Homework 2: Lasso Regression

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though scanning handwritten work is fine as well. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better.

1 Introduction

In this homework you will investigate regression with ℓ_1 regularization, both implementation techniques and theoretical properties. On the methods side, you'll work on coordinate descent (the "shooting algorithm"), homotopy methods, and [optionally] projected SGD. On the theory side you'll derive the largest ℓ_1 regularization parameter you'll ever need to try, and optionally you'll derive the explicit solution to the coordinate minimizers used in coordinate descent, you'll investigate what happens with ridge and lasso regression when you have two copies of the same feature, and you'll work out the details of the classic picture that "explains" why ℓ_1 regularization leads to sparsity.

1.1 Data Set and Programming Problem Overview

For the experiments, we are generating some artifical data using code in the file `setup_problem.py`. We are considering the regression setting with the 1-dimensional input space $\mathbf{R}$. An image of the training data, along with the target function (i.e. the Bayes prediction function for the square loss function) is shown in Figure 1 below.

You can examine how the target function and the data were generated by looking at `setup_problem.py`. The figure can be reproduced by running the `LOAD_PROBLEM` branch of the main function.

As you can see, the target function is a highly nonlinear function of the input. To handle this sort of problem with linear hypothesis spaces, we will need to create a set of features that perform nonlinear transforms of the input. A detailed description of the technique we will use can be found in the Jupyter notebook `basis-fns.ipynb`, included in the zip file.

In this assignment, we are providing you with a function that takes care of the featurization. This is the "featurize" function, returned by the `generate_problem` function in `setup_problem.py`. The `generate_problem` function also gives the true target function, which has been constructed to be a sparse linear combination of our features. The coefficients of this linear combination are also provided by `generate_problem`, so you can compare the coefficients of the linear functions you find to the target function coefficients. The `generate_problem` function also gives you the train and validation sets that you should use.

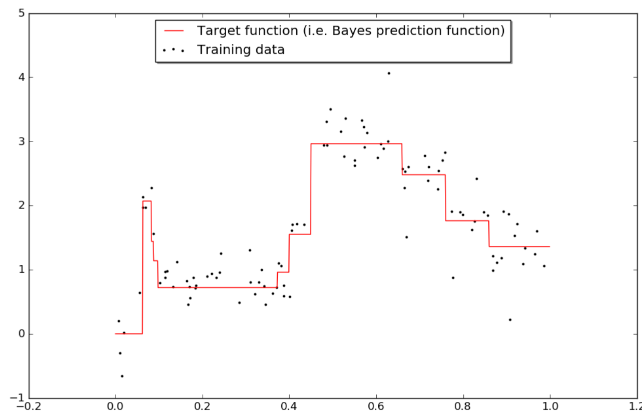

Figure 1: Training data and target function we will be considering in this assignment.

To get familiar with using the data, and perhaps to learn some techniques, it's recommended that you work through the main() function of the include file ridge_regression.py. You'll go through the following steps (on your own - no need to submit):

1. Load the problem from disk into memory with load_problem.
2. Use the featurize function to map from a one-dimensional input space to a d-dimensional feature space.
3. Visualize the design matrix of the featurized data. (All entries are binary, so we will not do any data normalization or standardization in this problem, though you may experiment with that on your own.)
4. Take a look at the class `RidgeRegression`. Here we've implemented our own `RidgeRegression` using the general purpose optimizer provided by scipy.optimize. This is primarily to introduce you to the sklearn framework, if you are not already familiar with it. It can help with hyperparameter tuning, as we will see shortly.
5. Take a look at compare_our_ridge_with_sklearn. In this function, we want to get some evidence that our implementation is correct, so we compare to sklearn's ridge regression. Comparing the outputs of two implementations is not always trivial – often the objective functions are slightly different, so you may need to think a bit about how to compare the results. In this case, sklearn has total square loss rather than average square loss, so we needed to account for that. In this case, we get an almost exact match with sklearn. This is because ridge regression is a rather easy objective function to optimize. You may not get as exact a match for other objective functions, even if both methods are "correct."
6. Next take a look at do_grid_search, in which we demonstrate how to take advantage of the fact that we've wrapped our ridge regression in an sklearn "Estimator" to do hyperparameter tuning. It's a little tricky to get GridSearchCV to use the train/test split that you want, but

an approach is demonstrated in this function. In the line assigning the param_grid variable, you can see my attempts at doing hyperparameter search on a different problem. Below you will be modifying this (or using some other method, if you prefer) to find the optimal L2 regularization parameter for the data provided.

7. Next is some code to plot the results of the hyperparameter search.

8. Next we want to visualize some prediction functions. We plotted the target function, along with several prediction functions corresponding to different regularization parameters, as functions of the original input space $\mathbf{R}$, along with the training data. Next we visualize the coefficients of each feature with bar charts. Take note of the scale of the y-axis, as they may vary substantially, buy default.

2 Ridge Regression

In the problems below, you do not need to implement ridge regression. You may use any of the code provided in the assignment, or you may use other packages. However, your results must correspond to the ridge regression objective function that we use, namely

$$J(w; \lambda) = \frac{1}{n} \sum_{i=1}^{n} \left(w^T x_i - y_i\right)^2 + \lambda \|w\|^2.$$

1. Run ridge regression on the provided training dataset. Choose the λ that minimizes the empirical risk (i.e. the average square loss) on the validation set. Include a table of the parameter values you tried and the validation performance for each. Also include a plot of the results.

2. Now we want to visualize the prediction functions. On the same axes, plot the following: the training data, the target function, an unregularized least squares fit (still using the featurized data), and the prediction function chosen in the previous problem. Next, along the lines of the bar charts produced by the code in compare_parameter_vectors, visualize the coefficients for each of the prediction functions plotted, including the target function. Describe the patterns, including the scale of the coefficients, as well as which coefficients have the most weight.

