Machine Learning and Computational Statistics, Spring 2015
Homework 2: Lasso

Due: Friday, February 13, 2015, at 4pm (Submit via NYU Classes)

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. You may include your code inline or submit it as a separate file. You may either scan hand-written work or, preferably, write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython).

1 Preliminaries

1.1 Dataset construction

Start by creating a design matrix for regression with $m = 150$ examples, each of dimension $d = 75$. We will choose a true weight vector $\boldsymbol{\theta}$ that has only a few non-zero components:

1. Let $X \in \mathbf{R}^{m \times d}$ be the "design matrix," where the i'th row of X is $x_i \in \mathbf{R}^d$. Construct a random design matrix X using `numpy.random.rand()` function.

2. Construct a true weight vector $\boldsymbol{\theta} \in \mathbf{R}^{d \times 1}$ as follows: Set the first 10 components of $\boldsymbol{\theta}$ to 10 or -10 arbitrarily, and all the other components to zero.

3. Let $y = (y_1, \ldots, y_m)^T \in \mathbf{R}^{m \times 1}$ be the response. Construct the vector $y = X\boldsymbol{\theta} + \epsilon$, where ϵ is an $m \times 1$ random noise vector generated using `numpy.random.randn()` with mean 0 and standard deviation 0.1.

4. Split the dataset by taking the first 80 points for training, the next 20 points for validation, and the last 50 points for testing.

Note that we are not adding an extra feature for the bias in this problem. By construction, the true model does not have a bias term.

1.2 Experiments with Ridge Regression

By construction, we know that our dataset admits a sparse solution. Here, we want to evaluate the performance of ridge regression (i.e. ℓ_2-regularized linear regression) on this dataset.

1. Run ridge regression on this dataset. Choose the λ that minimizes the square loss on the validation set. For the chosen λ, examine the model coefficients. Report on how many components with true value 0 have been estimated to be non-zero, and vice-versa (don't worry if they are

all nonzero). Now choose a small threshold (say 10^{-3} or smaller), count anything with magnitude smaller than the threshold as zero, and repeat the report. (For running ridge regression, you may either use your code from HW1, or you may use `scipy.optimize.minimize` (see the demo code provided for guidance). For debugging purposes, you are welcome, even encouraged, to compare your results to what you get from `sklearn.linear_model.Ridge`.)

2 Lasso

The Lasso optimization problem can be formulated as

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

where $h_w(x) = w^T x$.

Since the ℓ_1-regularization term in the objective function is non-differentiable, vanilla gradient descent cannot solve this optimization problem. Next, we will learn two techniques to solve the optimization problem.

2.1 Shooting algorithm

One standard way to solve an optimization problem is coordinate descent, in which at each step, we optimize over one component of the unknown 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:

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

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

Note that Murphy is suggesting to initialize the optimization with the ridge regession solution, though this is not necessary.

The solution should have a sparsity pattern that is similar to the ground truth. Estimators that preserve the sparsity pattern (with enough training data) are said to be **"sparsistent[1]"** (sparse + consistent). Formally, an estimator $\hat{\beta}$ of parameter β is said to be consistent if the estimator $\hat{\beta}$ converges to the true value β in probability. Analogously, if we define the support of a vector β as the indices with non-zero components, i.e. $\text{Supp}(\beta) = \{j \mid \beta_j \neq 0\}$, then an estimator $\hat{\beta}$ is said to be sparsistent if as the number of samples becomes large, the support of $\hat{\beta}$ converges to the support of β, or $\lim_{m\to\infty} P[\text{Supp}(\hat{\beta}_m) = \text{Supp}(\beta)] = 1$.

There are a few tricks that can make selecting the hyperparameter λ easier and faster. First, you can show that for any $\lambda > 2\|X^T(y-\bar{y})\|_\infty$, 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 absolute value of any component of the 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 $h_{w,b}(x) = w^T x + b$. For the experiments, you can exclude the bias term, in which case $\lambda_{\max} = 2\|X^T y\|_\infty$.)

Second, we can make use of the fact that when λ and λ' are close, so are the corresponding solutions $\hat{w}(\lambda)$ and $\hat{w}(\lambda')$. Start by finding $\hat{w}(\lambda_{\max})$ and initialize the optimization at $w = 0$. Next, λ is reduced (e.g. by a constant factor), and the optimization problem is solved using the previous optimal point as the starting point. This is called **warm starting** the optimization. The entire technique of computing a set of solutions for a chain of nearby λ's is called a **continuation** or **homotopy method**. In the context of finding a good regularization hyperparameter, it may be referred to as a **regularization path** approach. (Lots of names for this!)

1. Write a function that computes the Lasso solution for a given λ using the shooting algorithm described above. This function should take a starting point for the optimization as a parameter. Run it on the dataset constructed in (1.1), and select the λ that minimizes the square error on the validation set. Report the optimal value of λ found, and the corresponding test error. Plot the validation error vs λ.

2. Analyze the sparsity of your solution, reporting how many components with true value zero have been estimated to be non-zero, and vice-versa.

3. Implement the homotopy method described above. Compare the runtime for computing the full regularization path (for the same set of λ's you tried in the first question above) using the homotopy method compared to the basic shooting algorithm.

4. The algorithm as described above is not ready for a large dataset (at least if it has been implemented in basic Python) because of the implied loop over the dataset (i.e. where we sum over the training set). By using matrix and vector operations, we can eliminate the loops. This is called "vectorization" and can lead to dramatic speedup in languages such as Python, Matlab, and R. Derive matrix expressions for computing a_j and c_j. (Hint: A matlab version of this vectorized method can be found here: `https://code.google.com/p/pmtk3/source/browse/trunk/toolbox/Variable_selection/lassoExtra/LassoShooting.m?r=1393`).

5. Implement the matrix expressions and measure the speedup to compute the regularization path.

[1] Li, Yen-Huan, et al. "Sparsistency of l_1-Regularized M-Estimators."

6. (Optional) Derive the expression used in the algorithm for the coordinate minimizer w_j. (Hint: Most of it is worked out in KPM's book as well as our slides on subgradients. The missing piece is writing the derivative of the empirical loss in terms of a_j and c_j.)

7. (Optional) Derive the above expression for $\lambda_{\max}$.

2.2 (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 smooth 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.)

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?

2.3 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. Derive the relation between $\hat{\theta}_i$ and $\hat{\theta}_j$, the i^{th} and the j^{th} components of the optimal weight vector obtained by solving the Lasso optimization problem.

[Hint: Assume that in the optimal solution, $\hat{\theta}_i = a$ and $\hat{\theta}_j = b$. First show that a and b must have the same sign. Then, using this result, rewrite the optimization problem to derive a relation between a and b.]

2. Derive the relation between $\hat{\theta}_i$ and $\hat{\theta}_j$, the i^{th} and the j^{th} components of the optimal weight vector obtained by solving the ridge regression optimization problem.

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.

3 Feedback (not graded)

1. Approximately how long did it take to complete this assignment?

2. Any other feedback?