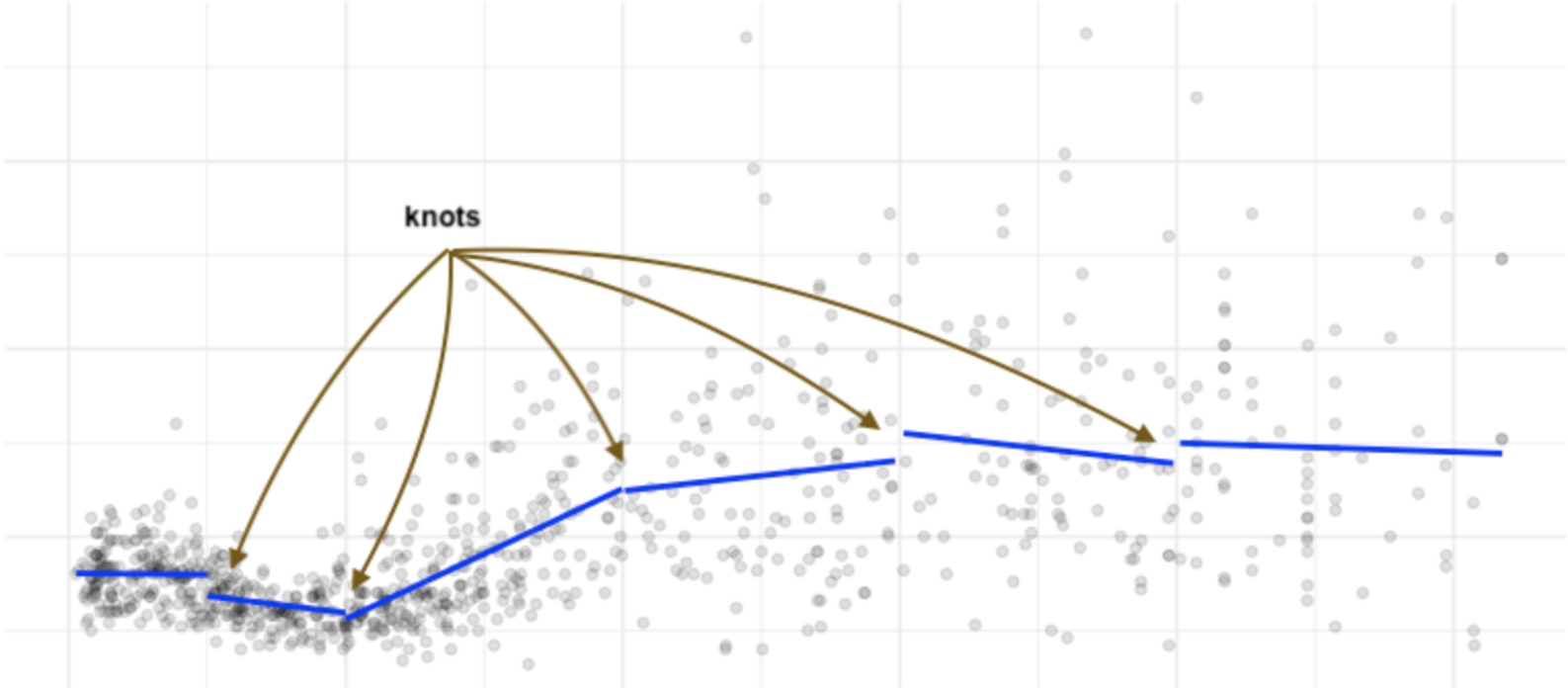
Lec 13: Regularized Learning

Ailin Zhang

Roadmap for Regularized Learning

- Ridge regression
- Lasso regression
- Coordinate descent
- Spline regression
- Least angle regression
- Stagewise regression for epsilon learning
- Bayesian regression
- Perceptron
- SVM
- Adaboost

Spline Regression - Graphical Understanding



Piecewise linear

Question: How to avoid the jumps?

