Introduction to Machine Learning Problems: LASSO and Model Selection Solutions

Willem Neuefeind Lessig
Completed: October 1, 2025

1. *Exhaustive search.* In this problem, we will look at how to exhaustively search over all possible subsets of features. You are given three python functions:

```
model = LinearRegression()  # Create a linear regression model object
model.fit(X,y)              # Fits the model
yhat = model.predict(X)     # Predicts targets given features
```

Given training data Xtr,ytr and test data Xts,yts, write a few lines of python code to:

(a) Find the best model using only one feature of the data (i.e. one column of Xtr and Xts).
Solution:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

best_mse = float('inf')
best_feature = None

for i in range(Xtr.shape[1]):
    model = LinearRegression()
    model.fit(Xtr[:, i:i+1], ytr)
    yhat = model.predict(Xts[:, i:i+1])
    mse = mean_squared_error(yts, yhat)

    if mse < best_mse:
        best_mse = mse
        best_feature = i

print(f"Best single feature: {best_feature}, MSE: {best_mse}")
```

(b) Find the best model using only two features of the data (i.e. two columns of Xtr and Xts).
Solution:

```
from itertools import combinations
```

```
best_mse = float('inf')
best_features = None

for feature_pair in combinations(range(Xtr.shape[1]), 2):
    model = LinearRegression()
    model.fit(Xtr[:, feature_pair], ytr)
    yhat = model.predict(Xts[:, feature_pair])
    mse = mean_squared_error(yts, yhat)

    if mse < best_mse:
        best_mse = mse
        best_features = feature_pair

print(f"Best two features: {best_features}, MSE: {best_mse}")
```

(c) Suppose we wish to find the best k of p features via exhaustive searching over all possible subsets of features. How many times would you need to call the fit function? What if $k = 10$ and $p = 1000$?

Solution: The number of times we need to call the fit function is $\binom{p}{k} = \frac{p!}{k!(p-k)!}$.

For $k = 10$ and $p = 1000$:

$$\binom{1000}{10} = \frac{1000!}{10! \cdot 990!} \approx 2.63 \times 10^{23}$$

This is computationally infeasible, which is why we need regularization methods like LASSO.

2. *Selecting a regularizer.* Suppose we fit a regularized least squares objective,

$$J(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \phi(\mathbf{w}),$$

where $\hat{y}_i$ is some prediction of y_i given the model parameters $\mathbf{w}$. For each case below, suggest a possible regularization function $\phi(\mathbf{w})$. There is no single correct answer.

Solutions:

(a) All parameters vectors $\mathbf{w}$ should be considered.
$\phi(\mathbf{w}) = 0$ (no regularization)

(b) Negative values of w_j are unlikely (but still possible).
$\phi(\mathbf{w}) = \sum_j \max(0, -w_j)$ or $\phi(\mathbf{w}) = \sum_j e^{-w_j}$

(c) For each j, w_j should not change that significantly from w_{j-1}.
$\phi(\mathbf{w}) = \sum_{j=2}^{p} (w_j - w_{j-1})^2$ (total variation regularization)

(d) For most j, $w_j = w_{j-1}$. However, it can happen that w_j can be different from w_{j-1} for a few indices j.
$\phi(\mathbf{w}) = \sum_{j=2}^{p} |w_j - w_{j-1}|$ (fused LASSO)

Variable	Units	Mean	Std dev
Median income, x_1	\$	50000	15000
Median age, x_2	years	45	10
House sale price, y	\$1000	300	100

Table 1: Features for Problem 3

3. *Normalization.* A data analyst for a real estate firm wants to predict house prices based on two features in each zip code. The features are shown in Table 1. The agent decides to use a linear model,

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2, \tag{1}$$

(a) What is the problem in using a LASSO regularizer of the form,

$$\phi(\boldsymbol{\beta}) = \sum_{j=1}^{2} |\beta_j|.$$

Solution: The problem is that the features have very different scales. Income is measured in dollars (mean 50,000) while age is measured in years (mean 45). This means β_1 will be much smaller than β_2 to achieve similar impact on the prediction. The LASSO penalty $|\beta_1| + |\beta_2|$ will unfairly penalize the coefficient for income more heavily, even though both features might be equally important.

(b) To uniformly regularize the features, she fits a model on the normalized features,

$$\hat{u} = \alpha_1 z_1 + \alpha_2 z_2, \quad z_j = \frac{x_j - \bar{x}_j}{s_j}, \quad u = \frac{\hat{y} - \bar{y}}{s_y},$$

where s_j and s_y are the standard deviations of the x_j and y. She obtains parameters $\boldsymbol{\alpha} = [0.6, -0.3]$? What are the parameters β in the original model (1)?

Solution: We need to transform back from the normalized model to the original model.

Starting with: $\hat{u} = \alpha_1 z_1 + \alpha_2 z_2$

Substituting the definitions:

$$\frac{\hat{y} - \bar{y}}{s_y} = \alpha_1 \frac{x_1 - \bar{x}_1}{s_1} + \alpha_2 \frac{x_2 - \bar{x}_2}{s_2}$$

Solving for $\hat{y}$:

$$\hat{y} = \bar{y} + s_y \left(\alpha_1 \frac{x_1 - \bar{x}_1}{s_1} + \alpha_2 \frac{x_2 - \bar{x}_2}{s_2} \right)$$

$$\hat{y} = \bar{y} - s_y \left(\alpha_1 \frac{\bar{x}_1}{s_1} + \alpha_2 \frac{\bar{x}_2}{s_2} \right) + s_y \frac{\alpha_1}{s_1} x_1 + s_y \frac{\alpha_2}{s_2} x_2$$

Therefore:

$$\beta_0 = \bar{y} - s_y \left(\alpha_1 \frac{\bar{x}_1}{s_1} + \alpha_2 \frac{\bar{x}_2}{s_2} \right) \tag{2}$$

$$\beta_1 = s_y \frac{\alpha_1}{s_1} = 100 \times \frac{0.6}{15000} = 0.004 \tag{3}$$

$$\beta_2 = s_y \frac{\alpha_2}{s_2} = 100 \times \frac{-0.3}{10} = -3 \tag{4}$$

$$\beta_0 = 300 - 100 \left(0.6 \times \frac{50000}{15000} + (-0.3) \times \frac{45}{10} \right) = 300 - 100(2 - 1.35) = 235$$

So $\boldsymbol{\beta} = [235, 0.004, -3]$.

4. *Normalization in python.* You are given python functions,