3. For the chosen λ, examine the model coefficients. For ridge regression, we don't expect any parameters to be exactly 0. However, let's investigate whether we can predict the sparsity pattern of the true parameters (i.e. which parameters are 0 and which are nonzero) by thresholding the parameter estimates we get from ridge regression. We'll predict that $w_i = 0$ if $|\hat{w}_i| < \varepsilon$ and $w_i \neq 0$ otherwise. Give the confusion matrix for $\varepsilon = 10^{-6}, 10^{-3}, 10^{-1}$, and any other thresholds you would like to try.

3 Coordinate Descent for Lasso (a.k.a. The Shooting algorithm)

The Lasso optimization problem can be formulated as[1]

$$\hat{w} \in \underset{w \in \mathbf{R}^d}{\arg\min} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2 + \lambda \|w\|_1,$$

where $h_w(x) = w^T x$, and $\|w\|_1 = \sum_{i=1}^{d} |w_i|$. Note that to align with Murpy's formulation below, and for historical reasons, we are using the total square loss, rather than the average square loss, in the objective function.

Since the ℓ_1-regularization term in the objective function is non-differentiable, it's not immediately clear how gradient descent or SGD could be used to solve this optimization problem directly. (In fact, as we'll see in the next homework on SVMs, we can use "subgradient" methods when the objective function is not differentiable, in addition to the two methods discussed in this homework assignment.)

Another approach to solving optimization problems is coordinate descent, in which at each step we optimize over one component of the unknown parameter vector, fixing all other components. The descent path so obtained is a sequence of steps, each of which is parallel to a coordinate axis in $\mathbf{R}^d$, hence the name. It turns out that for the Lasso optimization problem, we can find a closed form solution for optimization over a single component fixing all other components. This gives us the following algorithm, known as the **shooting algorithm**:

Algorithm 13.1: Coordinate descent for lasso (aka shooting algorithm)

Initialize $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$;
repeat
  for $j = 1, \ldots, D$ **do**
    $a_j = 2\sum_{i=1}^{n} x_{ij}^2$;
    $c_j = 2\sum_{i=1}^{n} x_{ij}(y_i - \mathbf{w}^T\mathbf{x}_i + w_j x_{ij})$;
    $w_j = \text{soft}(\frac{c_j}{a_j}, \frac{\lambda}{a_j})$;
until *converged*;

(Source: Murphy, Kevin P. Machine learning: a probabilistic perspective. MIT press, 2012.)

The "soft thresholding" function is defined as

$$\text{soft}(a, \delta) = \text{sign}(a) \max(|a| - \delta, 0),$$

for any $a, \delta \in \mathbf{R}$. Note also that x_i is the i'th row of X, but represented as a column vector.

NOTE: Algorithm 13.1 does not account for the case that $a_j = c_j = 0$, which occurs when the jth column of X is identically 0. One can either eliminate the column (as it cannot possibly help the solution), or you can set $w_j = 0$ in that case since it is, as you can easily verify, the coordinate minimizer. Note also that Murphy is suggesting to initialize the optimization with the

[1]

ridge regession solution. Although theoretically this is not necessary (with exact computations and enough time, coordinate descent will converge for lasso from any starting point), in practice it's helpful to start as close to the solution as we're able.

There are a few tricks that can make selecting the hyperparameter λ easier and faster. First, as we'll see in a later problem, you can show that for any $\lambda \geq 2\|X^T(y - \bar{y})\|_\infty$, the estimated weight vector $\hat{w}$ is entirely zero, where $\bar{y}$ is the mean of values in the vector y, and $\|\cdot\|_\infty$ is the infinity norm (or supremum norm), which is the maximum over the absolute values of the components of a vector. Thus we need to search for an optimal λ in $[0, \lambda_{\max}]$, where $\lambda_{\max} = 2\|X^T(y-\bar{y})\|_\infty$. (Note: This expression for $\lambda_{\max}$ assumes we have an unregularized bias term in our model. That is, our decision functions are of the form $h_{w,b}(x) = w^T x + b$. In our the experiments, we do not have an unregularized bias term, so we should use $\lambda_{\max} = 2\|X^T y\|_\infty$.)

The second trick is to use the fact that when λ and λ' are close, the corresponding solutions $\hat{w}(\lambda)$ and $\hat{w}(\lambda')$ are also close. Start with $\lambda = \lambda_{\max}$, for which we know $\hat{w}(\lambda_{\max}) = 0$. You can run the optimization anyway, and initialize the optimization at $w = 0$. Next, λ is reduced (e.g. by a constant factor close to 1), and the optimization problem is solved using the previous optimal point as the starting point. This is called **warm starting** the optimization. The technique of computing a set of solutions for a chain of nearby λ's is called a **continuation** or **homotopy method**. The resulting set of parameter values $\hat{w}(\lambda)$ as λ ranges over $[0, \lambda_{\max}]$ is known as a **regularization path**.

3.1 Experiments with the Shooting Algorithm

1. The algorithm as described above is not ready for a large dataset (at least if it has being implemented in Python) because of the implied loop in the summation signs for the expressions for a_j and c_j. Give an expression for computing a_j and c_j using matrix and vector operations, without explicit loops. This is called "vectorization" and can lead to dramatic speedup when implemented in languages such as Python, Matlab, and R. Write your expressions using X, w, $y = (y_1, \ldots, y_n)^T$ (the column vector of responses), $X_{\cdot j}$ (the jth column of X, represented as a column matrix), and w_j (the jth coordinate of w – a scalar).

