Proximal Gradient Descent for the LASSO Problem

Sam Brown Henry Finnila Ryan Voss

June 9, 2025

Contents

Introduction

Introduction - LASSO Regression

LASSO (Least Absolute Shrinkage and Selection Operator)

- Uses L1 regularization
- Prevents overfitting by shrinking less explanatory coefficients to zero
- Useful for variable selection and model interpretability
- Choose λ through cross-validation
  - Minimize MSE

$$\arg\min_{\beta} \left\{ \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1 \right\}, \qquad \beta \in \mathbb{R}^p \tag{1}$$

LASSO Applications

- Gene selection in high-dimensional microarray and RNA-seq studies [2].
- Feature selection for macroeconomic and financial time series forecasting [3].
- Sparse signal reconstruction in compressed sensing and image processing [4].
- Variable selection in marketing analytics for customer segmentation and churn prediction [5].
- Predictor identification in environmental and climate modeling [6].

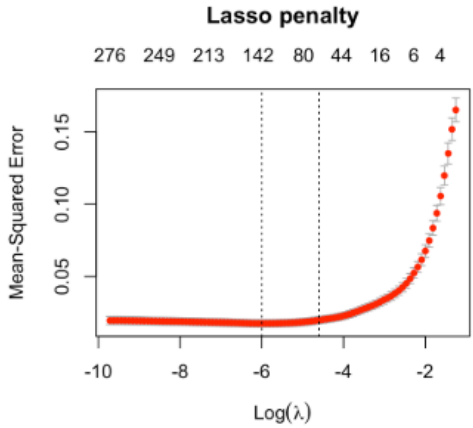


Figure: Minimizing MSE with choice of λ

Ridge vs. LASSO

Ridge Regression:

$$\arg\min_{\beta} \left\{ \tfrac{1}{2}\|y - X\beta\|_2^2 + \tfrac{\lambda}{2}\|\beta\|_2^2 \right\} \quad (2)$$

$$\beta = (X^T X + \lambda I)^{-1} X^T y \quad (3)$$

- Can only shrink coefficients asymptotically close to zero
- Better if multicollinearity is present

LASSO Regression:

$$\arg\min_{\beta} \left\{ \tfrac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1 \right\} \quad (4)$$

- Exact zeros arise from the subgradient interval at zero
- This lets LASSO completely remove certain predictors from a model

Challenges with Optimization

In the LASSO formula, decomposing

$$\arg\min_{\beta} \ \frac{1}{2}\|y - X\beta\|_2^2 + \ \lambda\,\|\beta\|_1.$$

into functions

$$g(x) = \frac{1}{2}\|y - X\beta\|_2^2$$

and

$$h(x) = \ \lambda\,\|\beta\|_1$$

- L1 penalty, $h(x)$, is not differentiable
- Standard gradient descent fails
- Need a new method...

Proposed Method

Proximal/Generalized Gradient Descent

Consider $f : \mathbb{R}^p \to \mathbb{R}$ that is not necessarily differentiable, but can be decomposed

$$f(x) = g(x) + h(x)$$

- g convex and differentiable
- h convex

LASSO objective can easily be decomposed in this manner.

$$\underbrace{\frac{1}{2}\|y - X\beta\|_2^2}_{g(x)} + \underbrace{\lambda\|\beta\|_1}_{h(x)}$$

Derivation (1/2)

Recall gradient descent update with step size α for differentiable g :

$$x_{k+1} = x_k - \alpha \nabla g(x_k)$$

Comes from quadratic approximation

$$x_{k+1} = \arg\min_z \left(g(x_k) + \nabla g(x_k)^\top (z - x_k) + \frac{1}{2\alpha} \|z - x_k\|_2^2 \right)$$

Applying this approximation to g gives PGD update

$$x_{k+1} = \arg\min_z \left(g(x_k) + \nabla g(x_k)^\top (z - x_k) + \frac{1}{2\alpha} \|z - x_k\|_2^2 + h(z) \right)$$

