ℓ_1 - Norm problems: Least Absolute Deviations, Huber Fitting and Lasso using ADMM

Report
MIS-4 End semester Project

Submitted by:
Team-7

DABBARA HARSHA - CB.EN.U4AIE20010

DIVI ESWAR CHOWDARY - CB.EN.U4AIE20012

B E PRANAV KUMAAR – CB.EN.U4AIE20052

RISHEKESAN S V – CB.EN.U4AIE20058

Under the supervision of
Prof.(Dr.) V. Sowmya

AMRITA SCHOOL OF ENGINEERING

04 July 2022

CONTENTS

Abstract

This study provides insights into ℓ_1 - Norm Problems based on Book[1]. which includes Least Absolute Deviation, Hubber Fitting and Least Absolute Shrinkage and Selection Operator[LASSO]. Throughout the Study We will be working on fundamentally understanding the mathematical concepts relating ℓ_1 - Norm Problems using Alternating direction method of multipliers[ADMM] using which we implement the same and verify its integrity using code.

I. Introduction

The L0 norm is the number of non-zero elements in a vector. The L0 norm is popular in compressive sensing which tries to find the sparsest solution to an under determined set of equations.

The L1 norm that is calculated as the sum of the absolute values of the vector.

$$||x||_1 = \sum_{i=1}^{N} |x_i|$$

The L2 norm that is calculated as the square root of the sum of the squared vector values. The max norm that is calculated as the maximum vector values.

$$||x||_2 = \sqrt{\sum_{i=1}^{N} |x_i|^2}$$

The list of problems we attempt to solve using ADMM are of similar form to L1 norm. Hence, they are referred as L1 norm problems. On further exploration we will also find that all of them fall under the ADMM form 1 solution format. Let's begin by exploring the precursors to ADMM,

II. Dual Ascent

Let us consider the following Convex constrained optimization problem:

$$\min_{x \in R^n} f(x) \tag{II.1}$$

$$s.t. \mathbf{A}x = \mathbf{b}. \tag{II.2}$$

Here, $\mathbf{A} \in R^{m \times n}$, $\mathbf{b} \in R^{m}$, and $f : R^{n} \rightarrow R$ is a convex function. y is a Lagrange multiplier Lagrange

$$L(x, y) = f(x) + y^t(Ax - b) \tag{II.3}$$

we can dual function for II.1 the above as

$$g(y) = \inf_{x \in R^n} L(x, y) \tag{II.4}$$

$$= -f^*(-A^T y) - b^T y \tag{II.5}$$

Here f^* is a convex conjugate of f.Now ,we have to optimize the dual function

$$\max_{y \in R^n} f(y) \tag{II.6}$$

Considering that strong duality holds for the above .The primal and dual solution are same.with this primal solution x^* is can recovered from dual optimal solution y^*

$$x^* = arg \min_{x \in R^n} L(x, y^*) \tag{II.7}$$

Now,Coming to Dual Ascent Method ,we use gradient ascent method to find out the residual for the equality constraint
Updating of Variables can be done in following way

1) $x^* = arg \min_{x \in R^n} L(x, y^*)$
2) $\nabla g(y^*) = Ax^* - b.$

Now the Iterative updates are

$$x^{k+1} = arg \min_{x \in R^n} L(x, y^k) \text{(minimization step)} \tag{II.8}$$

$$y^{k+1} = y^k + \alpha^k (A * X^k + 1 - b) \text{(for dual variable update)} \tag{II.9}$$

where α^k is the step-size.Finally, this algorithm is called **'dual ascent'** since dual function in II.4 increases in each step $(g(y^{k+1}) > g(y^k))$ with appropriate choice of α^k

A. Deriving the Dual Function

Now let us Understand How we got the dual in II.5

$$\min_{x \in R^n} f(x) \tag{II.10}$$
$$s.t. \mathbf{A}x = \mathbf{b}.$$

Formulating the Lagrangian for the Above Equation

$$L(x, y) = f(x) + y^t(A^x - b) \tag{II.11}$$

From the Above Equation II.11 ,we can understand that we have maximize the Lagrangian Multiplier ***y*** and minimize the function II.10

$$\min_x \max_y f(x) + y^t(A^x - b) \tag{II.12}$$

by Strong Duality

$$\max_y \min_x f(x) + y^t(A^x - b) \tag{II.13}$$

Now let us Expand the inner min of above Function

$$\min_x f(x) + y^t(A^x - b) \tag{II.14}$$
$$\min_x f(x) + \min_x y^t Ax - \min_x y^t b \tag{II.15}$$

$(AB)^T = B^T A^T$

$$\min_x f(x) + \min_x (A^t y)^t x - \min_x y^t b \tag{II.16}$$

Now we have to convert the min formulation to supremum Formulation

$$-\sup_x -f(x) + (A^t y)^t x - \min_x b^t y \tag{II.17}$$

After solving it We get

$$g(y) = -f^*(-A^T y) - b^y \tag{II.18}$$

f^* is a conjuagte Function

$$f^*(y) = \sup_{x \in dom f} y^t - f(x) \tag{II.19}$$

III. Dual Decomposition

The major benefit of the dual ascent method is that it can lead to a decentralized algorithm.Suppose objective f is separable, let's breakdown the objective function $f(x)$ as

$$f(x) = \sum_{i=1}^{N} f_i(x_i) \tag{III.1}$$

where x = [x1, x2, $\cdots$, xN] and each $x_i \in R^{n_i}$ is a sub-vector of x. Similarly, partitioning A = [A1, A2, $\cdots$, AN], we get

$$Ax = \sum_{i=1}^{N} A_i x_i \tag{III.2}$$

$$L(x, y) = \sum_{i=1}^{N} L_i(x_i, y) \tag{III.3}$$

$$L(x, y) = \sum_{i=1}^{N} (f_i(x) + y^T A_i x_i - \frac{1}{N} y^t b) \tag{III.4}$$

Clearly, the x-minimization step in II.8 has now been split into N separate problems that can be solved parallely. Hence, the update steps are

$$x^{k+1} = arg \min_{x \in R_i^n} L_i(x_i, y^k)(\text{minimization step}) \tag{III.5}$$

$$y^{k+1} = y^k + \alpha^k (A * X^k + 1 - b)(\text{for dual variable update}) \tag{III.6}$$

The minimization step in III.5 (is solved in parallel for each i = 1, 2, $\cdots$, N. Consequently, this decomposition in the dual ascent method is referred as the dual decomposition.

IV. Augmented Lagrangians and Method of Multipliers

The convergence of dual ascent algorithm is based on assumptions such as strict convexity or finiteness of f. To avoid such assumptions and ensure robustness to the dual ascent algorithm, Augmented Lagrangian methods were developed.

$$L_\rho = f(x) + y^t(Ax - b) + \frac{\rho}{2}||Ax - b||_2^2 \tag{IV.1}$$

Here $\rho > 0$ is called the penalty parameter. This augmented Lagrangian in IV.1 can be viewed as the unaugmented Lagrangian of the problem

$$\min_{x \in R^n} f + \frac{\rho}{2}||Ax - b||_2^2 \tag{IV.2}$$

$$s.t. \mathbf{A}x = \mathbf{b} \tag{IV.3}$$

Here IV.2 and II.1 are similar because for any feasible x, the term $\frac{\rho}{2}||Ax - b||_2^2$ becomes zero. The associated dual to IV.1 is

$$g_\rho(y) = \inf_{x \in R^n} L_\rho(x, y) \tag{IV.4}$$

Adding $\frac{\rho}{2}||Ax - b||_2^2$ to $f(x)$ makes g_ρ differentiable under milder conditions than on the original problem.