   Solution:

$$
\begin{aligned}
a_j &= 2\sum_{i=1}^{n} x_{i,j}^2 = 2X_{\cdot j}^T X_{\cdot j} \\
c_j &= 2\sum_{i=1}^{n} x_{i,j}(y_i - w^T x_i + w_j x_{ij}) \\
&= 2X_{\cdot j}^T (y - Xw + w_j X_{\cdot j})
\end{aligned}
$$

2. Write a function that computes the Lasso solution for a given λ using the shooting algorithm described above. For convergence criteria, continue coordinate descent until a pass through the coordinates reduces the objective function by less than 10^{-8}, or you have taken 1000 passes through the coordinates. Compare performance of cyclic coordinate descent to randomized coordinate descent, where in each round we pass through the

coordinates in a different random order (for your choices of λ). Compare also the solutions attained (following the convergence criteria above) for starting at 0 versus starting at the ridge regression solution suggested by Murphy (again, for your choices of λ). If you like, you may adjust the convergence criteria to try to attain better results (or the same results faster).Solution:

```
def shooting_algorithm(X, y, w0=None, l1_reg = 1., max_num_epochs = 1000,
    min_obj_decrease=1e-8, random=False):
    if w0 is None:
        w = np.zeros(X.shape[1])
    else:
        w = np.copy(w0)
    d = X.shape[1]
    epoch = 0
    obj_val = compute_lasso_objective(X, y, w, l1_reg)
    obj_decrease = min_obj_decrease + 1.
    while (obj_decrease>min_obj_decrease) and (epoch<max_num_epochs):
        obj_old = obj_val
        # Cyclic coordinates descent
        coordinates = range(d)
        # Randomized coordinates descent
        if random:
            coordinates = np.random.permutation(d)
        for j in coordinates:
            X_j = X[:, j]  # Extract the j'th column of X
            a_j = 2 * np.dot(X_j, X_j)
            if np.isclose(a_j, 0):
                w[j] = 0 # if a_j = 0, then w_j=0
                continue
            partial_residual = y - np.dot(X, w) + w[j]*X_j
            c_j = 2 * np.dot(X_j.T, partial_residual)
            w[j] = soft_threshold(c_j/a_j, l1_reg/a_j)
        obj_val = compute_lasso_objective(X, y, w, l1_reg)
        obj_decrease = obj_old - obj_val
        epoch += 1
    print("Ran for "+str(epoch)+" epochs.")
    return w
```

3. Run your best Lasso configuration on the training dataset provided, and select the λ that minimizes the average square loss on the validation set. Include a table of the parameter values you tried and the validation performance for each. Also include a plot of these results. Include also a plot of the prediction functions, just as in the ridge regression section, but this time add the best performing Lasso prediction function and remove the unregularized least squares fit. Similarly, add the lasso coefficients to the bar charts of coefficients generated in the ridge regression setting. Comment on the results, with particular attention to parameter sparsity and how the ridge and lasso solutions compare. What's the best model you found, and what's its validation performance?

Solution:

Figure showing Validation Loss versus λ:

$\lambda = 0.08$ would be the optimal value, and the corresponding validation loss is 0.8619681632728108.

4. Implement the homotopy method described above. Compute the Lasso solution for (at least) the regularization parameters in the set $\{\lambda = \lambda_{\max} 0.8^i \mid i = 0, \ldots, 29\}$. Plot the results (average validation loss vs λ).Solutions:

```
def do_grid_search_homotopy(X_train, y_train, X_val, y_val,
                            reg_vals=None, w0=None):
    if reg_vals is None:
        lambda_max = get_lambda_max_no_bias(X_train, y_train)
        reg_vals = [lambda_max * (.8**n) for n in range(0, 30)]
        if w0 is None:
            w0 = np.zeros(X_train.shape[1])
    elif w0 is None:
        w0 = get_ridge_solution(X_train, y_train, reg_vals[0])

    estimator = LassoRegression()
    lasso_reg_path_estimator = LassoRegularizationPath(estimator, tune_param_name="
    l1_reg")
    lasso_reg_path_estimator.fit(X_train, y_train,
                                 reg_vals=reg_vals[:], coef_init=w0,
                                 warm_start=True)
    return lasso_reg_path_estimator
```

5. [Optional] Note that the data in Figure 1 is almost entirely nonnegative. Since we don't have an unregularized bias term, we have to "pay for" this offset using our penalized parameters. Note also that $\lambda_{\max}$ would decrease significantly if the y values were 0 centered (using the training data, of course), or if we included an unregularized bias term. Experiment with one or both of these approaches, for both and lasso and ridge regression, and report your findings. **Solution:**

   Centering the data will dramatically decrease $\lambda_{\max}$ and speed up regularization path.

3.2 [Optional] Deriving the Coordinate Minimizer for Lasso

This problem is to derive the expressions for the coordinate minimizers used in the Shooting algorithm. This is often derived using subgradients (slide 15), but here we will take a bare hands approach (which is essentially equivalent).

In each step of the shooting algorithm, we would like to find the w_j minimizing

$$\begin{aligned} f(w_j) &= \sum_{i=1}^{n} \left(w^T x_i - y_i\right)^2 + \lambda |w|_1 \\ &= \sum_{i=1}^{n} \left[w_j x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda |w_j| + \lambda \sum_{k \neq j} |w_k|, \end{aligned}$$

where we've written x_{ij} for the jth entry of the vector x_i. This function is convex in w_j. The only thing keeping f from being differentiable is the term with $|w_j|$. So f is differentiable everywhere except $w_j = 0$. We'll break this problem into 3 cases: $w_j > 0$, $w_j < 0$, and $w_j = 0$. In the first two cases, we can simply differentiate f w.r.t. w_j to get optimality conditions. For the last case, we'll use the fact that since $f : \mathbf{R} \to \mathbf{R}$ is convex, 0 is a minimizer of f iff

$$\lim_{\varepsilon \downarrow 0} \frac{f(\varepsilon) - f(0)}{\varepsilon} \geq 0 \quad \text{and} \quad \lim_{\varepsilon \downarrow 0} \frac{f(-\varepsilon) - f(0)}{\varepsilon} \geq 0.$$

This is a special case of the optimality conditions described in slide 6 here, where now the "direction" v is simply taken to be the scalars 1 and -1, respectively.

1. First let's get a trivial case out of the way. If $x_{ij} = 0$ for $i = 1, \ldots, n$, what is the coordinate minimizer w_j? In the remaining questions below, you may assume that $\sum_{i=1}^{n} x_{ij}^2 > 0$.
Solution:

Since $x_{ij} = 0$ for $i = 1, \ldots, n$**,** we can rewrite $f(w_j)$ as:

$$\begin{aligned} f(w_j) &= \sum_{i=1}^{n} \left[w_j x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda \left| w_j \right| + \lambda \sum_{k \neq j} \left| w_k \right| \\ &= \sum_{i=1}^{n} \left[\sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda \left| w_j \right| + \lambda \sum_{k \neq j} \left| w_k \right| \end{aligned}$$

Only the term $\lambda \left| w_j \right|$depends on w_j. This is minimized when $w_j = 0$.