Derivation (2/2)

Equivalently,

$$x_{k+1} = \arg\min_z \left(\frac{1}{2\alpha} \|z - (x_k - \alpha \nabla g(x_k))\|_2^2 + h(z) \right)$$

- Remain close to GD update for g
- Enforce small h

The Proximal Operator

Define $\text{prox}_{\alpha,h} : \mathbb{R}^p \to \mathbb{R}^p$ by

$$\text{prox}_{\alpha,h}(x) = \arg\min_{z \in \mathbb{R}^p} \left(\frac{1}{2\alpha} \|z - x\|_2^2 + h(z) \right).$$

Note that proximal mapping depends on α and h but not g. Our update,

$$x_{k+1} = \arg\min_{z} \left(\frac{1}{2\alpha} \|z - (x_k - \alpha \nabla g(x_k))\|_2^2 + h(z) \right),$$

becomes

$$x_{k+1} = \text{prox}_{\alpha,h}(x_k - \alpha \nabla g(x_k)).$$

How is This Useful?

$$x_{k+1} = \text{prox}_{\alpha,h}(x_k - \alpha \nabla g(x_k))$$

$$\text{prox}_{\alpha,h}(x) = \arg\min_z \left(\frac{1}{2\alpha} \|z - x\|_2^2 + h(z) \right)$$

- Goal: Solve a minimization problem
- Proposed method: Solve a minimization problem on every iteration?
- Need an easy way to evaluate $\text{prox}_{\alpha,h}$
- Possible for many important h functions, including L_1 regularization (LASSO objective)

Proximal Mapping for the LASSO Problem

For the LASSO objective,

$$h(\beta) = \lambda \|\beta\|_1.$$

In this case,

$$\text{prox}_{\alpha,h}(\beta) = S_{\alpha\lambda}(\beta),$$

where $S_{\alpha\lambda} : \mathbb{R}^p \to \mathbb{R}^p$ is the soft thresholding operator defined element wise by

$$[S_{\alpha\lambda}(\beta)]_i = \begin{cases} \beta_i + \alpha\lambda & \text{if } \beta_i < -\alpha\lambda \\ 0 & \text{if } -\alpha\lambda < \beta_i < \alpha\lambda \\ \beta_i - \alpha\lambda & \text{if } \alpha\lambda < \beta_i, \end{cases}$$

for $i = 1, \cdots p$. This allows us to use proximal gradient descent for the LASSO problem. The resulting algorithm is often called the Iterative Soft Thresholding Algorithm (ISTA).

Convergence Guarantees

Theorem

Suppose that

- $f : \mathbb{R}^p \to \mathbb{R}$ can be decomposed into $f(x) = h(x) + g(x)$
- g is convex, differentiable, and L-smooth
- h is convex
- $\text{prox}_{\alpha,h}$ can be evaluated.

Then, proximal gradient descent with step size $0 < \alpha \leq \frac{1}{L}$ satisfies

$$|f(x_k) - f(x^*)| \leq \frac{\|x_0 - x^*\|_2^2}{2\alpha k}.$$

Remarks:

- Convergence is $\mathcal{O}(1/k)$
- Same convergence rate as gradient descent, but we are just counting iterations

Implementation

Implementation

To demonstrate the implementation of proximal gradient descent, we will solve the well-known LASSO (Least Absolute Shrinkage and Selection Operator) problem. As previously discussed, LASSO requires solving the following optimization problem:

$$\arg\min_{\beta \in \mathbb{R}^p} \left\{ \frac{1}{2} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\}$$

This problem is not differentiable due to the ℓ_1-norm term $\lambda\|\beta\|_1$, and therefore standard gradient descent cannot be directly applied. Instead, we use **proximal gradient descent**, which is specifically designed to handle non-smooth regularization terms.

We implement this optimization algorithm in `Python` using the `PyTorch` library.