Next, the gradient of the augmented dual function is found similarly as above. Applying the dual ascent algorithm to the modified problem in IV.2 gives

$$x^{k+1} = arg \min_{x \in R^n} L_\rho(x, y^k) \tag{IV.5}$$

$$y^{k+1} = y^k + \rho(A * X^k + 1 - b) \tag{IV.6}$$

a) The Above equations IV.5 and IV.6 are called ***Method of Multipliers****:* This method is same as the standard dual ascent, with the penalty parameter ρ is used in place of step size α_k. The advantage of the method of multipliers is that it converges under far more general conditions than dual ascent.

V. **ADMM** - Alternating Direction Method of Multipliers

Alternating Direction Method of Multipliers or ADMM is an algorithm which tries solves the problem faced using Method of Multipliers by combining the decomposability of dual ascent with the Greater convergence properties of Method of multipliers. ADMM is a simple and powerful iterative algorithm for convex optimization problems. It is almost 80 times faster for multivariable problems than conventional methods. ADMM replaces linear and quadratic programming in a single frame work.

ADMM solves the pronlems in particular form and adheres two forms

(A) Form 1
(B) Form 2

A. Form 1

Consider the unconstrained problem,
$\min_x \ f(x) + g(x)$

$$\text{Where} f : R^n \to R, g : R^n \to R \text{ are convex functions} \tag{V.1}$$

Now the alternating direction method of multipliers (ADMM) is defined as,

$$x^{(k+1)} := prox_{f\lambda}(z^k - u^k) \tag{V.2}$$
$$z^{(k+1)} := prox_{g\lambda}(x^{(k+1)} + u^k) \tag{V.3}$$
$$u^{(k+1)} := u^k + x^{(k+1)} - z^{(k+1)} \tag{V.4}$$

where k is the iteration counter

Derivation For Form I

Here we are changing the problem by introducing a new variable z and also importing a constraint. The variable has been split into two variables x and z . Therefore the number of variables has been doubled but the optimization problem remains same. Basically converting an unconstrained problem into a constrained one and trying to solve that by using the augmented Lagrangian

Now our Optimization Problem changed to

$$minf(x) + g(z) \tag{V.5}$$

$$\text{Subject to } x - z = 0 \tag{V.6}$$

And this is Equivalent to minimizing $f(x) + g(x)$

Now Lets Write Lagrangian For Above Equation V.5

$$L_\rho(x, y, z) = f(x) + g(z) + y^T(x - z) + \left(\frac{\rho}{2}\right) ||x - z||_2^2 \tag{V.7}$$

Where $\rho >0$ is a parameter and $y \in R^n$ is the Lagrangian dual variable

another term $(\rho/2)\ ||Ax - b||_2^2$ called augmented Lagrangian is also adding. This will be a highly convex function and so addition of this extra term increases the convexity of our original problem and it will converges fastly to the solution. So now there are three variables x,y and z

$$x^{k+1} := arg\min_x L_\rho\left(x, z^k, y^k\right) \tag{V.8}$$
$$y^{k+1} := arg\min_x L_\rho\left(x^{k+1}, z, y^k\right) \tag{V.9}$$
$$y^{k+1} := y^k + \rho\left(x^{k+1} - z^{k+1}\right) \tag{V.10}$$

In each iteration x and Z are using Previous values from Primal Solution and Y is accelerating term

Now we are trying to convert V.7 into proximal Form

For x and z values are Randomly chosen and trying to minimize the Lagrangian

$$x^{k+1} = arg\min_x f(x) + \langle y^k, x\rangle + ||x - z^k||_2^2 \tag{V.11}$$

$$z^{k+1} = arg\min_z g(z) + \langle y^k, z\rangle + ||x^{k+1} - z||_2^2 \tag{V.12}$$

$$y^{k+1} = y^k + \rho(x^{k+1} - z^{k+1}) \tag{V.13}$$

Arranging the all the linear terms into Quadratic form

$$x^{k+1} = arg\min_x f(x) + ||x - z^k + \frac{1}{\rho}y^k||_2^2 \tag{V.14}$$

$$z^{k+1} = arg\min_z g(z) + ||x^{k+1} - z + \frac{1}{\rho}y^k||_2^2 \tag{V.15}$$

$$y^{k+1} = y^k + \rho(x^{k+1} - z^{k+1}) \tag{V.16}$$

Now Let us make some assumptions that we will use those for make it to proximal Form

$$u^k = \frac{1}{\rho} y^k, \lambda = \frac{1}{\rho} \tag{V.17}$$

$$x^{k+1} = arg\min_x f(x) + ||x - z^k + u^k||_2^2 \tag{V.18}$$

$$z^{k+1} = arg\min_z g(z) + ||x^{k+1} - z + u^k||_2^2 \tag{V.19}$$

$$y^{k+1} = y^k + \rho(x^{k+1} - z^{k+1}) \tag{V.20}$$

i.e.

$$x^{k+1} = arg\min_x f(x) + ||x - (z^k - u^k)||_2^2 \tag{V.21}$$

$$z^{k+1} = arg\min_z g(z) + ||z - (x^{k+1} - u^k)||_2^2 \tag{V.22}$$

$$y^{k+1} = y^k + \rho(x^{k+1} - z^{k+1}) \tag{V.23}$$

Now As we know about Proximal Form

$$prox_{\lambda f}(v) = arg\min_x (f(x) + \frac{1}{2 * \lambda} ||x - v||_2^2) \tag{V.24}$$

Now Let us Write Everything into proximal Form

$$x^{k+1} := prox_{\lambda f}\left(z^k - u^k\right) \tag{V.25}$$

$$z^{k+1} := prox_{\lambda g}\left(x^{k+1} + u^k\right) \tag{V.26}$$

$$u^{k+1} := u^k + x^{k+1} - z^{k+1} \tag{V.27}$$

Now we could the Equation V.7 as Following by substituting assumption from V.17
Our New Equation is below

$$L_\rho(x, y, z) = f(x) + g(z) + \frac{1}{2 * \lambda} ||x - z + u||_2^2) \tag{V.28}$$

which is converted form This below Equation

$$L_\rho(x, y, z) = f(x) + g(z) + y^T(x - z) + \left(\frac{\rho}{2}\right) ||x - z||_2^2 \tag{V.29}$$

From Now on Our Lagrangian Function is V.28

B. Form II

Consider the constrained optimization problem,

$$\min_x f(x) \tag{V.30}$$

$$\text{Subject to } x \in C \tag{V.31}$$

where f and C are Convex

The Problem Now written in Form of **ADMM**,

$$\min_x f(x) + g(z) \tag{V.32}$$

$$\text{Subject to } x - z = 0 \tag{V.33}$$

Where g(.) is an indicator function of and is defined as

$$g(z) = \begin{cases} 0 \text{ if } z \in C \\ \infty \text{ Otherwise} \end{cases} \tag{V.34}$$