2. Give an expression for the derivative $f(w_j)$ for $w_j \neq 0$. It will be convenient to write your expression in terms of the following definitions:

$$\begin{aligned} \text{sign}(w_j) &:= \begin{cases} 1 & w_j > 0 \\ 0 & w_j = 0 \\ -1 & w_j < 0 \end{cases} \\ a_j &:= 2 \sum_{i=1}^{n} x_{ij}^2 \\ c_j &:= 2 \sum_{i=1}^{n} x_{ij} \left(y_i - \sum_{k \neq j} w_k x_{ik} \right). \end{aligned}$$

Solution:

$$\begin{aligned} f'(w_j) &= \sum_{i=1}^{n} \left[2x_{ij} \left(w_j x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i \right) \right] + \lambda \text{sign}\left(w_j \right) \\ &= 2w_j \sum_{i=1}^{n} x_{ij}^2 - 2 \sum_{i=1}^{n} x_{ij} \left(y_i - \sum_{k \neq j} w_k x_{ik} \right) + \lambda \text{sign}\left(w_j \right) \\ &= w_j a_j - c_j + \lambda \text{sign}\left(w_j \right) \end{aligned}$$

3. If $w_j > 0$ and minimizes f, show that $w_j = \frac{1}{a_j}\left(c_j - \lambda \right)$. Similarly, if $w_j < 0$ and minimizes f, show that $w_j = \frac{1}{a_j}\left(c_j + \lambda \right)$. Give conditions on c_j that imply that a minimizer w_j is positive

and conditions for which a minimizer w_j is negative.
Solution: Setting $f'(w_j) = 0$, we get

$$w_j = \frac{1}{a_j}\left[c_j - \lambda \text{sign}(w_j)\right].$$

Thus if $c_j > \lambda$, then $w_j = \frac{1}{a_j}\left[c_j - \lambda\right]$ satisfies this condition and is a minimizer (and is positive). If $c_j < -\lambda$ then $w_j = \frac{1}{a_j}\left[c_j + \lambda\right]$ is a minimizer (and is negative).

4. Derive expressions for the two one-sided derivatives at $f(0)$, and show that $c_j \in [-\lambda, \lambda]$ implies that $w_j = 0$ is a minimizer.
Solution:
Since $f(w_j)$ is not differentiable at $w_j = 0$, we can examine directly what happens when we move from $w_j = 0$ to $w_j = \varepsilon$, for any ε:

$$\begin{aligned} f(\varepsilon) &= \sum_{i=1}^{n}\left[\varepsilon x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i\right]^2 + \lambda|\varepsilon| + \lambda\sum_{k\neq j}|w_k| \\ f(0) &= \sum_{i=1}^{n}\left[\sum_{k \neq j} w_k x_{ik} - y_i\right]^2 + \lambda\sum_{k\neq j}|w_k| \end{aligned}$$

Generally speaking,

$$(a+b)^2 - b^2 = a^2 + 2ab,$$

so

$$\begin{aligned} f(\varepsilon) - f(0) &= \sum_{i=1}^{n}\left[\varepsilon^2 x_{ij}^2 + 2\varepsilon x_{ij}\left(\sum_{k\neq j} w_k x_{ik} - y_i\right)\right] + \lambda|\varepsilon| \\ &= \frac{1}{2}\varepsilon^2 a_j - \varepsilon c_j + \lambda|\varepsilon| \end{aligned}$$

and

$$\frac{f(\varepsilon) - f(0)}{\varepsilon} = \frac{1}{2}\varepsilon a_j - c_j + \lambda \text{sign}(\varepsilon).$$

So the derivative to the right is

$$\lim_{\varepsilon \downarrow 0} \frac{f(\varepsilon) - f(0)}{\varepsilon} = -c_j + \lambda$$

and the derivative to the left is

$$\begin{aligned} \lim_{\varepsilon \downarrow 0} \frac{f(-\varepsilon) - f(0)}{\varepsilon} &= \lim_{\varepsilon \downarrow 0} \frac{1}{\varepsilon}\left(\frac{1}{2}\varepsilon^2 a_j + \varepsilon c_j + \lambda|-\varepsilon|\right) \\ &= \lim_{\varepsilon \downarrow 0} \frac{1}{\varepsilon}\left(\frac{1}{2}\varepsilon^2 a_j + \varepsilon c_j + \lambda\varepsilon\right) \\ &= \lim_{\varepsilon \downarrow 0} \frac{1}{2}\varepsilon a_j + c_j + \lambda \\ &= c_j + \lambda. \end{aligned}$$

$w_j = 0$ is a minimum iff both one-sided derivatives are nonnegative. The first limit is nonnegative iff $c_j \le \lambda$ and the second limit is nonnegative iff $c_j \ge -\lambda$. Thus $w_j = 0$ is a minimum iff $c_j \in [-\lambda, \lambda]$.

5. Putting together the preceding results, we conclude the following:

$$w_j = \begin{cases} \frac{1}{a_j}(c_j - \lambda) & c_j > \lambda \\ 0 & c_j \in [-\lambda, \lambda] \\ \frac{1}{a_j}(c_j + \lambda) & c_j < -\lambda \end{cases}$$

Show that this is equivalent to the expression given in 3.
Solution:

$$\begin{aligned} \textbf{soft}\left(\frac{c_j}{a_j}, \frac{\lambda}{a_j}\right) &= \textbf{sign}\left(\frac{c_j}{a_j}\right) \times \max\left(\left|\frac{c_j}{a_j}\right| - \frac{\lambda}{a_j}, 0\right) \\ &= \textbf{sign}(c_j) \times \frac{1}{a_j} \max(|c_j| - \lambda, 0) \\ &= \begin{cases} \frac{1}{a_j}(c_j - \lambda) & c_j > \lambda \\ 0 & c_j \in [-\lambda, \lambda] \\ \frac{1}{a_j}(c_j + \lambda) & c_j < -\lambda \end{cases} \end{aligned}$$

4 Lasso Properties

4.1 Deriving $\lambda_{\max}$

In this problem we will derive an expression for $\lambda_{\max}$. For the first three parts, use the Lasso objective function excluding the bias term i.e, $J(w) = \|Xw - y\|_2^2 + \lambda \|w\|_1$. We will show that for any $\lambda \ge 2\|X^T y\|_\infty$, the estimated weight vector $\hat{w}$ is entirely zero, where $\|\cdot\|_\infty$ is the infinity norm (or supremum norm), which is the maximum absolute value of any component of the vector.