Data Initialization

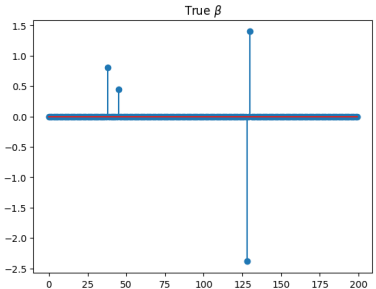
For this implementation, we use synthetic data to keep things simple and easy to control. The code below creates a high-dimensional dataset with a sparse true coefficient vector `betaTrue`, which works well for testing Lasso regression.

Listing 1: Synthetic data creation

```
import torch
import matplotlib.pyplot as plt
torch.manual_seed(1)
# synthetic data
N = 40      # training examples
d = 200     # number of features
nnz = 4     # number of nonzero components
m = torch.randperm(d) # random permutation of indices
betaTrue = torch.zeros(d) # intitalize beta vector
betaTrue[m[:nnz]] = 5 * torch.randn(nnz) # assign 4 random values
X = torch.randn(N, d) # feature matrix
ones = torch.ones(X.size(0), 1) # for intercept (if necessary)
X_new = torch.cat((ones, X), dim=1) # combine intercept column with X
noise = 0.1 * torch.randn(N)
y = X @ betaTrue + noise
```

Motivation for Using Lasso

- Lasso fits our setup with only a few nonzero coefficients.
- `nnz` sets how many predictors matter; noise is added to y.
- Plotting β_{True} shows the sparsity Lasso aims to recover.



Defining the Proximal Operator

To solve the Lasso problem, we must handle the non-differentiable ℓ_1 penalty term. This is done using the **proximal operator** for the L1 norm, which shrinks each component of a vector toward zero by a fixed amount, promoting sparsity.

We implement this using a `soft_threshold` function, which moves each value closer to zero by `alpha`, and sets small values exactly to zero. The `proxL1Norm` function applies this elementwise and optionally skips the first entry (commonly used for intercept terms).

The operator corresponds to the soft-thresholding function:

$$[S_{\alpha\lambda}(\beta)]_i = \begin{cases} \beta_i + \alpha\lambda & \text{if } \beta_i < -\alpha\lambda \\ 0 & \text{if } -\alpha\lambda < \beta_i < \alpha\lambda \\ \beta_i - \alpha\lambda & \text{if } \beta_i > \alpha\lambda \end{cases}$$

Proximal Operator Implementation

The code below implements the soft-thresholding operator using PyTorch's `clamp` function to enforce sparsity in Lasso regression.

```python
def soft_threshold(beta, alpha):
    # shrinks each value toward zero
    return torch.sign(beta) * torch.clamp(torch.abs(beta) - alpha, min=0.0)

def proxL1Norm(betaHat, alpha, penalizeALL=True):
    # applies soft-thresholding; optionally skips the intercept
    out = soft_threshold(betaHat, alpha)
    if not penalizeALL:
        out[0] = betaHat[0]
    return out
```

This operation matches the mathematical definition of $S_{\alpha\lambda}(\beta)$ and is key to enabling Lasso's sparse recovery.

Proximal Gradient Descent Implementation

```python
def solveLasso_proxGrad(X, y, lmbda):
    maxIter = 300
    alpha = 0.005 # step size

    N, d = X.shape # number of samples, number of features
    beta = torch.zeros(d, dtype=torch.float32) # initialize
    costFunVals = torch.zeros(maxIter) # intitialize
    for t in range(maxIter):
        grad = X.T @ (X @ beta - y) # compute gradient of differentiable term
        beta = proxL1Norm(beta - alpha * grad, alpha * lmbda) # gradient step
            then apply prox

        residual = X @ beta - y # compute and store residual
        cost = 0.5 * torch.norm(residual)**2 + lmbda * torch.sum(torch.abs(beta)
            )
        costFunVals[t] = cost

        print(f"Iteration: {t}, Objective function value: {cost.item():.4f}")
    return beta, costFunVals
```

Proximal Gradient Update Step

The core of the `solveLasso_proxGrad` function is the following line:

```
beta = proxL1Norm(beta - alpha * grad, alpha * lmbda)
```

This line performs the proximal gradient update:

- First, it takes a gradient step on the smooth squared loss.
- Then, it applies the proximal operator to handle the non-smooth ℓ_1 penalty.