In the Above Equation all our constraints are represent into an indicator function $G(z)$ in a different variable and the variables made equal with introducing a new constraint. Now Lets us Consider the augmented Lagrangian function V.28

$$L_\rho(x, y, z) = f(x) + g(z) + \frac{1}{2 * \lambda}||x - z + u||_2^2 \tag{V.35}$$

According to the definition of the indicator function, if $g(z) = 0$ means we are actually minimizing the original function . So when the algorithm proceeds at any stage we are forced to choose an z , which is an element of C .If it is not, then $Z = \infty$ and further optimization is not possible.
For Minimizing the x we know the Solution of z^k and y^k

$$x^{k+1} = arg\min_x(f(x) + \frac{1}{2\lambda}(||x - z^k + u^k||_2^2) \tag{V.36}$$

Now ,We Have to be Care-full for Updating the Value of Z .Because if $Z = x + u$,means $\frac{1}{2\lambda}(||x - z^k + u^k||_2^2) = 0$,but $g(z) = \infty$. Here going for the concept of projection. In order to choose z which minimizes the Lagrangian the concept of projection is used.

The update of is obtained as,$z^{k+1} = \Pi_C\ (x^{k+1} + u^k\ )$, project$(x^{k+1} + u^k\ )$ onto space of C

$$z^{k+1} = \Pi_C\ (x^{k+1} + u^k\ ) \tag{V.37}$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1} \tag{V.38}$$

If $x - z = 0$ then $u^{k+1} = u^k$

Finally Iterative Steps are :

$$x^{k+1} = arg\min_x(f(x) + \frac{1}{2\lambda}(||x - z^k + u^k||_2^2) \tag{V.39}$$

$$z^{k+1} = \Pi_C\ (x^{k+1} + u^k\ ) \tag{V.40}$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1} \tag{V.41}$$

Now We are Done with Understanding the Overview of **ADMM** ,Now let use this on Few Problems .

This Includes the Following $L1$ Norm Problems

1) **L**east **A**bsolute **D**eviation
2) Huber Fitting problem
3) **L**east **A**bsolute **S**hrinkage and **S**election **O**perator.

VI. LAD

LAD is the acronym for **L**east **A**bsolute **D**eviation

A. Machine Learning Context

In Machine Learning, one of the main alternatives to the least-squares($\|Ax - b\|_2^2$) approach when attempting to estimate the parameters of a regression is the LAD($\|Ax - b1\|_1$) method. The use of the absolute value results in a more robust solution in handling outliers and has the possibility of multiple solutions. The LAD cost is formulated as,

$$J = \sum_{i=1}^{m} |Y_i - \Theta^T X| \tag{VI.1}$$

Now by Using the Weighted Least Square method we can write the same function as,

$$J = \sum_{i=1}^{m} \left(Y_i - \Theta^T X\right)^2 \left(1/|Y_i - \Theta^T X|\right) \tag{VI.2}$$

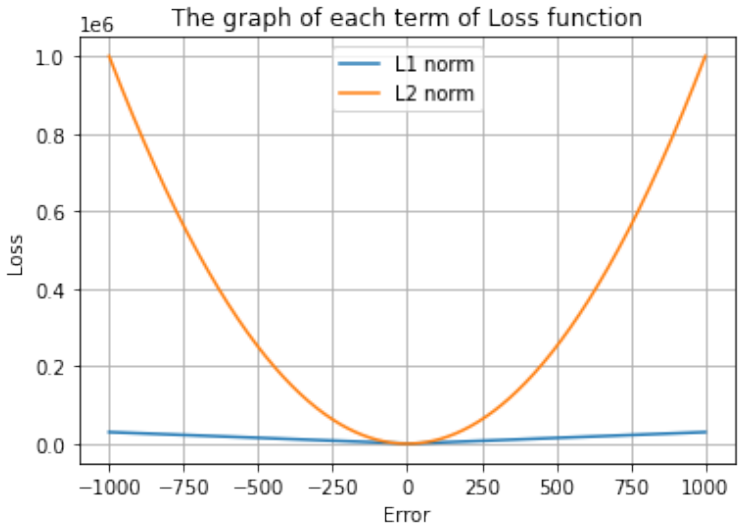


Fig. 1. The comparison of L1-norm, L2-norm loss vs Error

On rearranging the terms to obtain Θ_{new},

$$\Theta_{new} = \left(x^T * W * X\right)^{-1} * \left(x^T * W * Y\right) \tag{VI.3}$$

this serves as the update for Θ

From the plot in Fig 1, for a wide range of error values the L1 norm loss is significantly lesser than L2 norm loss, this supports the theory of handling outliers present in data by punishing the model less than the L2 norm will. This in turn eliminates the need to strictly pre-process the data. Hence, it is worth while to study this matheatically.

B. Solving the LAD Problem

The LAD problem can thus be defined as,

$$\text{minimize} \quad \|z\|_1 \tag{VI.4}$$

$$\text{subject to,} \quad Ax - b = z \tag{VI.5}$$

Solution,

The corresponding Augmented Lagrangian form can be written as,

$$L\left(x, y, z\right) = \|z\|_1 + y^T(Ax - b - z) + \rho/2\|Ax - b - z\|_2^2 \tag{VI.6}$$

Combining the y dependent term into the L2 norm term,

$$L\left(x, y, z\right) = \|z\|_1 + \rho/2\|Ax - b - z + (y/\rho)\|_2^2 \tag{VI.7}$$

Replacing y/ρ with u,

$$L\left(x, y, z\right) = \|z\|_1 + \rho/2\|Ax - b - z + u\|_2^2 \tag{VI.8}$$

In order to find the update for x, the minimization of the Lagrangian with respect to x is done as,

Assuming z^k and u^k are known,

$$x^{k+1} = \arg\min_x L_p(x, z^k, u^k) \tag{VI.9}$$

$$x^{k+1} = \arg\min_x 1/2\|Ax - b - z^k + u^k\|_2^2 \tag{VI.10}$$

Equating first order partial differential term with respect to x to **0** (0 vector) and simplifying to obtain x^{k+1}

$$\frac{\delta L\left(x^{k+1}, z, u^k\right)}{\delta z} = 2A^T\left(Ax - b - z^k - u^k\right) = \mathbf{0} \tag{VI.11}$$

$$x^{k+1} = \left(A^T A\right)^{-1} A^T (b + z^k - u^k) \tag{VI.12}$$

Similarly, in order to find the update for z, the minimization of the Lagrangian with respect to z is done as,

x^{k+1} has been computed and u^k is assumed to be known,

$$z^{k+1} = \arg\min_z L_p(x^{k+1}, z, u^k) \tag{VI.13}$$

Consider the z dependent terms,

$$z^{k+1} = \arg\min_z \quad \|z\|_1 + (\rho/2)\|Ax^{k+1} - b - z + u^k\|_2^2 \tag{VI.14}$$

bringing ρ into a common denominator constant,

$$z^{k+1} = \arg\min_z \quad \|z\|_1 + (1/(2/\rho))\|Ax^{k+1} - b - z + u^k\|_2^2 \tag{VI.15}$$