1. The one-sided directional derivative of $f(x)$ at x in the direction v is defined as:

$$f'(x; v) = \lim_{h \downarrow 0} \frac{f(x + hv) - f(x)}{h}$$

Compute $J'(0; v)$. That is, compute the one-sided directional derivative of $J(w)$ at $w = 0$ in the direction v. [Hint: the result should be in terms of $X, y, \lambda,$ and v.]
Solution:
For any $w \in \mathbf{R}^d$ we have

$$\begin{aligned} J(w) =& (Xw - y)^T (Xw - y) + \lambda \|w\|_1 \\ =& w^T X^T X w - 2 w^T X^T y + y^T y + \lambda \|w\|_1 \end{aligned}$$

So for any $h > 0$ and $v \in \mathbf{R}^d$ we have

$$\begin{aligned} J(0) &= y^T y \\ J(0 + hv) &= h^2 v^T X^T X v - 2hv^T X^T y + y^T y + \lambda \|hv\|_1 \\ \frac{J(0 + hv) - J(0)}{h} &= hv^T X^T X v - 2v^T X^T y + \lambda \|v\|_1 \end{aligned}$$

Thus the one-sided directional derivative of J at 0 is

$$\begin{aligned} J'(0,v) &= \lim_{h\downarrow 0}\frac{J(0+hv)-J(0)}{h} \\ &= -2v^TX^Ty+\lambda\|v\|_1 \end{aligned}$$

2. Since the Lasso objective is convex, w^* is a minimizer of $J(w)$ if and only if the directional derivative $J'(w^*;v)\geq 0$ for all $v\neq 0$. Show that for any $v\neq 0$, we have $J'(0;v)\geq 0$ if and only if $\lambda\geq C$, for some C that depends on X, y, and v. You should have an explicit expression for C.

   Solution:

$$\begin{aligned} J'(0,v) &\geq 0 \\ \iff -2v^TX^Ty+\lambda\|v\|_1 &\geq 0 \\ \iff \lambda &\geq \frac{2v^TX^Ty}{\|v\|_1} \end{aligned}$$

3. In the previous problem, we get a different lower bound on λ for each choice of v. Show that the maximum of these lower bounds on λ is $\lambda_{\max}=2\|X^Ty\|_\infty$. Conclude that $w=0$ is a minimizer of $J(w)$ if and only if $\lambda\geq 2\|X^Ty\|_\infty$.
   Solution:

   Claim: $\|y\|_\infty=\max_{x:\|x\|_1=1}x^Ty$.
   In words, this is true because we can put all the weight in x on the component corresponding to the entry of y with the largest absolute value.
   Proof. $x^Ty=\sum_i x_iy_i\leq\|y\|_\infty\sum_i x_i\leq\|y\|_\infty\sum_i|x_i|\leq\|y\|_\infty$. Let $i^*=\arg\max_i|y_i|$, and let $x_{i^*}=\mathrm{sign}(y_{i^*})$ and $x_j=0$ for $j\neq i$. Then $\|x\|_1=1$, and $x^Ty=y_{i^*}=\|y\|_\infty$. **QED**

   Applying the claim, note that

$$\begin{aligned} \lambda &\geq \frac{2v^TX^Ty}{\|v\|_1}\text{ for all }v\neq 0 \\ \iff \lambda &\geq \sup_{v\neq 0}\frac{2v^TX^Ty}{\|v\|_1} \\ &= 2\max_{v:\|v\|_1=1}v^T\left(X^Ty\right) \\ &= 2\|X^Ty\|_\infty \end{aligned}$$

   Putting this together with the previous problem gives our conclusion.

4. [Optional] Let $J(w,b)=\|Xw+b\mathbf{1}-y\|_2^2+\lambda\|w\|_1$, where $\mathbf{1}\in\mathbf{R}^n$ is a column vector of 1's. Let $\bar{y}$ be the mean of values in the vector y. Show that $(w^*,b^*)=(0,\bar{y})$ is a minimizer of $J(w,b)$ if and only if $\lambda\geq\lambda_{\max}=2\|X^T(y-\bar{y})\|_\infty$.

Solution:

For any $w \in \mathbf{R}^d, b \in \mathbf{R}$ we have

$$\begin{aligned} J(w,b) =& (Xw + b\mathbf{1} - y)^T (Xw + b\mathbf{1} - y) + \lambda \|w\|_1 \\ =& w^T X^T X w - 2w^T X^T (b\mathbf{1} - y) + (b\mathbf{1} - y)^T (b\mathbf{1} - y) + \lambda \|w\|_1 \end{aligned}$$

So for any $h > 0$, $v_w \in \mathbf{R}^d$ and $v_b \in \mathbf{R}$, we have

$$\begin{aligned} J(0,\bar{y}) &= (\bar{y}\mathbf{1} - y)^T (\bar{y}\mathbf{1} - y) \\ J(0 + hv_w, \bar{y} + hv_b) &= h^2 v_w^T X^T X v_w - 2hv_w^T X^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) \\ &\quad + (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y)^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) + \lambda h \|v_w\|_1 \\ &= h^2 v_w^T X^T X v_w - 2hv_w^T X^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) + h^2 v_b^2 \mathbf{1}^T \mathbf{1} \\ &\quad + 2hv_b \mathbf{1}^T (\bar{y}\mathbf{1} - y) + (\bar{y}\mathbf{1} - y)^T (\bar{y}\mathbf{1} - y) + \lambda h \|v_w\|_1 \\ \frac{J(0 + hv_w, \bar{y} + hv_b) - J(0,\bar{y})}{h} &= hv_w^T X^T X v_w - 2v_w^T X^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) + hv_b^2 \mathbf{1}^T \mathbf{1} \\ &\quad + 2v_b \mathbf{1}^T (\bar{y}\mathbf{1} - y) + \lambda \|v_w\|_1 \end{aligned}$$