$(x-k)_+ = \max(0, x-k)$

Note that knot locations are defined before estimating regression coefficients

Spline Regression Example

$$y = \beta_0 + \beta_1 x + \beta_2 (x - \kappa)_+$$

- $\beta_0 = \mathbb{E}[y \mid x = 0]$ (assuming $\kappa > 0$)
- $\beta_1 =$ Expected change in y for a 1-unit increase in x, when $x < \kappa$
- $\beta_2 =$ Change in slope between $x < \kappa$ and $x > \kappa$
- $\beta_1 + \beta_2 =$ Expected change in y for a 1-unit increase in x, when $x \geq \kappa$
- It can be easily summarized that these basis functions lead to a composite function $f(x)$ that:
  - Is everywhere continuous
  - Is linear everywhere except the knots
  - Has a different slope for each region

Spline Regression

Consider the piecewise linear spline model, where the training examples are (x_i, y_i), for $i = 1, ..., n$, and x_i is one-dimensional.

The model assumes

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j \max(0, x_i - k_j) + \epsilon_i,$$

where k_j is the j-th knot of the spline, and β_j is the change of slope at knot k_j. We can estimate β by minimizing

$$\sum_{i=1}^{n} \left[y_i - \left(\beta_0 + \sum_{j=1}^{p} \beta_j \max\left(0, x_i - k_j\right) \right) \right]^2 + \lambda \sum_{j=1}^{p} \beta_j^2,$$

Basis Function (Splines)

Spline regression models are in fact special cases of a basis function approach.

The idea is to have at hand a family of basis functions or transformations that can be applied to a variable X: function $b_1(X), b_2(X), ..., b_K(X)$. Instead of fitting a linear model in X, we fit the model

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + ... + \beta_K b_K(x_i) + \epsilon_i$$

All of the tools for linear models are available in this setting.

Piecewise quadratic models

A piecewise quadratic model with multiple knots:

we can write this as

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \sum_{j=1}^{p} \beta_{j+2} (x - k_j)_+^2$$

where $\{k_j\}_{j=1}^{p}$ are the locations of the change points

- Similar extension for cubics:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \sum_{j=1}^{p} \beta_{j+3} (x - k_j)_+^3$$

- Piecewise quadratic models are smooth and have continuous first derivatives

Natural Splines

Natural splines impose two additional constraints at each boundary region:

- The spline function is constrained to be close to linear when $X <$ smallest knot.
- The spline function is constrained to be close to linear when $X >$ largest knot.

These additional constraints of natural splines generally result in more stable estimates at the boundaries.

$$y = \beta_0 + \beta_1 x + \sum_{j=1}^{p} \beta_{j+1} N_j(x)$$

Discussion: How do you design $N_j(x)$?

$$N_j(X) = d_j(X) - d_{p-1}(X),\ d_j(X) = \frac{(X-k_j)_+^3 - (X-k_p)_+^3}{k_p - k_j}$$

Splines Summary

- A piecewise m degree polynomial that is continuous up to its first $m-1$ derivatives
- By requiring continuous derivatives, we ensure that the resulting function is as smooth as possible
- We can obtain more flexible curves by increasing the degree of the spline and/or by adding knots
- However, there is a tradeoff:
  - Few knots/low degree: Resulting class of functions may be too restrictive (bias)
  - Many knots/high degree: We run the risk of overfitting (variance)

How to choose knots?

The regression spline is most flexible in regions that contain a lot of knots.

- Place knots where it is clearly obvious that we have a distributional shift in direction.
- Place more knots on regions where we see more variability.
- Place fewer knots in places which look more stable.

Alternatively, we can place knots in a uniform fashion (e.g. at the 25th, 50th and 75 th percentiles).

Cross-validation

we remove a portion of the data (say 10 %), fit a spline with a certain number of knots to the remaining data, and then use the spline to make predictions for the held-out portion. We repeat this process multiple times until each observation has been left out once, and then compute the overall cross-validated RSS. This procedure can be repeated for different numbers of knots K. Then the value of K giving the smallest RSS is chosen.