Proximal operator : Consider the proximal operator on a variable v with some constant α to be defined as,

$$prox_f(v) = \arg\min_x \left(f(x) + 1/2\alpha\|x - v\|_2^2\right) \equiv S_\alpha(v) \tag{VI.16}$$

On comparing equations VI.15 and VI.16, the similarity of the Right hand sides leads us to simplify the obtained z^{k+1} to be represented in a more compact form like,

$$z^{k+1} = S_{(1/\rho)}(Ax^{k+1} - b + u^k) \tag{VI.17}$$

Both x^{k+1} and z^{k+1} have been computed, using them the update for y can be written as,

$$u^{k+1} = u^k + (Ax^{k+1} - b - z^{k+1}) \tag{VI.18}$$

In essence,

$$x^{k+1} = \left(A^T A\right)^{-1} A^T (b + z^k - u^k)$$

$$z^{k+1} = S_{(1/\rho)}(Ax^{k+1} - b + u^k)$$

$$u^{k+1} = u^k + (Ax^{k+1} - b - z^{k+1})$$

VII. Huber Fitting problem

A. Machine Learning Context

In statistics and ML, The Huber loss is a loss function used in robust regression and is less sensitive to outliers in the data. The generalised form is,

$$J = (1/n)\sum_{i=1}^{n} \begin{cases} (1/2)(Y_i - h(X_i))^2 & |Y_i - h(X_i)| \le \delta \\ \delta|Y_i - h(X_i)| - 1/2 & |Y_i - h(X_i)| > \delta \end{cases}$$

The use of Huber fitting is justified by the fact that methods such as Least Square(LS) and Least Absolute Deviation(LAD) perform better than the other in different situations. So the best qualities of both are brilliantly combined to give us the Huber Fitting method.

From Fig 2, the plot of the Loss vs Prediction Error for the LAD(labeled as Absolute error), LS(labeled as Squared error) and Huber methods it is observable that the Huber loss is clearly lower than LAD and LS.
When we replace $Y_i - h(X_i)$ with z and δ with 1 we arrive at the starting point of Huber fitting problem.

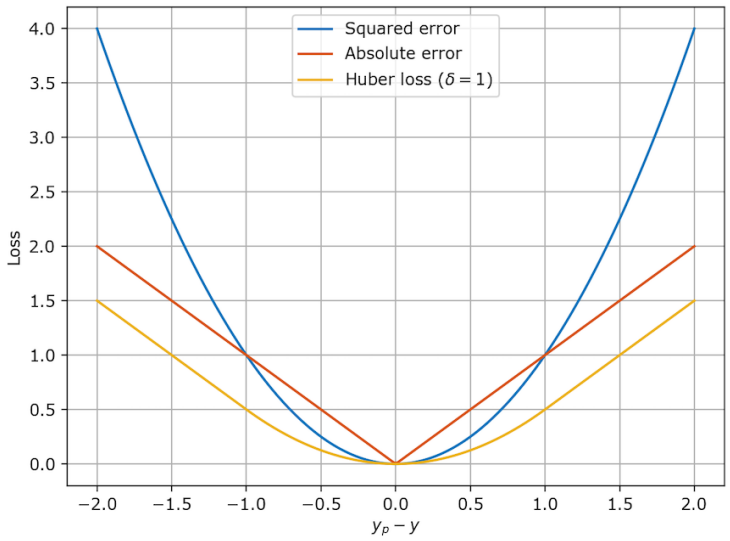


Fig. 2. Loss vs Error plot of LAD, LS and Huber

B. Solving Huber Fitting problem

$$\|z\|_h = \begin{cases} (1/2)z^2 & |z| \leq 1 \\ |z| - 1/2 & |z| > 1 \end{cases}$$

$$g^{hub}(z) = \sum_{i=1}^{m} huber(z_i) \tag{VII.1}$$

satisfying the given Huber conditions.

The Huber fitting problem can thus be defined as,

$$\text{minimize}_{x,z} \;\; g(z) \tag{VII.2}$$

$$\text{subject to,} \;\; Ax - b = z \tag{VII.3}$$

here x $\in$R^n, A is a matrix of size mxn and b is a vector of dimension $mx1$.

Solution,
The corresponding Augmented Lagrangian form can be written as,

$$L_\rho(x, y, z) = g(z) + y^T(Ax - b - z) + \frac{\rho}{2}\left(||Ax - b - z||_2^2\right) \tag{VII.4}$$

Replacing $\frac{y}{\rho}$ with u and $1/\rho$ with λ,

$$L_\rho(x, u, z) = g(z) + \frac{1}{2\lambda}||Ax - b - z + u||_2^2 \tag{VII.5}$$

In order to find the update for x, the minimization of the Lagrangian with respect to x is done as,

Assuming z^k and u^k are known,

$$L_\rho\left(x, u^k, z^k\right) = g\left(z^k\right) + \frac{1}{2\lambda}\left||Ax - b - z^k + u^k\right||_2^2 \tag{VII.6}$$

Equating first order partial differential term with respect to x to **0** (0 vector) and simplifying to obtain x^{k+1}

$$\frac{\delta L\left(x, z^k, u^k\right)}{\delta x} = A^T\left(Ax - b - z^k + u^k\right) = \mathbf{0} \tag{VII.7}$$

$$A^T Ax = A^T\left(b + z^k - u^k\right) \tag{VII.8}$$

Finally,

$$x^{k+1} = \left(A^T A\right)^{-1} A^T (b + z^k - u^k) \tag{VII.9}$$

Similarly, in order to find the update for z, the minimization of the Lagrangian with respect to z is done as,

x^{k+1} has been computed and u^k is assumed to be known,

$$L_\rho \left(x^{k+1}, u^k, z\right) = g\left(z\right) + \frac{1}{2\lambda} \left|\left|Ax^{k+1} - b - z + u^k\right|\right|_2^2 \tag{VII.10}$$

$$Note\ g\left(z\right)\ consist\ of\ two\ piece\ wise\ continuous\ functions. \tag{VII.11}$$