Thus the one-sided directional derivative of J at $0, \bar{y}$ is

$$\begin{aligned} J'(0,\bar{y}; v_w, v_h) &= \lim_{h \downarrow 0} \frac{J(0 + hv_w, \bar{y} + hv_b) - J(0,\bar{y})}{h} \\ &= -2v_w^T X^T (\bar{y}\mathbf{1} - y) + 2v_b \underbrace{\mathbf{1}^T (\bar{y}\mathbf{1} - y)}_{=0} + \lambda \|v_w\|_1 \\ &= -2v_w^T X^T (\bar{y}\mathbf{1} - y) + \lambda \|v_w\|_1 \end{aligned}$$

So $(0, \bar{y})$ is a minimum iff

$$\begin{aligned} J'(0,\bar{y}; v_w, v_h) &\geq 0 \text{ for all } (v_w, v_h) \in \mathbf{R}^{d+1} \\ \iff -2v_w^T X^T (\bar{y}\mathbf{1} - y) + \lambda \|v_w\|_1 &\geq 0 \text{ for all } v_w \neq 0 \text{ (always true for } v_w = 0) \\ \iff \lambda &\geq \frac{2v_w^T X^T (\bar{y}\mathbf{1} - y)}{\|v_w\|_1} \text{ for all } v_w \neq 0 \\ \iff \lambda &\geq \sup_{v_w \in \mathbf{R}^d} \frac{2v_w^T X^T (\bar{y}\mathbf{1} - y)}{\|v_w\|_1} \\ &= 2 \max_{v_w : \|v_w\|_1 = 1} v_w^T X^T (\bar{y}\mathbf{1} - y) \\ &= 2\|X^T (\bar{y}\mathbf{1} - y)\|_\infty, \end{aligned}$$

where the last step is from the Lemma in the previous problem.

(From Peter Li) Alternative Solution for 3.1 Using Subgradients:

Let $g(\theta)$ be a subgradient of the lasso objective function (S_θ is the subgradient of $\|\cdot\|_1$at θ):

$$g(\theta) = 2X^T X\theta - 2X^T y + \lambda S_\theta$$

Then θ is a minimizer of our loss iff 0 is a subgradient at $\theta = 0$. This means $g(0) = 0$ must be possible. $g(0) = 0$ if and only if $S_0 = \frac{2X^T y}{\lambda}$. S_0(the subgradient of the l_1 norm at 0) is a vector where each entry is in $[-1, 1]$[2]. For this to hold, $\lambda \geq \left\|2X^T y\right\|_\infty$.

4.2 Feature Correlation

In this problem, we will examine and compare the behavior of the Lasso and ridge regression in the case of an exactly repeated feature. That is, consider the design matrix $X \in \mathbf{R}^{m \times d}$, where $X_{\cdot i} = X_{\cdot j}$ for some i and j, where $X_{\cdot i}$ is the i^{th} column of X. We will see that ridge regression divides the weight equally among identical features, while Lasso divides the weight arbitrarily. In an optional part to this problem, we will consider what changes when $X_{\cdot i}$ and $X_{\cdot j}$ are highly correlated (e.g. exactly the same except for some small random noise) rather than exactly the same.

1. Without loss of generality, assume the first two colums of X are our repeated features. Partition X and θ as follows:

$$X = \begin{pmatrix} x_1 & x_2 & X_r \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_r \end{pmatrix}$$

We can write the Lasso objective function as:

$$\begin{aligned} J(\theta) &= \|X\theta - y\|_2^2 + \lambda \|\theta\|_1 \\ &= \|x_1\theta_1 + x_2\theta_2 + X_r\theta_r - y\|_2^2 + \lambda|\theta_1| + \lambda|\theta_2| + \lambda \|\theta_r\|_1 \end{aligned}$$

With repeated features, there will be multiple minimizers of $J(\theta)$. Suppose that

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

is a minimizer of $J(\theta)$. Give conditions on c and d such that $\left(c, d, r^T\right)^T$ is also a minimizer of $J(\theta)$. [Hint: First show that a and b must have the same sign, or at least one of them is zero. Then, using this result, rewrite the optimization problem to derive a relation between a and b.]

Solution:

Suppose $a < 0 < b$. Then we could construct $\hat{\theta}'$ with smaller objective value. In particular, take

$$\hat{\theta}' = \begin{pmatrix} a + b \\ 0 \\ r \end{pmatrix}.$$

Then the ℓ_1 penalty term is smaller, since $|a + b| < |a| + |b|$ if a and b have different signs. Meanwhile, the loss piece stays the same. So $J(\hat{\theta}') < J(\hat{\theta})$, which contradicts that $\hat{\theta}$ minimizes $J(\theta)$.

[2]See proposition 3.5 in Carlos Fernandez-Granda's Convex Optimization notes.

Now we can rewrite the objective function as

$$J(\theta) = \|(x_1(\theta_1 + \theta_2) + X_r\theta_r - y\|_2^2 + \lambda|\theta_1 + \theta_2| + \lambda \|\theta_r\|_1 ,$$

from which we can see that we can take so long as $c + d = a + b$ and c and d have the same sign, then $\left(c, d, r^T\right)^T$ also minimizes the Lasso objective.

2. Using the same notation as the previous problem, suppose

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

minimizes the ridge regression objective function. What is the relationship between a and b, and why?
Solution:

Following the notation from above, we can write the ridge regression objective function as

$$\begin{aligned} J(\theta) &= \|X\theta - y\|_2^2 + \lambda \|\theta\|_2 \\ &= \|(\theta_1 + \theta_2)x_1 + X_r\theta_r - y\|_2^2 + \lambda(\theta_1^2 + \theta_2^2) + \lambda \|\theta_r\|_2^2 \end{aligned}$$

Note that the empirical risk piece of the objective function is invariant to changes to θ_1 and θ_2, so long as their sum remains the same. However, the regularization piece $\lambda(\theta_1^2 + \theta_2^2)$ will change. The regularization piece is minimized when $\theta_1 = \theta_2$, which can be shown with basic calculus.