R code

```
n = 20
p = 500
sigma = .1
lambda = 1.
x = runif(n)
x = sort(x)
Y = x^2 + rnorm(n)*sigma
X = matrix(x, nrow=n)
for (k in (1:(p-1))/p)
  X = cbind(X, (x>k)*(x-k))
beta = myRidge(X, Y, lambda)
Yhat = cbind(rep(1, n), X)%*%beta
plot(x, Y, ylim = c(-.2, 1.2), col = "red")
par(new = TRUE)
plot(x, Yhat, ylim = c(-.2, 1.2), type = 'l', col = "green")
```

Smoothing Splines and Regularization

- The complexity of the fit is controlled by regularization.

Among all functions $f(x)$ with two continuous derivatives, find one that minimizes the penalized residual sum of squares

$$\text{RSS}(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 \, dt,$$

where λ is a fixed smoothing parameter. Two special cases are:

- $\lambda = 0$: f can be any function that interpolates the data.
- $\lambda = \infty$: the simple least squares line fit, since no second derivative can be tolerated.

Regularization

$$f(x) = \beta_0 + \beta_1 b_1(x) + \beta_2 b_2(x) + \beta_3 b_3(x) + ... + \beta_K b_K(x) = \mathbf{B}\beta$$

$$\text{RSS}(\beta, \lambda) = (\mathbf{y} - \mathbf{B}\beta)^T(\mathbf{y} - \mathbf{B}\beta) + \lambda\beta^T\mathbf{\Omega}_B\beta$$

where $\{\mathbf{B}\}_{ij} = b_j(x_i)$ and $\{\mathbf{\Omega}_B\}_{jk} = \int B_j''(t)B_k''(t)dt$. The solution is easily seen to be

$$\hat{\beta} = \left(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{\Omega}_B\right)^{-1}\mathbf{B}^T\mathbf{y}$$

a generalized ridge regression. The fitted smoothing spline is given by

$$\hat{f}(x) = \sum_{j=1}^{p} B_j(x)\hat{\beta}_j = \mathbf{B}\left(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{\Omega}_B\right)^{-1}\mathbf{B}^T\mathbf{y}$$

Spline Generalization

- Nonparametric Logistic Regression
- Multidimensional Splines
- Wavelet Smoothing

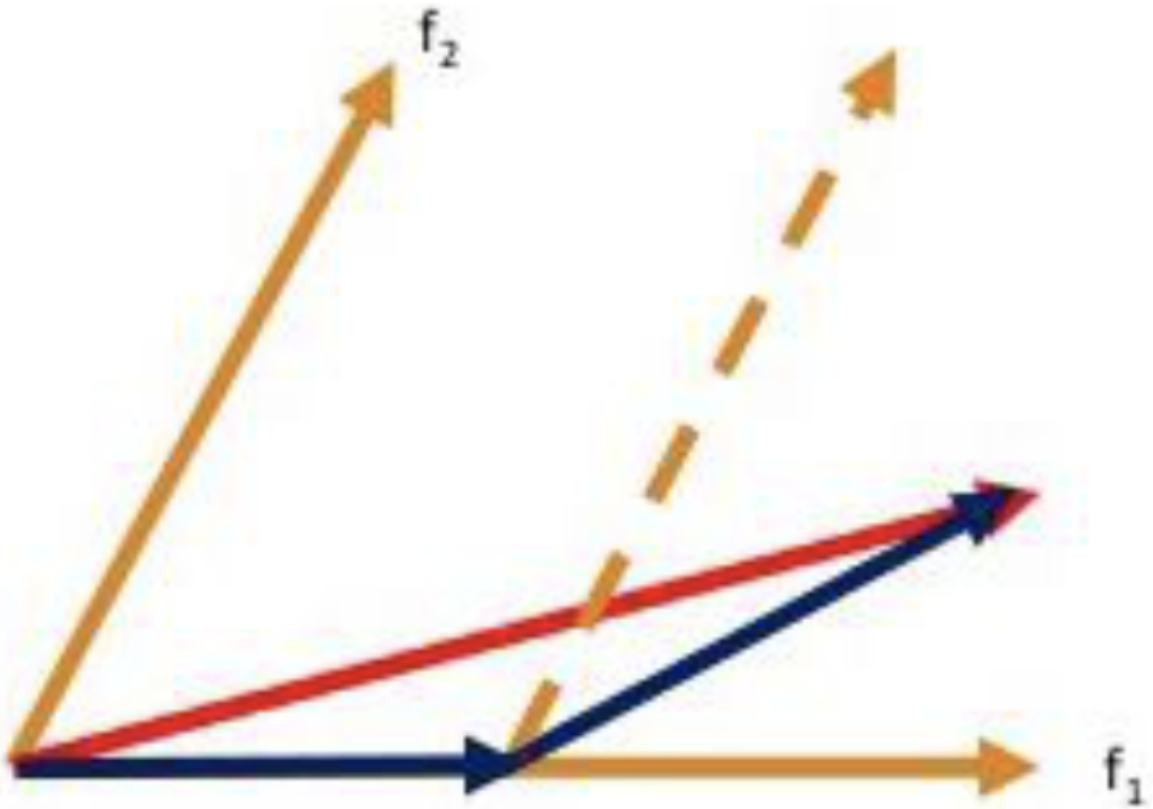
The above spline model can be extended in the following directions:

- Rectified neural network. The piecewise linear form of the linear spline also underlies modern multi-layer neural networks, where $\max(0, r)$ is called rectified linear unit (ReLU).

Least Angle Regression (LAR)

- LAR is a type of forward step-wise regression.
  - Forward step-wise regression builds a model sequentially, adding one variable at a time. At each step, it identifies the best variable to include in the active set, and then updates the least squares fit to include all the active variables.
- LAR is connected to Lasso regression.
- LAR was defined in the Efron et al., 2004. It is a relatively newer algorithm and is viewed as a democratic version of the forward step-wise regression.

Least Angle Regress (LAR)

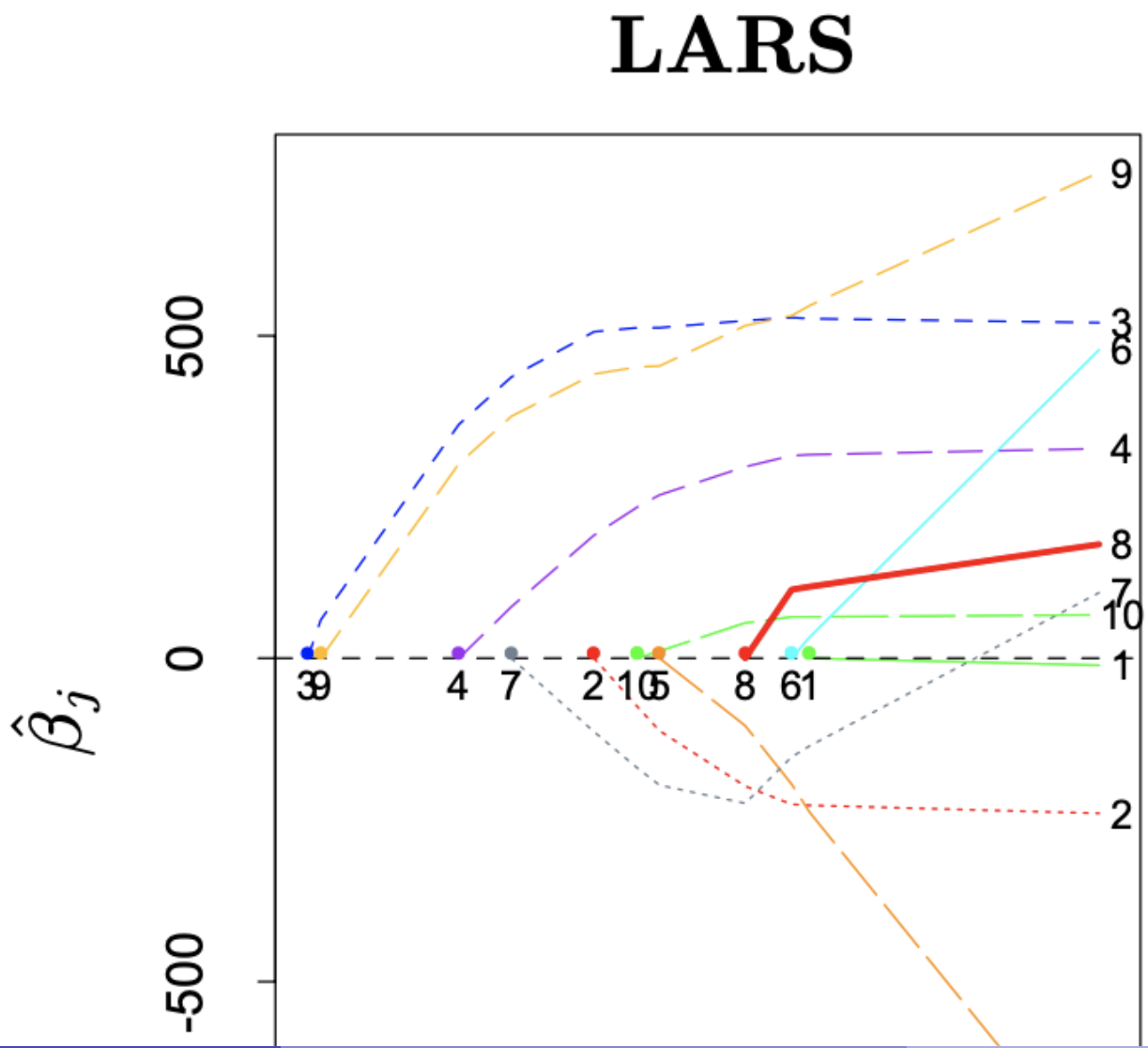


Least Angle Regression (LAR)

The basic steps of the Least-angle regression algorithm are:

1. Start with all coefficients β equal to zero.
2. Find the predictor x_j most correlated with y
3. Increase the coefficient β_j in the direction of the sign of its correlation with y. Take residuals $R = y - \hat{y}$ along the way. Stop when some other predictor x_k has as much correlation with r as x_j has.
4. Increase (β_j, β_k) in their joint least squares direction, until some other predictor x_m has as much correlation with the residual$r$.
