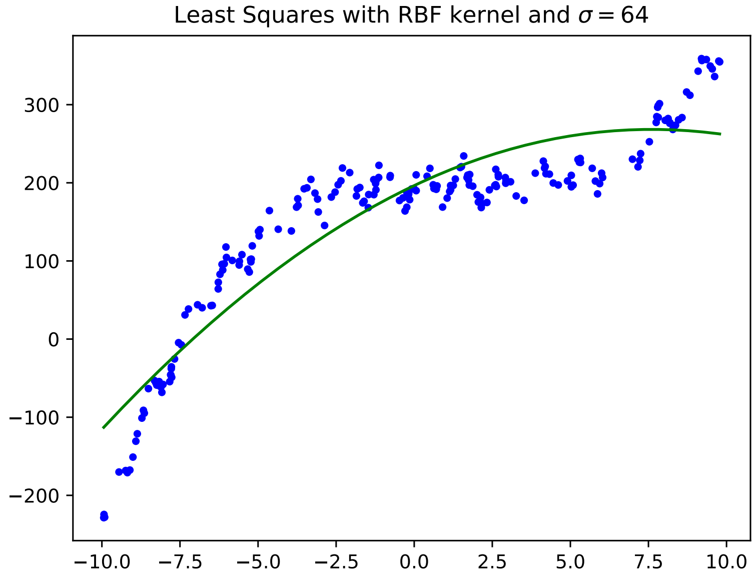
4.1 Proper Training and Validation Sets

Rubric: {reasoning:3}

If you run `python main.py -q 4`, it will load the dataset and split the training examples into "train" and "validation" sets. It will then search for the best value of σ for the RBF basis.[3] Once it has the "best" value of σ, it re-trains on the entire dataset and reports the training error on the full training set as well as the error on the test set.

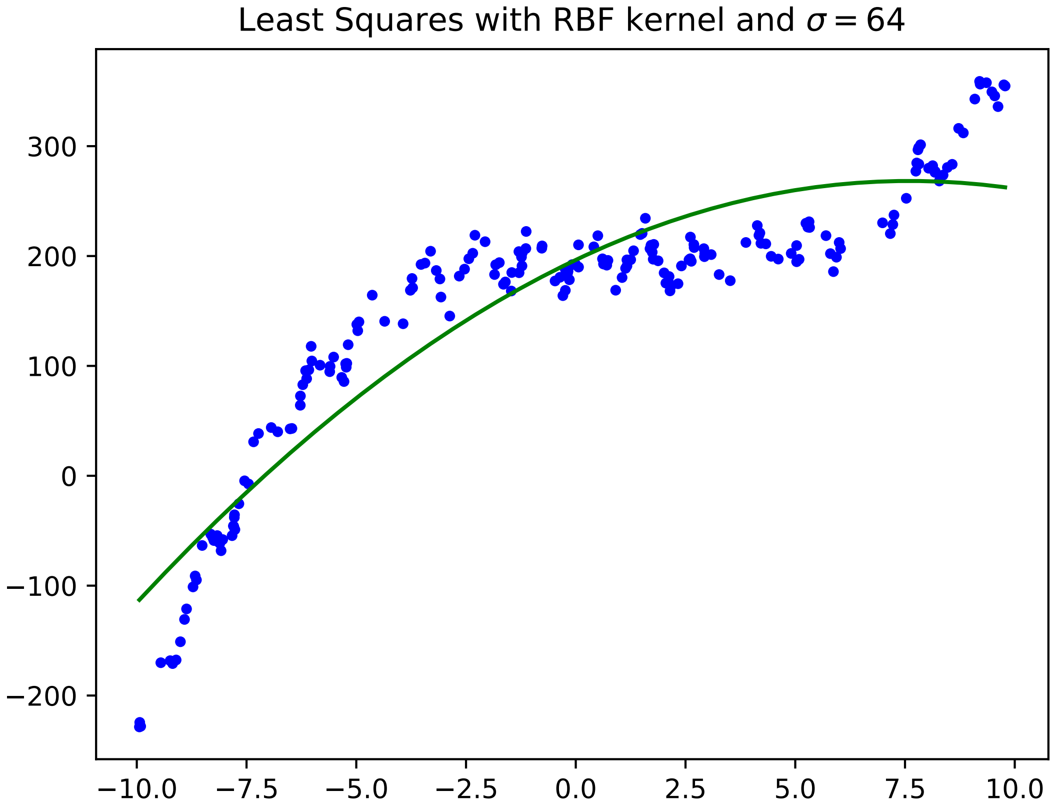
Unfortunately, there is a problem with the way this is done: the data isn't shuffled before being split. As a result, the IID assumption is badly broken and we end up with poor test error. Here is the plot:

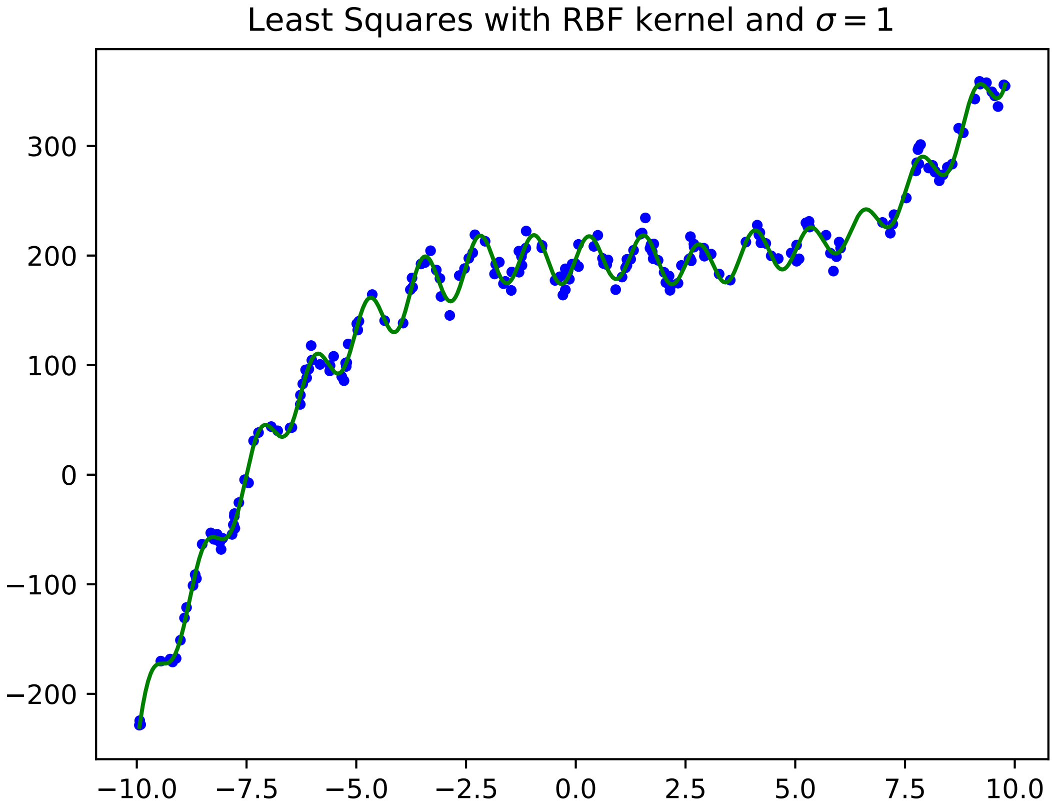
[3] if you look at the code you'll see that it also uses a tiny bit of regularization since Z^TZ tends to be very close to singular; more on this later in the course.



Fix the problem by either randomizing the split yourself or using `sklearn.model_selection.train_test_split` with `train_size=0.5`. Compare the train/test errors and plot before vs. after fixing this problem.

Answer: The URL of the code is `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a3/blob/master/code/main.py`. Before randomizing the split, the chosen sigma is 16, the training error is 2184.1 and the test error is 2495.9. After randomizing, the chosen sigma is 1, the training error is 39.5 and the test error is 71.2. The figures are shown below.


Least Squares with RBF kernel and $\sigma = 64$
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



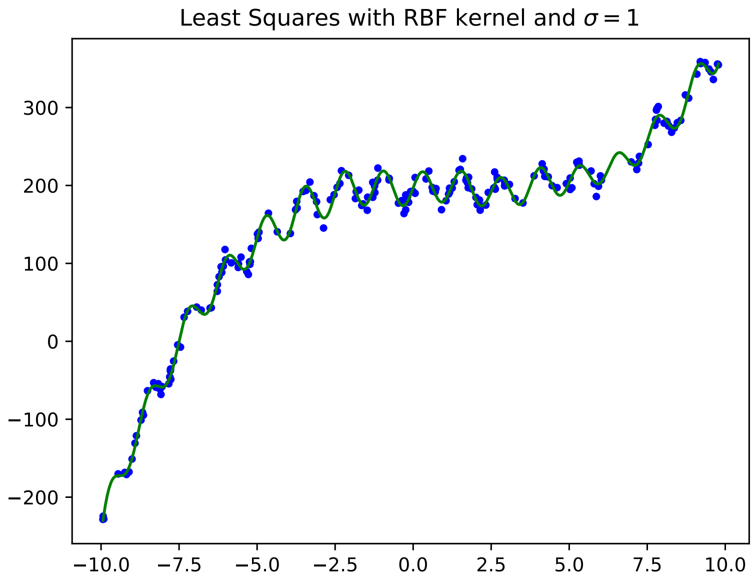
4.2 Cross-Validation

Rubric: {code:3,reasoning:1}

Now that we've dealt with the randomization, something's still a bit disturbing: if you run the script more than once it might choose different values of σ. This variability would be reduced if we had a larger "train" and "validation" set, and one way to simulate this is with *cross-validation*.