```
model = SomeModel()          # Creates a model
model.fit(Z,u)               # Fits the model, expecting normalized features
yhat = model.predict(Z)      # Predicts targets given features
```

Given training data Xtr,ytr and test data Xts,yts, write python code to:

- Normalize the training data to remove the mean and standard deviation from both Xtr and ytr.
- Fit the model on the normalized data.
- Predict the values yhat on the test data.
- Measure the RSS on the test data.

Solution:

```
import numpy as np

# Normalize training data
Xtr_mean = np.mean(Xtr, axis=0)
Xtr_std = np.std(Xtr, axis=0)
ytr_mean = np.mean(ytr)
ytr_std = np.std(ytr)

Ztr = (Xtr - Xtr_mean) / Xtr_std
utr = (ytr - ytr_mean) / ytr_std

# Fit the model on normalized data
model = SomeModel()
model.fit(Ztr, utr)

# Normalize test data using training statistics
Zts = (Xts - Xtr_mean) / Xtr_std
```

```
# Predict on normalized test data
u_pred = model.predict(Zts)

# Transform predictions back to original scale
yhat = u_pred * ytr_std + ytr_mean

# Measure RSS on test data
rss = np.sum((yts - yhat)**2)
print(f"RSS on test data: {rss}")
```

5. *Discretization.* Suppose we wish to fit a model,

$$y \approx \hat{y} = \sum_{j=1}^{K} \beta_j e^{-\alpha_j x}, \tag{5}$$

for parameters α_j and β_j. Since the parameters α_j are not known, this model is nonlinear and cannot be fit with least squares. A common approach in such circumstances is to use an alternate linear model,

$$y \approx \hat{y} = \sum_{j=1}^{p} \tilde{\beta}_j e^{-\tilde{\alpha}_j x}, \tag{6}$$

where the values $\tilde{\alpha}_1, \ldots, \tilde{\alpha}_p$ are a *fixed*, large set of possible values for α_j, and $\tilde{\beta}_j$ are the coefficients in the model. Since the values $\tilde{\alpha}_j$ are fixed, only the parameters $\tilde{\beta}_j$ need to be learned. Hence, the model (6) is linear. The model (6) is equivalent to (5) if only a small number K of the coefficients $\tilde{\beta}_j$ are non-zero. You are given three python functions:

```
model = Lasso(lam=lam)          # Creates a linear LASSO model
                                # with a regularization lam
beta = model.fit(Z,y)           # Finds the model parameters using the
                                # LASSO objective
                                #   ||y-Z*beta||^2 + lam*||beta||_1
yhat = model.predict(Z)         # Predicts targets given features Z:
                                #    yhat = Z*beta
```

Note this syntax is slightly different from the sklearn syntax. You are also given training data xtr,ytr and test data xts,yts. Write python code to:

- Create $p = 100$ values of $\tilde{\alpha}_j$ uniformly in some interval $\tilde{\alpha}_j \in [a, b]$ where a and b are given.
- Fit the linear model (6) on the training data for some given lam.
- Measure the test error.
- Find coefficients α_j and β_j corresponding to the largest $k = 3$ values in $\tilde{\beta}_j$. You can use the function np.argsort.

Solution:

```
import numpy as np

# Create p=100 values of alpha uniformly in [a,b]
p = 100
alpha_tilde = np.linspace(a, b, p)

# Create feature matrix Z for training data
Ztr = np.zeros((len(xtr), p))
for j in range(p):
    Ztr[:, j] = np.exp(-alpha_tilde[j] * xtr)

# Create feature matrix Z for test data
Zts = np.zeros((len(xts), p))
for j in range(p):
    Zts[:, j] = np.exp(-alpha_tilde[j] * xts)

# Fit LASSO model
model = Lasso(lam=lam)
beta_tilde = model.fit(Ztr, ytr)

# Predict on test data
yhat_ts = model.predict(Zts)

# Measure test error (MSE)
test_error = np.mean((yts - yhat_ts)**2)
print(f"Test error: {test_error}")

# Find coefficients corresponding to largest k=3 values
k = 3
largest_indices = np.argsort(np.abs(beta_tilde))[-k:]

# Extract the corresponding alpha and beta values
alpha_j = alpha_tilde[largest_indices]
beta_j = beta_tilde[largest_indices]

print(f"Selected alpha values: {alpha_j}")
print(f"Selected beta values: {beta_j}")
```

6. *Minimizing an ℓ_1 objective.* In this problem, we will show how to minimize a simple scalar function with an ℓ_1-term. Given y and $\lambda > 0$, suppose we wish to find the minimum,

$$\widehat{w} = \arg\min_w J(w) = \frac{1}{2}(y - w)^2 + \lambda|w|.$$

Write $\widehat{w}$ in terms of y and λ. Since $|w|$ is not differentiable everywhere, you cannot simple set $J'(w) = 0$ and solve for w. Instead, you have to look at three cases:

(i) First, suppose there is a minima at $w > 0$. In this region, $|w| = w$. Since the set $w > 0$ is open, at any minima $J'(w) = 0$. Solve for w and test if the solution indeed satisfies $w > 0$.

(ii) Similarly, suppose $w < 0$. Solve for $J'(w) = 0$ and test if the solution satisfies the assumption that $w < 0$.

(iii) If neither of the above cases have a minima, then the minima must be at $w = 0$.

Solution:

Case (i): $w > 0$ For $w > 0$, we have $|w| = w$, so:

$$J(w) = \frac{1}{2}(y - w)^2 + \lambda w$$

$$J'(w) = -(y - w) + \lambda = -y + w + \lambda$$

Setting $J'(w) = 0$:

$$w = y - \lambda$$

This solution is valid if $w > 0$, which means $y - \lambda > 0$, or $y > \lambda$.

Case (ii): $w < 0$ For $w < 0$, we have $|w| = -w$, so:

$$J(w) = \frac{1}{2}(y - w)^2 + \lambda(-w) = \frac{1}{2}(y - w)^2 - \lambda w$$

$$J'(w) = -(y - w) - \lambda = -y + w - \lambda$$

Setting $J'(w) = 0$:

$$w = y + \lambda$$

This solution is valid if $w < 0$, which means $y + \lambda < 0$, or $y < -\lambda$.

Case (iii): $w = 0$ If neither case (i) nor case (ii) applies, then the minimum is at $w = 0$. This occurs when $-\lambda \leq y \leq \lambda$.

Final Solution:

$$\widehat{w} = \begin{cases} y - \lambda & \text{if } y > \lambda \\ 0 & \text{if } -\lambda \leq y \leq \lambda \\ y + \lambda & \text{if } y < -\lambda \end{cases}$$

This can also be written as the soft-thresholding function:

$$\widehat{w} = \text{sign}(y) \max(0, |y| - \lambda)$$