3. [Optional] What do you think would happen with Lasso and ridge when $X_{\cdot i}$ and $X_{\cdot j}$ are highly correlated, but not exactly the same. You may investigate this experimentally or theoretically.

5 [Optional] The Ellipsoids in the ℓ_1/ℓ_2 regularization picture

Recall the famous picture purporting to explain why ℓ_1 regularization leads to sparsity, while ℓ_2 regularization does not. Here's the instance from Hastie et al's *The Elements of Statistical Learning:*

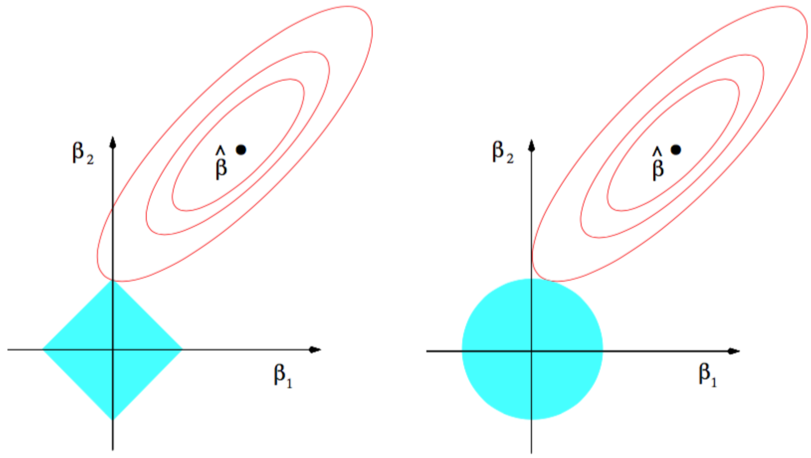

FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \le t$ and $\beta_1^2 + \beta_2^2 \le t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

(While Hastie et al. use β for the parameters, we'll continue to use w.)

In this problem we'll show that the level sets of the empirical risk are indeed ellipsoids centered at the empirical risk minimizer $\hat{w}$.

Consider linear prediction functions of the form $x \mapsto w^T x$. Then the empirical risk for $f(x) = w^T x$ under the square loss is

$$\begin{aligned}\hat{R}_n(w) &= \frac{1}{n}\sum_{i=1}^{n}\left(w^T x_i - y_i\right)^2 \\ &= \frac{1}{n}\left(Xw - y\right)^T\left(Xw - y\right).\end{aligned}$$

Assume that X has full rank, so that $X^T X$ is invertible.

1. [Optional] Let $\hat{w} = \left(X^T X\right)^{-1} X^T y$. Show that $\hat{w}$ has empirical risk given by

$$\hat{R}_n(\hat{w}) = \frac{1}{n}\left(-y^T X\hat{w} + y^T y\right)$$

Solution:

$$
\begin{aligned}
\frac{1}{n}(X\hat{w}-y)^{T}(X\hat{w}-y) &= \frac{1}{n}\left(X\left(X^{T}X\right)^{-1}X^{T}y-y\right)^{T}\left(X\left(X^{T}X\right)^{-1}X^{T}y-y\right) \\
&= \frac{1}{n}y^{T}X\left(X^{T}X\right)^{-1}X^{T}X\left(X^{T}X\right)^{-1}X^{T}y \\
&\quad -\frac{2}{n}y^{T}X\left(X^{T}X\right)^{-1}X^{T}y+\frac{1}{n}y^{T}y \\
&= -\frac{1}{n}y^{T}X\left(X^{T}X\right)^{-1}X^{T}y+\frac{1}{n}y^{T}y \\
&= -\frac{1}{n}y^{T}X\hat{w}+\frac{1}{n}y^{T}y
\end{aligned}
$$

2. [Optional] Show that for any w we have

$$
\hat{R}_n(w) = \frac{1}{n}\left(w-\hat{w}\right)^{T}X^{T}X\left(w-\hat{w}\right)+\hat{R}_n(\hat{w}).
$$

Note that the RHS (i.e. “right hand side”) has one term that’s quadratic in w and one term that’s independent of w. In particular, the RHS does not have any term that’s linear in w. On the LHS (i.e. “left hand side”), we have $\hat{R}_n(w) = \frac{1}{n}\left(Xw-y\right)^{T}\left(Xw-y\right)$. After expanding this out, you’ll have terms that are quadratic, linear, and constant in w. Completing the square is the tool for rearranging an expression to get rid of the linear terms. The following “completing the square” identity is easy to verify just by multiplying out the expressions on the RHS:

$$
x^{T}Mx-2b^{T}x = \left(x-M^{-1}b\right)^{T}M(x-M^{-1}b)-b^{T}M^{-1}b
$$

Solutions:

$$
\left(Xw-y\right)^{T}\left(Xw-y\right) = w^{T}X^{T}Xw-2y^{T}Xw+y^{T}y
$$

Let’s complete the square with$M = X^{T}X$ and$b = X^{T}y$. Then

$$
\begin{aligned}
\left(Xw-y\right)^{T}\left(Xw-y\right) &= \left(w-\left(X^{T}X\right)^{-1}X^{T}y\right)X^{T}X\left(w-\left(X^{T}X\right)^{-1}X^{T}y\right) \\
&\quad -y^{T}X\left(X^{T}X\right)^{-1}X^{T}y+y^{T}y \\
&= \left(w-\hat{w}\right)X^{T}X\left(w-\hat{w}\right)-y^{T}X\hat{w}+y^{T}y
\end{aligned}
$$

Putting it together,

$$
\begin{aligned}
\hat{R}_n(w) &= \frac{1}{n}\left(Xw-y\right)^{T}\left(Xw-y\right) \\
&= \frac{1}{n}\left(w-\hat{w}\right)X^{T}X\left(w-\hat{w}\right)-\frac{1}{n}y^{T}X\hat{w}+\frac{1}{n}y^{T}y \\
&= \frac{1}{n}\left(w-\hat{w}\right)X^{T}X\left(w-\hat{w}\right)+\hat{R}_n(\hat{w})
\end{aligned}
$$

3. [Optional] Using the expression derived for $\hat{R}_n(w)$ in 2, give a very short proof that $\hat{w} = \left(X^T X\right)^{-1} X^T y$ is the empirical risk minimizer. That is:

$$\hat{w} = \arg\min_w \hat{R}_n(w).$$

   Hint: Note that $X^T X$ is positive semidefinite and, by definition, a symmetric matrix M is positive semidefinite iff for all $x \in \mathbf{R}^d$, $x^T M x \geq 0$.