1) Case – I :

$$g\left(z\right) = \frac{z^2}{2}\ , then \tag{VII.12}$$

$$\frac{\delta L\left(x^{k+1}, z, u^k\right)}{\delta z} = z - \frac{1}{\lambda}\left(Ax^{k+1} - b - z + u^k\right) = \mathbf{0} \tag{VII.13}$$

$$\left(1 + \lambda\right) z^{k+1} = \left(Ax^{k+1} - b - u^k\right) \tag{VII.14}$$

$$z^{k+1} = \frac{1}{1+\lambda}\left(Ax^{k+1} - b + u^k\right) \tag{VII.15}$$

2) Case – II:

$$g\left(z\right) = |z| - \frac{1}{2}, then \tag{VII.16}$$

$$\frac{\delta L\left(x^{k+1}, z, u^k\right)}{\delta z} = (\frac{\lambda}{1+\lambda}) S_{\lambda}\left(Ax^{k+1} - b + u^k\right) = \mathbf{0} \tag{VII.17}$$

$$= \ (\frac{\lambda}{1+\lambda})\left(Ax^{k+1} - b + u^k\right) \pm \lambda \tag{VII.18}$$

$$z^{k+1} = (\frac{\lambda}{1+\lambda}) S_{\lambda}\left(Ax^{k+1} - b + u^k\right) \tag{VII.19}$$

here S_λ represents shrinkage by $\pm\lambda$ sign
Finally we can combine equations VII.15 and VII.19 to get,

$$z^{k+1} = \frac{1}{1+\lambda}\left(Ax^{k+1} - b + u^k\right) + (\frac{\lambda}{1+\lambda}) S_{\lambda}\left(Ax^{k+1} - b + u^k\right)$$

Both x^{k+1} and z^{k+1} have been computed, using them the update for y can be written as,
The original Lagrangian multiplier term is $u^T(Ax - b - z)$

$$u^{k+1} = u^k + (\ Ax^{k+1} - b - z^{k+1}) \tag{VII.20}$$

In essence,

$$x^{k+1} = \left(A^T A\right)^{-1} A^T \left(b + z^k - u^k\right)$$

$$z^{k+1} = \frac{1}{1+\lambda}\left(Ax^{k+1} - b + u^k\right) + \left(\frac{\lambda}{1+\lambda}\right) S_{\lambda}\left(Ax^{k+1} - b + u^k\right)$$

$$u^{k+1} = u^k + (\ Ax^{k+1} - b - z^{k+1})$$

VIII. LASSO

LASSO is the acronym for **L**east **A**bsolute **S**hrinkage and **S**election **O**perator.

A. Machine Learning Context

The inspiration towards formulating LASSO problem from a machine learning perspective is derived from the following, Consider a machine learning model to have 'm' features and 'n' data points, conventionally we define the terms in use as,

$$X_i = \begin{bmatrix} X_i 1 \\ . \\ . \\ . \\ X_i m \end{bmatrix}, \quad Y_i \in \mathbb{R}; \quad i = 1, ..., n$$

This model produces predictions using on each data point by the function say,

$$h(X_i) = \beta_0 + \beta_1 X_{i1} + ... + \beta_m X_{im}; for \;\; i = 1, ..., n \tag{VIII.1}$$

The Cost function J(β) that helps to improve the predictions made by the model is defined as,

$$J(\beta) = \sum_{i=1}^{n} 1/2(h(X_i) - Y_i)^2 \tag{VIII.2}$$

here the objective is,

$$\text{minimize} \;\; J(\beta) \tag{VIII.3}$$

A well defined model is one which uses only as many as required features to provide accurate predictions, the advantages of it are,

1) The non-relevant features are not required to be measured or taken into consideration for training the model.
2) The possibility of data clutter diminishing the model performance is reduced.

The identification of high impact or significantly contributing features will allow the right selection of such features. When the scale of data is increased it becomes computationally exhausting to individually correlate the features with the result to do the same. Instead it can be achieved by modifying the objective function by adding an additional term like so,

$$f = J(\beta) + \tau \| J(\beta) \|_0 \tag{VIII.4}$$

here $\| J(\beta) \|_0$ is a term that describes the number of non-zero terms and τ is the term that defines the strength of consideration of significant characters, in other words when the value of τ increases then the sparsity of the solution β is increased.

$$\tau \propto sparsity \;\; of \;\; \beta^* \tag{VIII.5}$$

The objective function $f(n)$ from equation 1 is intractable i.e., there exists no efficient algorithms to solve it. On replacing $\|J(\beta)\|_0$ with equivalent L1 norm $\|J(\beta)\|_1$ we obtain,

$$f = J(\beta) + \tau \| J(\beta) \|_1 \tag{VIII.6}$$

here the objective is,

$$\text{minimize} \;\; f \tag{VIII.7}$$

equivalently,

$$\text{minimize } f = \text{minimize } 1/2 \|X\beta - Y\|_2^2 + \tau \|J(\beta)\|_1 \tag{VIII.8}$$

Equation 2 is of the base from and serves as the starting point for the LASSO problem.

B. Solving the LASSO problem

Using a more conventional variable assignment for solving general LASSO problem we can define it as,

$$\text{minimize} \;\; 1/2 \|Ax - b\|_2^2 + \tau \|x\|_1 \tag{VIII.9}$$

$$\equiv \text{minimize} \;\; 1/2 (Ax - b)^T (Ax - b) + \tau \sum_i |x_i| \tag{VIII.10}$$

Representing the same in ADMM format,

$$f(x) = 1/2 \|Ax - b\|_2^2 \;\; \text{and} \;\; g(z) = \tau \|z\|_1 \tag{VIII.11}$$

$$\underset{x,z}{\text{minimize}} f(x) + g(z) \tag{VIII.12}$$

$$\text{subject to, } \; x - z = 0 \tag{VIII.13}$$

Solution,
The corresponding Augmented Lagrangian form can be written as,

$$L_p(x, z, y) = 1/2\|Ax - b\|_2^2 + \tau\|z\|_1 + y^T(x - z) + \rho/2\|x - z\|_2^2 \tag{VIII.14}$$

In order to find the update for x, the minimization of the Lagrangian with respect to x is done as,

Assuming z^k and y^k are known,

$$x^{k+1} = \arg\min_x L_p(x, z^k, y^k) \tag{VIII.15}$$

Consider the x dependent terms,

$$= \arg\min_x \; 1/2\|Ax - b\|_2^2 + y^{kT}(x) + \rho/2\|x - z^k\|_2^2 \tag{VIII.16}$$

Expanding L2 norm term,

$$\arg\min_x \; 1/2(x^T A^T Ax - 2x^T A^T b + b^T b) + y^{kT}(x) + \rho/2\|x - z^k\|_2^2 \tag{VIII.17}$$

Equating first order partial differential term with respect to x to **0** (0 vector) and simplifying to obtain x^{k+1}

$$\frac{\delta L\left(x, z^k, u^k\right)}{\delta x} = \mathbf{0} \tag{VIII.18}$$

$$1/2 * (2A^T Ax - 2A^T b) + y^k + \rho(x - z^k) = \mathbf{0} \tag{VIII.19}$$

$$A^T Ax^{k+1} - A^T b + y^k + \rho(x^{k+1} - z^k) = \mathbf{0} \tag{VIII.20}$$

$$(A^T A + \rho I)x^{k+1} - A^T b = \rho z^k - y^k \tag{VIII.21}$$

$$(A^T A + \rho I)x^{k+1} = A^T b + \rho z^k - y^k \tag{VIII.22}$$

Finally,

$$x^{k+1} = (A^T A + \rho I)^{-1}(A^T b + \rho z^k - y^k) \tag{VIII.23}$$

Similarly, in order to find the update for z, the minimization of the Lagrangian with respect to z is done as,

x^{k+1} has been computed and y^k is assumed to be known,

$$z^{k+1} = \arg\min_z L_p(x^{k+1}, z, y^k) \tag{VIII.24}$$

Consider the z dependent terms,

$$z^{k+1} = \arg\min_z \; \tau\|z\|_1 - y^T z + \rho/2\|x^{k+1} - z\|_2^2 \tag{VIII.25}$$