5. Increase $(\beta_j, \beta_k, \beta_m)$ in their joint least squares direction, until some other predictor x_n has as much correlation with the residual r.
6. Continue until: all predictors are in the model

LAR Solution Path



LAR vs Lasso

If β is the Lasso solution, at any given λ, let $\mathbf{R} = \mathbf{Y} - \sum_{j=1}^{p} \mathbf{X}_j \beta_j$, then $\hat{\beta}_j = \beta_j + \langle \mathbf{R}, \mathbf{X}_j \rangle / \|\mathbf{X}_j\|_{\ell_2}^2$. then

$$\langle \mathbf{R}, \mathbf{X}_j \rangle = \begin{cases} \lambda, & \text{if } \beta_j > 0, \\ -\lambda, & \text{if } \beta_j < 0, \\ s\lambda & \text{if } \beta_j = 0. \end{cases}$$

where $|s| < 1$.

Thus in the above process, for all of those selected $\mathbf{X}_j$, the algorithm maintains that $\langle \mathbf{R}, \mathbf{X}_j \rangle$ to be λ or $-\lambda$, for all selected $\mathbf{X}_j$. If we interpret $|\langle \mathbf{R}, \mathbf{X}_j \rangle|$ in terms of the angle between $\mathbf{R}$ and $\mathbf{X}_j$, then we may call the above process as least angle regression (LAR). In fact, the solution path is piecewise linear, and the LARS computes the linear pieces analytically instead of gradually reducing λ as in coordinate descent.

Lasso can be thought of as restricted versions of LAR