   Solution:

   Since $X^T X$ is positive semidefinite, $(w - \hat{w})\, X^T X\, (w - \hat{w}) \geq 0$ for all w. In particular, it achieves its minimum value of 0 when $w = \hat{w}$. To spell it out in more detail, note that

$$\begin{aligned}
\arg\min_w \hat{R}_n(w) &= \arg\min_w \frac{1}{n}\left((w - \hat{w})\, X^T X\, (w - \hat{w})\right) + \hat{R}_n(\hat{w}) \\
&= \arg\min_w \frac{1}{n}\left((w - \hat{w})\, X^T X\, (w - \hat{w})\right) \\
&= \arg\min_w \left((w - \hat{w})\, X^T X\, (w - \hat{w})\right) \\
&= \hat{w}
\end{aligned}$$

4. [Optional] Give an expression for the set of w for which the empirical risk exceeds the minimum empirical risk $\hat{R}_n(\hat{w})$ by an amount $c > 0$. If X is full rank, then $X^T X$ is positive definite, and this set is an ellipse – what is its center?
   Solution:

   We're talking about the set of all w's that satisfy the following (equivalent) equations:

$$\begin{aligned}
\hat{R}_n(w) - \hat{R}_n(\hat{w}) &= c \\
c &= \frac{1}{n}\left((w - \hat{w})\, X^T X\, (w - \hat{w})\right) \\
cn &= (w - \hat{w})\, X^T X\, (w - \hat{w})
\end{aligned}$$

   When $X^T X$ is positive definite, this is an equation of an ellipse centered at $\hat{w}$.

6 [Optional] Projected SGD via Variable Splitting

In this question, we consider another general technique that can be used on the Lasso problem. We first use the variable splitting method to transform the Lasso problem to a differentiable problem with linear inequality constraints, and then we can apply a variant of SGD.

Representing the unknown vector θ as a difference of two non-negative vectors θ^+ and θ^-, the

ℓ_1-norm of θ is given by $\sum_{i=1}^{d} \theta_i^+ + \sum_{i=1}^{d} \theta_i^-$. Thus, the optimization problem can be written as

$$(\hat{\theta}^+, \hat{\theta}^-) = \underset{\theta^+, \theta^- \in \mathbf{R}^d}{\arg\min} \sum_{i=1}^{m} (h_{\theta^+,\theta^-}(x_i) - y_i)^2 + \lambda \sum_{i=1}^{d} \theta_i^+ + \lambda \sum_{i=1}^{d} \theta_i^-$$
$$\text{such that } \theta^+ \geq 0 \text{ and } \theta^- \geq 0,$$

where $h_{\theta^+,\theta^-}(x) = (\theta^+ - \theta^-)^T x$. The original parameter θ can then be estimated as $\hat{\theta} = (\hat{\theta}^+ - \hat{\theta}^-)$.

This is a convex optimization problem with a differentiable objective and linear inequality constraints. We can approach this problem using projected stochastic gradient descent, as discussed in lecture. Here, after taking our stochastic gradient step, we project the result back into the feasible set by setting any negative components of θ^+ and θ^- to zero.

1. [Optional] Implement projected SGD to solve the above optimization problem for the same λ's as used with the shooting algorithm. Since the two optimization algorithms should find essentially the same solutions, you can check the algorithms against each other. Report the differences in validation loss for each λ between the two optimization methods. (You can make a table or plot the differences.)

Solution:

```
def positive_project(x):
    return np.maximum(x, 0, x)

def projection_SGD_split(X, y, theta_positive_0, theta_negative_0, lambda_reg = 1.0,
     alpha = 0.1, num_iter = 1000):
    m, n = X.shape
    theta_positive = np.zeros(n)
    theta_negative = np.zeros(n)
    theta_positive[0:n] = theta_positive_0
    theta_negative[0:n] = theta_negative_0
    times = 0
    theta = theta_positive - theta_negative
    loss = compute_sum_sqr_loss(X, y, theta)
    loss_change = 1.
    while (loss_change>1e-6) and (times<num_iter):
        loss_old = loss
        for i in range(m):
            X_sample = X[i, :]
            y_sample = y[i]
            var_1 = np.dot(X_sample, theta.T)
            var_2 = var_1 - y_sample
            var_3 = 2*var_2*X_sample
            grad_positive = var_3 + lambda_reg
            grad_negative = (-1.)*var_3 + lambda_reg
            theta_positive = positive_project(theta_positive - alpha*grad_positive)
            theta_negative = positive_project(theta_negative - alpha*grad_negative)
            theta = theta_positive - theta_negative
        loss = compute_sum_sqr_loss(X, y, theta)
        loss_change = np.abs(loss - loss_old)
        times +=1
```

```
    print('(SGD) Ran for {} epochs. Loss:{} Lambda: {}'.format(times,loss,lambda_reg
    ))
    return theta
```

2. [Optional] Choose the λ that gives the best performance on the validation set. Describe the solution $\hat{w}$ in term of its sparsity. How does the sparsity compare to the solution from the shooting algorithm?

   Solution:
   $\lambda = 0.0073$ minimizes the square loss on the validation set in this case. For the chosen λ, with true value 0 as the threshold, the sparsity is 0. With threshold 10^{-3}, the sparsity is 39. With threshold 10^{-2}, the sparsity is 70. The sparsity is better than the shooting method.