Pushing the $y^T z$ term into the L2 norm and dividing by τ

$$z^{k+1} = \arg\min_z \; \|z\|_1 + \rho/2\tau\|x^{k+1} - z + (1/\rho)y^k\|_2^2 \tag{VIII.26}$$

grouping non z terms,

$$z^{k+1} = \arg\min_z \; \|z\|_1 + \rho/2\tau\|z - \left(x^{k+1} + (1/\rho)y^k\right)\|_2^2 \tag{VIII.27}$$

bringing ρ into a common denominator constant,

$$z^{k+1} = \arg\min_z \; \|z\|_1 + 1/(2\tau/\rho)\|z - \left(x^{k+1} + (1/\rho)y^k\right)\|_2^2 \tag{VIII.28}$$

Proximal operator : Consider the proximal operator on a variable v with some constant α to be defined as,

$$prox_f(v) = \arg\min_x \left(f(x) + 1/2\alpha\|x - v\|_2^2\right) \equiv S_\alpha(v) \tag{VIII.29}$$

On comparing equations VIII.28 and VIII.29, the similarity of the Right hand sides leads us to simplify the obtained z^{k+1} to be represented in a more compact form like,

$$z^{k+1} = S_{\tau/\rho}(x^{k+1} + y^{k+1}/\rho) \tag{VIII.30}$$

Both x^{k+1} and z^{k+1} have been computed, using them the update for y can be written as,

$$y^{k+1} = y^k + \rho(x^{k+1} - z^{k+1}) \tag{VIII.31}$$

Replacing $(1/\rho)y^k$ with u^k and putting it all together, the updates for each variable is,

$$x^{k+1} = (A^T A + \rho I)^{-1}(A^T b + \rho(z^k - u^k))$$
$$z^{k+1} = S_{\tau/\rho}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + (x^{k+1} - z^{k+1})$$

IX. APPLICATION, RESULTS AND INFERENCE

LAD	Huber	LASSO
$min \ \|z\|_1$ s.t $Ax - b = z$	$\min_{x,z} \sum_{i=1}^{m} huber(z_i)$ $s.t\ Ax - z = b$	$f(x) = \frac{1}{2}\|Ax - b\|_2^2$ and $g(z) = \tau\|z\|_1$ $minimize_{(x,z)}\ f(x) + g(z)$ $s.t\ x - z = \mathbf{0}$
$x^{k+1} := (A^T A)^{-1} A^T (b + z^k - u^k)$ $z^{k+1} = S_{(1/\rho)}(Ax^{k+1} - b + u^k)$ $u^{k+1} = u^k + (Ax^{k+1} - b - z^{k+1})$	$x^{k+1} = (A^T A)^{-1} A^T (b + z^k - u^k)$ $z^{k+1} = \frac{1}{1+\lambda}(Ax^{k+1} - b + u^k) + \frac{\lambda}{1+\lambda} S_\lambda (Ax^{k+1} - b + u^k)$ $u^{k+1} = u^k + (x^{k+1} - u^{k+1})$	$x^{k+1} = (A^T A + \rho I)^{-1}(\rho z^k - y^k + A^T b)$ $z^{k+1} = S_{\frac{\tau}{\rho}}(x^{k+1} + \frac{1}{\rho} y^k)$ $u^{k+1} = u^k + (x^{k+1} - u^{k+1})$

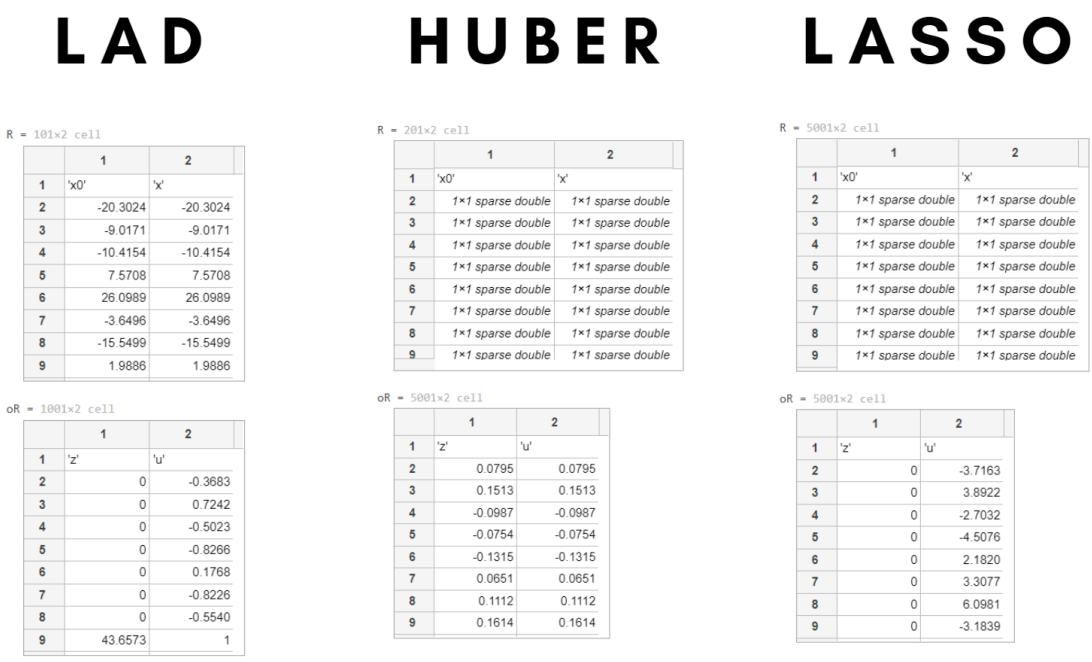
Fig. 3. variable update equations for LAD, Huber and LASSO

Using the final variable update equations formulated and shown in Fig 3 as a guideline, the code for the same has been implemented in matlab and included in section X. The correctness of the optimal solution can be verified as the code has been applied on synthesized data with defined properties, we are creating b as a direct product of A and x referred to as $x_o tp$ which are initialized randomly combined with relatively minimal noise in order to not modify the properties significantly. As convention the initial values of the update variables are initialized randomly, hence, obtaining the value of $x_o tp$ as the final value of x will verify the correctness of the updates. This verification method is applied on each LAD, Huber and LASSO respectively will result in,

From Fig 4 we see that x is equal to x_o in each method. Thereby, the codes are successful in achieving the updates correctly.

The code for implementation of a feature selector using LASSO in python is provided in section X. The goal of the implementation is to verify the practical correctness using LASSO as a selector, this is achieved by first computing the,

1) Non LASSO implementation,
   a) Manually calculating the correlation values of each feature of the data vs to the result.



