Introduction to Machine Learning Problems: LASSO and Model Selection

Prof. Sundeep Rangan

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`).

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

(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

(a) For this problem, we simply loop through the features:

```
p = X.shape[1]  # Number of features
for i in range(p):

    # Fit the model on the training data
    model = LinearRegression()
    model.fit(Xtr[:,i])

    # Test the model
    yhat = model.predict(Xts[:,i])

    # Score the prediction
    mse[i] = np.mean((yhat-yts)**2)

iopt = np.argmin(mse)
```

(b) In this case, we loop through all pairs of features:

```
p = X.shape[1]  # Number of features
```

```
Isel_list = []  # List of sets of features tested
mse = []        # MSE for each set
for i0 in range(p-1):
    for i1 in range(i0,p):
        # Record which are the features being used
        Isel = [i0, i1]
        Isel_list.append(Isel)

        # Fit the model on the training data
        model = LinearRegression()
        model.fit(Xtr[:,Isel])

        # Test the model
        yhat = model.predict(Xts[:,Isel])

        # Score the prediction
        mse.append( np.mean((yhat-yts)**2) )

iopt = np.argmin(mse)
Isel_opt = Isel[iopt]
```

(c) You would need to search over $\binom{p}{k}$ subsets of features. If $p = 1000$ and $k = 10$, $\binom{p}{k} \approx 2.63(10)^{23}$. Exhaustive search is obviously not possible in this case.

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.

(a) All parameters vectors $\mathbf{w}$ should be considered.

(b) Negative values of w_j are unlikely (but still possible).

(c) For each j, w_j should not change that significantly from w_{j-1}.

(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.

Solution

(a) Do not use regularization, so set $\phi(\mathbf{w}) = 0$.

(b) You can use any function that penalizes negative values. For example:

$$\phi(\mathbf{w}) = \sum_j \phi_j(x_j), \quad \phi_j(x_j) = \begin{cases} w_j^2 & \text{if } w_j < 0 \\ 0 & \text{if } w_j \geq 0. \end{cases}$$

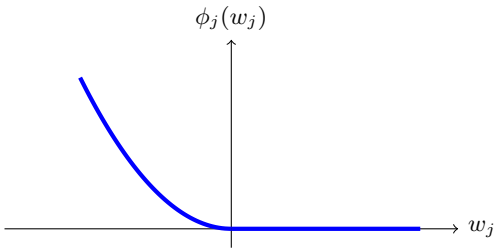

Figure 1: Regularizer to penalize negative w_j.

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

In this way, the cost is penalized for all components where $w_j < 0$. The function is shown in Fig. 1. Of course, there are many functions that have this property and you will get full credit for any choice. For example, you could also take the function,

$$\phi_j(x_j) = \begin{cases} -w_j & \text{if } w_j < 0 \\ 0 & \text{if } w_j \geq 0. \end{cases}$$

(c) One idea is to penalize the differences,

$$\phi(\mathbf{w}) = \sum_{j=2}^{p} (w_j - w_{j-1})^2.$$

(d) We use an ℓ_1 version of the regularizer in part (c):

$$\phi(\mathbf{w}) = \sum_{j=2}^{p} |w_j - w_{j-1}|.$$

This will promote sparsity so that $w_j = w_{j-1}$ for most j.

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,

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2, \quad z_i = \frac{x_i - \bar{x}_i}{\sigma_j}. \tag{1}$$

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

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

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

$$\hat{u} = \alpha_1 z_1 + \alpha_2 z_2, \quad z_i = \frac{z_j - \bar{z}_j}{\sigma_j}, \quad u = \frac{\hat{y} - \bar{y}}{\sigma_y}$$

She obtains parameters $\boldsymbol{\alpha} = [0.6, -0.3]$? What are the parameters β in the original model (1).

Solution

(a) Since the features are in different units with different variances, you cannot use the LASSO regularization without normalization. In this case, $\sigma_1 \gg \sigma_2$, so a change in β_1 has a much larger effect than a change in β_2. Hence you cannot penalize the coefficients equally.