1. What are two different "best" values of σ you've observed after re-running the code a few times? (Not all students will have the same answer here; that's OK.)
   Answer: For most times, the sigma is chosen as 1. But sometimes it may change. For example, sigma=4 appears sometime with the training error as 234.1 and test error as 256.8.

2. Implement 10-fold cross-validation to select σ, and hand in your code. What value of σ does this procedure typically select?
   Answer: With 10-fold cross-validation, the chosen sigma is always 1. The figure is shown below.



4.3 Cost of Non-Parametric Bases

Rubric: {reasoning:3}

When dealing with larger datasets, an important issue is the dependence of the computational cost on the number of training examples n and the number of features d. What is the cost in big-O notation of training the model on n training examples with d features under (a) the linear basis and (b) Gaussian RBFs (for a fixed σ)? What is the cost of classifying t new examples under each of these two bases? When are RBFs cheaper to train? When are RBFs cheaper to test?
Answer: (a) Under linear basis the cost contains forming X^TX and solve system and is $O(nd^2) + O(d^3) = O(nd^2 + d^3)$. The classifying new examples needs to let each new example predict with w, this takes $O(td)$; (b)Under Gaussian RBFs, the cost contains forming Z^TZ and solve system and is $O(dn^2) + O(n^3) = O(dn^2 + n^3)$.
The classifying new examples should let each new example x_t compute $g(\|x_i - x_t\|)$ to predict and the cost is $O(tdn)$.
For training, we should compare $O(nd^2 + d^3)$ and $O(dn^2 + n^3)$. If $d < n$, linear basis is cheaper, otherwise, RBF is cheaper.
For testing, we should compare $O(td)$ and $O(tdn)$. If $n = 1$, they are the same, otherwise, linear basis is cheaper.

5 Very-Short Answer Questions

5.1 Essentials

Rubric: {reasoning:10}

1. In regression, why do we compute the squared error $(y_i - \hat{y}_i)^2$ rather than testing the equality $(y_i = \hat{y}_i)$?

Answer: Because y_i and $\hat{y}_i$ could only be close but not exactly match.

2. Describe a situation in which the least squares estimate would not be unique when $d = 2$ and $n = 4$.
   Answer: If features are not linearly independent, this could happen. For example, when $X = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \\ 4 & 8 \end{bmatrix}$, the estimate would not be unique.

3. What is the computational complexity of computing the closed-form (exact) solution to a linear least squares problem where we have one feature ($d = 1$) and use polynomial basis of degree p?
   Answer: By transform X to Z with p, Z is $n * (p + 1)$ matrix. The complexity is $O(np^2 + p^3)$ which is the similar form as the one with X.

4. In what circumstance would a regression tree with linear regressions at the leaves be a better choice than a linear least squares regression model?
   Answer: When the data is continuous, it is better to use regression tree and when the data is discrete, linear LSR is better.

5. When fitting a model, why do we care if our loss function is convex?
   Answer: When the loss function is convex, we can get global minimal solution. When it is not, we may possibly get local ones. If it is convex, gradient descent can find the global minimal solution.

6. When should we consider using gradient descent to approximate the solution to the least squares problem instead of exactly solving it with the closed form solution?
   Answer: The cost with closed form solution is $O(nd^2 + d^3)$, but using gradient decent, the cost is only $O(ndt)$ which is smaller when d is pretty large.

7. Why is optimization non-trivial? Can't we just set the gradient to zero and be done immediately?
   Answer: It is difficult to set the gradient to zero and solve the function. Because the function can be non-differential in some point or no solution for gradient to zero. So we need to the smoothing approximation for help.

8. Why do we need gradient descent for the robust regression problem, as opposed to just using the normal equations? Hint: it is NOT because of the non-differentiability. Recall that we used gradient descent even after smoothing away the non-differentiable part of the loss.
   Answer: For robust regression, it is often hard to get the norm equation when computing the ∇f. But gradient descent can be applied in all kinds of cases.

9. What is the problem with having too small of a learning rate in gradient descent?
   Answer: It may converge too slow.

10. What is the problem with having too large of a learning rate in gradient descent?
    Answer: If the learning rate is too large, it may result in overfitting. The cost function increase after some iteration and it may never converge to the optimal.

5.2 These ones are optional and not for marks

1. In `LinearModelGradient` there's code that checks your gradient using `scipy.optimize.approx_fprime`. But, wait a minute: if we can check the gradient, that means we already have it. So, why do we even bother taking the gradient by hand?
   Answer: `scipy.optimize.approx_fprime` could not get the exact gradient but the finite-difference approximation of the gradient, thus we have to calculate by hand.

2. What would go wrong if we tried to apply gradient descent to the un-smoothed absolute value loss?
   Answer: As it is un-smoothed, there may not existing $\nabla f = 0$ and it may never converge.