	1	2
1	'x0'	'x'
2	-20.3024	-20.3024
3	-9.0171	-9.0171
4	-10.4154	-10.4154
5	7.5708	7.5708
6	26.0989	26.0989
7	-3.6496	-3.6496
8	-15.5499	-15.5499
9	1.9886	1.9886

	1	2
1	'x0'	'x'
2	*1×1 sparse double*	*1×1 sparse double*
3	*1×1 sparse double*	*1×1 sparse double*
4	*1×1 sparse double*	*1×1 sparse double*
5	*1×1 sparse double*	*1×1 sparse double*
6	*1×1 sparse double*	*1×1 sparse double*
7	*1×1 sparse double*	*1×1 sparse double*
8	*1×1 sparse double*	*1×1 sparse double*
9	*1×1 sparse double*	*1×1 sparse double*

	1	2
1	'x0'	'x'
2	*1×1 sparse double*	*1×1 sparse double*
3	*1×1 sparse double*	*1×1 sparse double*
4	*1×1 sparse double*	*1×1 sparse double*
5	*1×1 sparse double*	*1×1 sparse double*
6	*1×1 sparse double*	*1×1 sparse double*
7	*1×1 sparse double*	*1×1 sparse double*
8	*1×1 sparse double*	*1×1 sparse double*
9	*1×1 sparse double*	*1×1 sparse double*

	1	2
1	'z'	'u'
2	0	-0.3683
3	0	0.7242
4	0	-0.5023
5	0	-0.8266
6	0	0.1768
7	0	-0.8226
8	0	-0.5540
9	43.6573	1

	1	2
1	'z'	'u'
2	0.0795	0.0795
3	0.1513	0.1513
4	-0.0987	-0.0987
5	-0.0754	-0.0754
6	-0.1315	-0.1315
7	0.0651	0.0651
8	0.1112	0.1112
9	0.1614	0.1614

	1	2
1	'z'	'u'
2	0	-3.7163
3	0	3.8922
4	0	-2.7032
5	0	-4.5076
6	0	2.1820
7	0	3.3077
8	0	6.0981
9	0	-3.1839

Fig. 4. variable update result verification for LAD, Huber and LASSO

b) Take the absolute of the value obtained for each correlation, this is done because high correlation and anti-correlation both are important contributors to the predictions of the model.
c) Sort the absolute correlation features in descending order(Highest to lowest order).

2) LASSO implementation
a) Use the LASSO model to select the important features.
b) Sort the correlations selected features in descending order(Highest to lowest order).

The following results in Fig 5 are two approaches of implementations performed on 4 different data-sets taken from sklearns toy data-sets,

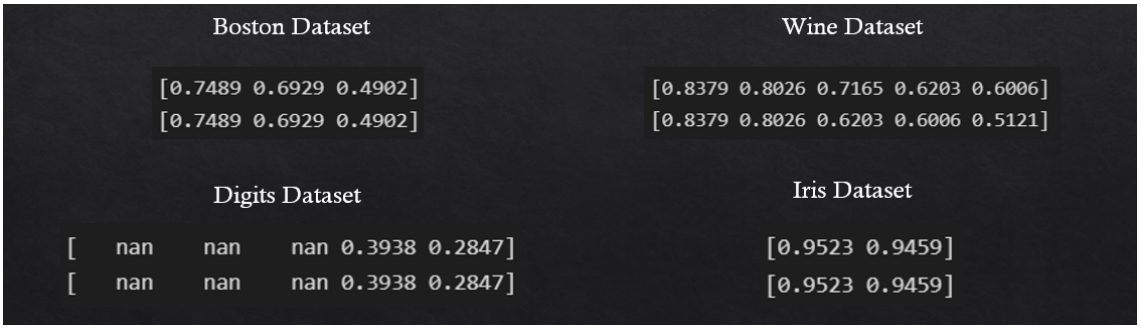


Fig. 5. Non-LASSO vs LASSO implementation results

When manually calculating the highest absolute correlation among all features with respect to the result for feature selection we are considering the most important features. Hence, when both the implementations results are nearly identical from this we can infer, that the LASSO model is indeed correctly selecting the most important features.

X. APPENDIX

The pseudo code for the implementations of LAD, Huber and LASSO are given below,

Algorithm 1: Pseudo code for variable updates in Least Absolute Deviation(LAD)

Require: A and b
Ensure: $A(m \times n), b(m \times 1)$

▷ Randomly initialise to dimension

$x \leftarrow (n \times 1)$
$z \leftarrow (m \times 1)$
$u \leftarrow (m \times 1)$

▷ Assigning required terms

$\rho \leftarrow c_0$
$P \leftarrow (A^T A)^{-1} A^T$

▷ Set maximum iterations

$maxiter \leftarrow c_1$
for $i = 1 \rightarrow maxiter$ **do**
  $x \leftarrow P(b + z - u)$
  $e \leftarrow Ax$
  $z \leftarrow$ **function** Shrinkage$(e - b + u, 1/\rho)$
  $u \leftarrow u + (e - z - b)$
  end
print x, z, u

Algorithm 2: Pseudo code for variable update in Huber fitting

Require: A and b
Ensure: $A(m \times n), b(m \times 1)$

▷ Randomly initialise to dimension

$x \leftarrow (n \times 1)$
$z \leftarrow (m \times 1)$
$u \leftarrow (m \times 1)$

▷ Assigning required terms

$\rho \leftarrow c_0$
$P \leftarrow (A^T A)^{-1}$

▷ Set maximum iterations

$maxiter \leftarrow c_1$
for $i = 1 \rightarrow maxiter$ **do**
  $x \leftarrow P * (A^T(b + z - u))$
  $e \leftarrow Ax$
  $tmp \leftarrow e - b + u$
  $z \leftarrow \rho/(1 + \rho) tmp + 1/(1 + \rho)$**function** shrinkage$(tmp, 1/\rho)$
  $u \leftarrow u + (e - z - b)$
  end
print x, z, u

The matlab codes implementing the pseudo code for LAD is,

```
rand('seed', 0);
randn('seed', 0);

m = 1000; % number of examples
n = 100;  % number of features

% synthesizing required coditions
A = randn(m,n);
x0 = 10*randn(n,1);
b = A*x0;
idx = randsample(m, ceil(m/50));
b(idx) = b(idx) + 1e2*randn(size(idx)); % adding devviations
rho = 1;
```

Algorithm 3: Pseudo code for variable update in LASSO

Require: A and b
Ensure: $A(m \times n), b(m \times 1)$

▷ Randomly initialise to dimension

$x \leftarrow (n \times 1)$
$z \leftarrow (n \times 1)$
$u \leftarrow (n \times 1)$

▷ Assigning required terms

$\lambda\_max \leftarrow$ **function** normalize$(A^T b)$
$\lambda \leftarrow 0.1 \times \lambda\_max$
$\rho \leftarrow c_0$
$P \leftarrow (A^T A + \rho$

