NYU Center for Data Science

,

DS-GA 1003: Machine Learning (Spring 2020)

Homework 3: Lasso Regression and Projected Gradient Descent

Due: Monday, March 2, 2020 at 12pm

In this problem set, you will implement two approaches to lasso regression. Along the way, you will compare ridge regression and lasso regression on a sparse dataset.

Instructions. You should upload your code and plots to Gradescope. Please map the Gradescope entry on the rubric to your name and NetId. You must follow the policies for submission detailed in Homework 0.

Dataset. We have provided you with several files accessible through JupyterHub or GitHub. We encourage you to modify the code in `lasso_regression_skeleton_code.ipynb` for your implementation of Lasso Regression.

Dataset

We are generating data using code in the file `feature_transformation.ipynb`. Note that the data was serialized to `lasso_data.pickle` 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 the generation of the target function and the data by studying the code in `feature_transformation.ipynb`. Note that 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 `feature_transformation.ipynb`.

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 `feature_transformation.ipynb`. 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.

You should work through the file `ridge_regression.ipynb`. You'll go through the following steps

1. Load the problem from `load_problem`. Use the featurize function to map from a one-dimensional input space to a d-dimensional feature space.
2. 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

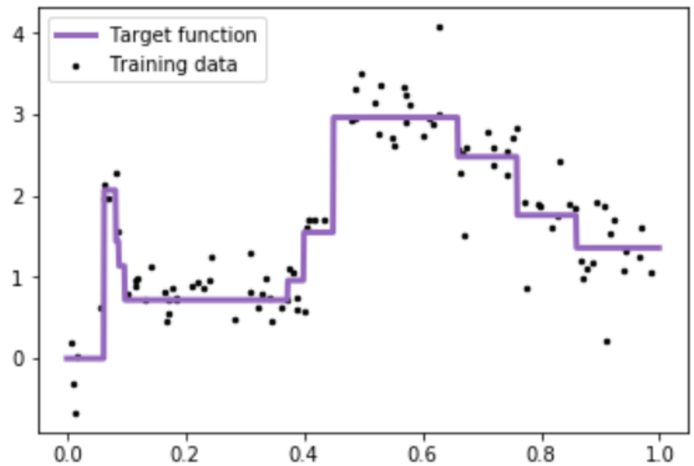

Figure 1: Training data and Target function

that on your own.)

3 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.

4 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."

5 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 to find the optimal L2 regularization parameter for the data provided.

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

7 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.

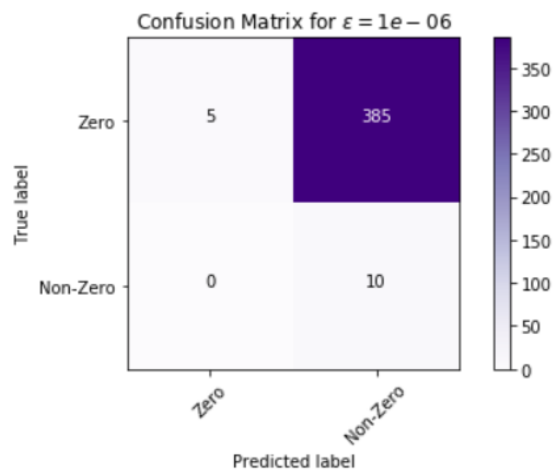

Figure 2: Confusion Matrix

Ridge Regression

In the problems below, you do not need to implement ridge regression. You may use `ridge_regression.ipynb`. Your results are for the ridge regression objective function, 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. Your images should resemble Figure 2

Lasso Regression

The Lasso optimization problem can be formulated as

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

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. Please see the notes for Lecture 4.

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 w = (X^T X + λI)^{-1} X^T y;
repeat
    for j = 1, ..., D do
        a_j = 2 Σ_{i=1}^n x_{ij}^2;
        c_j = 2 Σ_{i=1}^n x_{ij}(y_i − w^T x_i + w_j x_{ij});
        w_j = soft(c_j/a_j, λ/a_j);
until converged;
```

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

The "soft thresholding" function is defined as

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

for any $a, \delta \in \mathbf{R}$.

Note that the algorithm 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 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 an optional problem, you can show that for any $\lambda \geq 2\|X^T y\|_\infty$, the estimated weight vector $\hat{w}$ is entirely zero, where $\|\cdot\|_\infty$ is the infinity norm (or supremum norm), which is the maximum over the absolute values of the components of a vector. Thus we need to search for an optimal λ in $[0, \lambda_{\max}]$, where $\lambda_{\max} = 2\|X^T y\|_\infty$.

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** . The resulting set of parameter values $\hat{w}(\lambda)$ as λ ranges over $[0, \lambda_{\max}]$ is known as a **regularization path**.

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. 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).

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).

3 Run your best Lasso configuration on the training dataset provided, and select the λ that minimizes the square error 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?

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).

5 Note that the data in Figure 1 is almost entirely nonnegative. Since we don't have an unregu-

larized bias term, we have "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.

Throughout the problem you are encouraged to use the code provided to you in `lasso_regression_skeleton_code.ipynb`.

Projected Gradient Descent

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 $\hat{\theta} = (\hat{\theta}^+ - \hat{\theta}^-)$. Please see the derivation in Lecture 4 slides.

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 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. Please plot the differences as in Figure 3.

2 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?

Throughout the problem you are encouraged to use the code provided to you in `lasso_regression_skeleton_code.ipynb`.

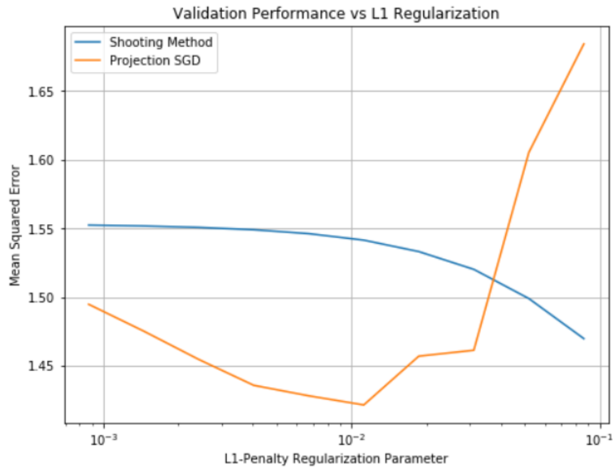

Figure 3: Comparison of Coordinate Descent and Projected SGD

Deriving $\lambda_{\max}$ (Optional)

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 $\lambda \geq 2\|X^T y\|_\infty$, 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.]

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.

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 $\lambda_{\max} = 2\|X^T y\|_\infty$. Conclude that $w = 0$ is a minimizer of $J(w)$ if and only if $\lambda \geq 2\|X^T y\|_\infty$.

Acknowledgement: This problem set is based on assignments developed by David Rosenberg of NYU and Bloomberg.