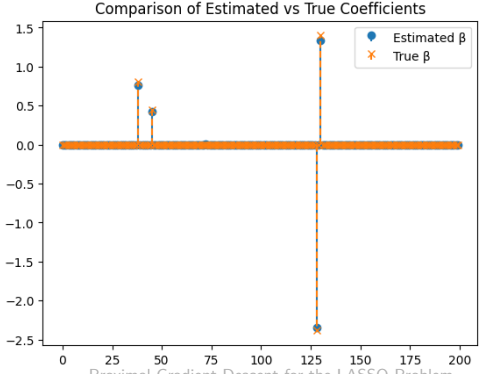
$$\beta^{(k+1)} = \text{prox}_{\alpha,h}\left(\beta^{(k)} - \alpha_k \nabla f\left(\beta^{(k)}\right)\right)$$

The second argument in `proxL1Norm` is the scaled regularization parameter $\alpha \cdot \lambda$. As shown earlier, this causes each component of β to be shifted toward zero by this exact amount, as defined by the soft-thresholding operator.

Testing the Implementation

To test our proximal gradient descent code, we call the function with our feature matrix, target vector, and a specified λ value:

```
lmbda = 2
beta, costFunVals = solveLasso_proxGrad(X, y, lmbda)
```



Lasso with scikit-learn (1/2)

- We now use the built-in Lasso class from scikit-learn.
- This is more efficient and comes with tools like preprocessing and cross-validation.
- We use the california_housing dataset, which is convenient and well-supported.

Import libraries and prepare data:

```python
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load and split data
X, y = fetch_california_housing(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Lasso with scikit-learn (2/2)

- We initialize the Lasso model and train it on the scaled training data.
- Then, we evaluate it on the test set using Mean Squared Error (MSE).

Model training and evaluation:

```
# Initialize and train Lasso model
lasso = Lasso(alpha=1.0)
lasso.fit(X_train, y_train)

# Predict and evaluate
y_pred = lasso.predict(X_test)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

The MSE gives a quantitative measure of the model's prediction error on unseen data.

Discussion

Discussion of our Algorithm

Strengths

- **Sparsity promotion.** The soft-thresholding operator can set coefficients to zero
- **Simplicity and ease of implementation.**
- **Guaranteed convergence.** Under standard assumptions ISTA converges at a $\mathcal{O}(1/k)$ rate .

Weaknesses

- **Slow convergence.** The $\mathcal{O}(1/k)$ rate can be slow in practice
- **Step-size sensitivity.** Tuning the parameters it one of the most crucial, yet difficult parts of the process.

Thank you!

References

References


R. Tibshirani, Convex Optimization: Proximal Gradient Methods, Lecture 8, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2015. [Online]. Available: https://www.stat.cmu.edu/ ryantibs/convexopt-S15/lectures/08-prox-grad.pdf


F. Hu, R. Zhao, and J. Xu, "Supervised group Lasso with applications to microarray data analysis," *BMC Bioinformatics*, vol. 9, p. 10, 2008.


A. Bonavito and L. Morelli, "Forecasting macroeconomic time series: Lasso-based approaches," *Econometric Reviews*, vol. 33, no. 5–6, pp. 594–616, 2014.


D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.


L. Wang, H. Li, and M. Zhang, "Customer churn prediction based on Lasso and random forest models," in *Proc. 2018 IEEE Int. Conf. Cloud Comput. Big Data Anal.*, Chengdu, China, 2018, pp. 231–235.


P. Liang, K. Fraedrich, and R. Schnur, "Predictor selection for downscaling GCM data with Lasso," *J. Geophys. Res. Atmos.*, vol. 117, no. D11, 2012.