▷ Set maximum iterations

$maxiter \leftarrow c_1$
for $i = 1 \rightarrow maxiter$ **do**
  $tmp \leftarrow A^T b + \rho(z - u)$
  $x \leftarrow P \times tmp$
  $z \leftarrow$ **function** shrinkage$(x + u, \lambda/\rho)$
  $u \leftarrow u + (x - z)$
  end
print x, z, u

```
% maximum iterations set instead of convergence
MAX_ITER = 1000;

[m, n] = size(A);
x = zeros(n,1);
z = zeros(m,1);
u = zeros(m,1);
P = inv(A'*A)*A'

for k = 1:MAX_ITER
    % x-update
    x = P*(b + z - u);

    e = A*x;

    % z-update
    z = shrinkage(e - b + u, 1/rho);
    % u-update
    u = u + (e - z - b);
end

x
z
u
```

The matlab codes implementing the pseudo code for Huber Fitting is,

```
randn('seed', 0);
rand('seed',0);

m = 5000;  % number of examples
n = 200;   % number of features
```

```
% synthesizing required coditions
x0 = sprandn(n,1,rho);
A = randn(m,n);
A = A*spdiags(1./norms(A)',0,n,n); % normalize columns
b = A*x0 + sqrt(0.01)*randn(m,1);
b = b + 10*sprand(m,1,200/m);       % adding sparse, large noise
rho = 1;

MAX_ITER = 1000;
[m, n] = size(A);

% save a matrix-vector multiply
Atb = A'*b;
x = zeros(n,1);
z = zeros(m,1);
u = zeros(m,1);
P= inv(A'*A)

for k = 1:MAX_ITER
    % x-update
    x_new = P*(A'*(b+z-u));

    e   =  A*x_new;
    tmp = e - b + u;

    % z - update
    z_new = rho/(1 + rho)*tmp + 1/(1 + rho)*shrinkage(tmp,1/rho);
    % u - update
    u_new = u + (e - z_new - b);
end

x_new
z_new
u_new
```

The matlab codes implementing the pseudo code for LASSO is,

```
randn('seed', 0);
rand('seed',0);

m = 1500;        % number of examples
n = 5000;        % number of features
rho = 100/n;       % sparsity density

% synthesizing required coditions
x0 = sprandn(n,1,rho);
A = randn(m,n);
A = A*spdiags(1./sqrt(sum(A.^2))',0,n,n); % normalize columns
b = A*x0 + sqrt(0.001)*randn(m,1);          % adding devviations

lambda_max = norm( A'*b, 'inf' );
lambda = 0.1*lambda_max;

% maximum iterations set instead of convergence
MAX_ITER = 1000;
```

```
[m, n] = size(A);

% save a matrix-vector multiply
P = inv(A'*A+rho*eye(n))

x = zeros(n,1);
z = zeros(n,1);
u = zeros(n,1);

for k = 1:MAX_ITER
    tmp = A'*b + rho*(z - u);     % temporary value

    % x-update
    x = P*tmp;
    % z-update
    z = shrinkage(x + u, lambda/rho);
    % u-update
    u = u + (x - z);
end

x
z
u
```

The python code that produces the Loss vs Error plot of L1-norm and L2-norm (Fig 1) is,

```
import numpy as np
import matplotlib.pyplot as plt

def SE(x,y,intc,beta):
    return (1./len(x))*(0.5)*sum(y - beta * x - intc)**2

def L1(intc,beta,lam):
    return lam*(np.abs(intc)+np.abs(beta))

def L2(intc,beta,lam):
    return lam*(intc**2 + beta**2)

N = 100
x = np.random.randn(N)
y = 2 * x + np.random.randn(N)

beta_N = 10000
beta = np.linspace(-1000,1000,beta_N)
intc = 0.0

L1_array = np.array([L1(intc,i,lam=30) for i in beta])
L2_array = np.array([L2(intc,i,lam=1) for i in beta])

fig1 = plt.figure()
ax1 = fig1.add_subplot(1,1,1)
plt.ylabel("Loss")
plt.xlabel("Error")
plt.grid()
ax1.plot(beta,L1_array,label='L1 norm')
ax1.plot(beta,L2_array,label='L2 norm')
```

```
plt.title('The graph of each term of Loss function')
plt.legend()
fig1.show()
```

The matlab code that produces the Loss vs Error plot of LAD, LS and Huber(Fig 2) is,

```
x = linspace(-2,2,100);

LAD = abs(x)

LS  = power(x,2)

delta = 1
set_1 = abs(x) <= delta;
set_2 = abs(x) > delta;

Huber = 0.5*set_1.*power(x,2) + set_2.*(abs(x) - 0.5)

hold on
grid on
xlim([-2,2]);
xticks(-2:0.5:2);
ylim([0,4]);
yticks(0:0.5:4);

% Error = actual - predicted
xlabel("yp - y")
ylabel("Loss")
plot(x, LAD)
plot(x, LS)
plot(x, Huber)
legend('Absolute', 'Squared', 'Huber (delta=1)')
```

The python code for implementing an inbuilt LASSO based Feature selector is,

```
import numpy as np
import scipy.stats
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Lasso

# importing boston dataset
from sklearn.datasets import load_boston
X,y = load_boston(return_X_y=True)
features = load_boston()['feature_names']

# importing wine dataset
from sklearn.datasets import load_wine
X,y = load_wine(return_X_y=True)
features = load_wine()['feature_names']

# importing iris dataset
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y=True)
```

```
features = load_iris()['feature_names']

# importing digits dataset
from sklearn.datasets import load_digits
X,y = load_digits(return_X_y=True)
features = load_digits()['feature_names']

# data split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, ...
random_state=42)

# using pipeline to define the Lasso model and use it to scale the data prior to ...
selecting
pipeline = Pipeline([
                    ('scaler',StandardScaler()),
                    ('model',Lasso())
])

# using 5 fold mean cross validation across 0.1 to 10 with a step size of 0.1
# this is done to identify a best fit value for alpha
# neg_mean_squared_error because the grid search tries to maximize the performance ...
metrics
search = GridSearchCV(pipeline,
                      {'model__alpha':np.arange(0.1,10,0.1)},
                      cv = 5, scoring="neg_mean_squared_error",verbose=3
                      )

# training and extracting model info
search.fit(X_train,y_train)

best_alhpa = search.best_params_
print("Best value for alpha is =", '%.6f'%float(best_alhpa['model__alpha']))

coefficients = search.best_estimator_.named_steps['model'].coef_
importance = np.abs(coefficients)
print(importance)

# assigning default threshold value
threshold = 0

# using threshold to select the features whos importance was decide by Lasso
np.array(features)[importance > threshold]
np.array(features)[importance <= threshold]



# perfoming mannual correlation on data
r_col = y_train
n_ft = len(X_train[0])
n_cof = [None]*n_ft

for i in range(n_ft):
    i_col = df = X_train[:,i]
    n_cof[i] = scipy.stats.pearsonr(r_col, i_col)[0]

importance_corr = np.abs(np.round(n_cof,4))

print(importance_corr)
```

```
# comaring the results side by side
imp_arr = importance_corr*[importance > threshold]
rel_imp = imp_arr[np.nonzero(imp_arr)]

print(np.sort(importance_corr)[::-1][:n_imp])

print(np.sort(rel_imp)[::-1][:n_imp])
```

References

[1] Stephen Boyd et al. “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”. In: *Foundations and Trends in Machine Learning* 3 (Jan. 2011), pp. 1–122. DOI: 10.1561/2200000016.