

# Tutorial for GraphLab Python API: Lasso Shooting Algorithm

Aapo Kyrölä  
Department of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213  
akyrola@cs.cmu.edu

December 10, 2010

## 1 Introduction

This simple tutorial shows how to write a solver for linear regression with L1-penalty (Lasso) using the Python API for GraphLab.

We present the famous Lasso [3] optimization problem as follows:

$$\arg \min_x \|Ax - y\|_2^2 + \lambda \|x\|_1$$

Here  $\lambda$  is a parameter that determines the *sparsity* of the solution vector. Higher values result in sparser solutions, as the minimization emphasizes the L1-penalty at the cost of the least squares fit. Matrix  $A$  is (preferably) a sparse *data matrix* and  $y$  is the vector of *observations*;  $x$  is the unknown vector of predictors.

This problem can be efficiently solved using *coordinate minimization*. This was first introduced by Wenjiang Fu in [2], and the algorithm is called “Shooting”. It minimizes the objective by cyclically setting each  $x_i$  to the minimum, keeping other  $x_j, i \neq j$  fixed. As the objective is convex, this is guaranteed to eventually converge to the global minima.

The *shoot* for each coordinate is then done as follows:

$$\begin{aligned} S_j &= A_{(:,j)}^T (Ax) - 2(y^T A)_j + (A^T A)_{j,j} x_j^{t-1} \\ x_j^t &\leftarrow \text{sign}(S_j) (|S_j| - \lambda)_+ \end{aligned}$$

Above, operator  $(\cdot)_+$  denotes soft thresholding and  $A_{(:,j)}$  denotes column  $j$  of  $A$ .

A very important optimization is based on the observation that  $Ax$  in the algorithm is a vector, and we can update it efficiently without recomputing it. Let us write  $z = Ax$ . Then the *shoot* can be written as follows, amended with an efficient update of  $z$ :

$$\begin{aligned}
S_j &= A_{(:,j)}^T z^{t-1} - 2(y^T A)_j + (A^T A)_{j,j} x_j^{t-1} \\
x_j^t &\leftarrow \text{sign}(S_j) (|S_j| - \lambda)_+ \\
z^{t+1} &\leftarrow z_t + A_{(:,j)}^T (x_j^{t+1} - x_j^t)
\end{aligned} \tag{1}$$

Note also that  $(y^T A)_j$  is constant, as well as the covariance  $(A^T A)_{j,j}$ , and can thus be precomputed.

## 2 GraphLab representation

We now represent the Shooting algorithm under GraphLab abstraction. The code can be found under directory `python/`.

### 2.1 Data Graph

We can see from Equation 1, that each variable  $x_j$  is dependent on such elements  $k$  of  $z_k$  that  $A_{j,k} \neq 0$ . Assuming  $A$  is sparse, this suggests a natural bipartite graph representation:

- On the “left” side, create a vertex for each variable  $x_j$ .
- On the “right” side, create a vertex for each variable  $z_k$ . Note that  $z = Ax$ , and so  $z_k$  is an estimate of observed variable  $y_k$ . Thus, we store the observation  $y_k$  to each vertex on the right side. This helps in computing the objective value.
- Connect vertices from left to right, if corresponding  $A_{j,k} \neq 0$ . Store the value of the matrix element in the edge.

For our Python implementation, we define following two Python classes for the vertices:

```

class lasso_estimate_vertex:
    def __init__(self, value, observed):
        self.curval = value
        self.lastval = value
        self.observed = observed
        self.vtype = 1

class lasso_variable_vertex:
    def __init__(self, value):
        self.value = value
        self.covar = 0
        self.Ay = 0
        self.initialized = False
        self.vtype = 0

```

For vertices on the “right” side,  $z_k$ , we use `lasso_estimate_vertex` and for variables  $x_j$  `lasso_variable_vertex`. For the latter, note that we have field `self.initialized` set to `False`. When *update function* is first run, it will compute values for fields `Ay` and `covar` ( $= A_{j,j}$ ).

We have provided a simple dataset “testphoto.txt” under `python/testdata`. This is a simple compressed sensing dataset with 954 variables and 477 observed values. That is, matrix  $A$  has dimensions  $477 \times 954$ .

Script to load the graph data is in file **lasso\_load.py**.

## 2.2 Update function

Update function simply implements the shooting step of Equation 1. In addition, if a variable is not initialized, its covariance and offset  $(y^T A)_j$  is computed. This allows us to perform initialization in parallel. Update is only executed for the variable vertices, which read and write values in the neighboring  $z_k$  -vertices. Update function is listed below, and is in file **lasso\_update.py**.

```
# ||Ax-y||_2^2 + lambda ||x||_1

import math
lamb = 0.5

def update(scope, scheduler):
    # Of class lasso_variable_vertex or lasso_estimate_vertex
    lassov = scope.getVertex().value

    if (lassov.vtype == 0):
        if lassov.initialized == False:
            # Initialize covariance
            lassov.covar = 2.0*sum([e.value*e.value for e in scope.getOutboundEdges()])
            # Initialize (Ay)_i
            lassov.Ay = 2.0*sum([e.value * scope.getNeighbor(e.to).value.observed for e in scope.getOutboundEdges()])
            lassov.initialized = True

        # Compute (Ax)_i
        curest = sum([e.value * scope.getNeighbor(e.to).value.curval for e in scope.getOutboundEdges()])
        newval = soft_threshold(lamb, curest*2 - lassov.covar*lassov.value - lassov.Ay)/lassov.covar
        if newval != lassov.value:
            delta = newval-lassov.value
            lassov.value = newval
            for e in scope.getOutboundEdges():
                scope.getNeighbor(e.to).value.curval += delta * e.value

def soft_threshold (lamb, x):
    if (x > lamb):
        return (lamb-x)
    elif (x < -lamb):
        return (-lamb-x)
    else:
        return 0.0

update(scope, scheduler)
```

## 2.3 GraphLab parameters

We run the algorithm with round robin schedule for a hundred full iterations. Below is the contents of script **lasso\_config.py**.

```
graphlab.setScheduler("round_robin")
graphlab.setScopeType("vertex")
graphlab.setIterations(100)
```

## 2.4 Post-processing

Finally, script **lasso\_post.py** computes the objective value  $\|Ax - y\|_2^2 + \lambda\|x\|_1$ , and outputs it to the console. The optimized values of  $x_j$  are now stored in the graph.

## 2.5 Running the application

Run script **examples/python\_lasso.sh**.

## 2.6 Is Shooting algorithm parallel?

It might look dubious that we allow optimizing several coordinates in parallel. Indeed, this is not allowed in principle, since variables are dependent on each other via  $z = Ax$ . However, in practice, if  $A$  is sparse, the dependencies are rare and optimization seems to work most of the times. However, if in doubt, GraphLab also allows running the algorithm with one thread only or with a stronger *scope consistency*.

## 3 Notes

To further improve on performance, it is advisable to do pathwise optimization. This means starting with a high value for  $\lambda$ , solve the optimization for a decreasing sequence of lambdas using the previous results as a warm start. See [1] for further information.

## References

- [1] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Department of Statistics, Stanford University, Tech. Rep*, 2008.
- [2] Wenjiang J. Fu. Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.
- [3] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.