(b) To recover the original model,

$$\begin{aligned}\widehat{y} &= \bar{y} + \sigma_y \widehat{u} = \bar{y} + \sigma_y \left[\sum_j \alpha_j z_j\right] = \bar{y} + \sigma_y \left[\sum_j \alpha_j \frac{x_j - \bar{x}_j}{\sigma_j}\right] \\ &= \beta_0 + \sum_j \beta_j x_j,\end{aligned}$$

where

$$\begin{aligned}\beta_0 &= \bar{y} - \sigma_y \sum_j \frac{\alpha_j \bar{x}_j}{\sigma_j} = 300 - 100\left[\frac{0.6(50000)}{15000} - \frac{(0.3)(45)}{10}\right] = 235 \\ \beta_1 &= \frac{\sigma_y \alpha_1}{\sigma_1} = \frac{(100)(0.6)}{15000} = 0.004 \\ \beta_2 &= \frac{\sigma_y \alpha_2}{\sigma_2} = -\frac{(100)(0.3)}{10} = 3.\end{aligned}$$

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 One solution is as follows:

```
# Normalize the data
# Note the python broadcasting
xstd = np.std(Xtr,axis=0)
xmean = np.mean(Xtr,axis=0)
Ztr = (Xtr-xmean[:,None])/xstd[:,None)

ystd = np.std(y)
ymean = np.mean(y)
u = (y-ymean)/ystd


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

# Rescale the test data.
# Note:  Use the normalization learnt in training
Zts = (Xts-xmean[:,None])/xstd[:,None]

# Predict the test data
uts = model.predict(Zts)

# Rescale the output prediction
yhat = ymean + ystd*ustd

# Measure the RSS
rss = np.sum((yts-yhat)**2)
```

A commmon mistake is to use re-learn the normalization on the test data. For example, many beginning students do the following:

```
xstd_ts = np.std(Xts,axis=0)
xmean_ts = np.mean(Xts,axis=0)
Zts = (Xts-xmean_ts[:,None])/xstd_ts[:,None]
```

This is wrong! The scaling has to be considered as part of the learning process and should be fixed in test.

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

$$y \approx \hat{y} = \sum_{j=1}^{K} \beta_j e^{-\alpha_j x}, \tag{2}$$

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,

$$y \approx \hat{y} = \sum_{j=1}^{p} \tilde{\beta}_j e^{-\tilde{\alpha}_j x}, \tag{3}$$

where the values $\tilde{\alpha}_j$ are a fixed, large number p of possible values for α_j and $\tilde{\beta}_j$ are the coefficients in the model. The values $\tilde{\alpha}_j$ are *fixed*, so only the parameters $\tilde{\beta}_j$ need to be learned. Hence, the model (3) is linear. The model (3) is equivalent to (2) 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 lamb
beta = model.fit(Z,y)           # Finds the model parameters using the
                                # LASSO objective
                                #   ||y-Z*beta||^2 + lamb*||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 (3) 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 We can do the following python code:

```
# Discretize the space of exponents
p = 100
alphat = np.linspace(a,b,p)

# Transform the training data:
# Ztr[i,j] = np.exp(-xtr[i]*alphat[j])
Ztr = np.exp(-xtr[:,None]*alphat[None,:])

# Fit the linear model on the transformed features
model = Lasso(lam=lam)
betat = model.fit(Ztr,ytr)

# Predict the values on the test data
Zts = np.exp(-xts[:,None]*alphat[None,:])
yhat = model.predict(Zts, betat)

# Measure the RSS per sample
rss = np.mean((yts-yhat)**2)

# Extract the top k features
k = 3
I = np.argsort(np.abs(betat))
```

```
I = I[p-k:p]  # The largest k coefficients
beta  = betat[I]
alpha = alphat[I]
```

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 For case (i), $|w| = w$ and $J(w) = \frac{1}{2}(y-w)^2 + \lambda w$. Hence,

$$J'(w) = w - y + \lambda = 0 \Rightarrow \widehat{w} = y - \lambda.$$

In this case, $\widehat{w} = y - \lambda > 0$ when $y > \lambda$. Similarly, for case (ii), $|w| = -w$ and $J(w) = \frac{1}{2}(y-w)^2 - \lambda w$. Hence,

$$J'(w) = w - y - \lambda = 0 \Rightarrow \widehat{w} = y + \lambda.$$

Therefore, $\widehat{w} = y + \lambda < 0$ when $y < -\lambda$. Case (iii) will occur when neither of the conditions for case (i) and (ii) are satisfied, which occurs when $|y| \leq \lambda$. This gives the solution,

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