10-708: Probabilistic Graphical Models 10-708, Spring 2014

26 : Structured Sparse Additive Models

Lecturer: Eric P. Xing *Scribes: Ruikun Luo, Hao Zhang*

1 Nonparemetric regression and kernel smoothing

1.1 Parametric models: Linear Regression with non-linear basis functions

Although the linear regression with linear basis is widely used in different areas, it is not powerful enough for lots of the real world cases as not all the models are linear in the real world. However, we can use non-linear basis functions to deal with non-linear relationships. It is just a linear combination of some function of x, $\phi_j(x)$.

$$y = \theta_0 + \sum_{j=1}^{m} \theta_j \phi_j(x) \tag{1}$$

where $\phi_j(x)$ are fixed basis functions and we define that $\phi_0(x) = 1$. There are many basis functions, e.g:

- Polynomial $\phi_j(x) = x^{j-1}$
- Radial basis functions $\phi_j(x) = \exp(-\frac{(x-\mu_j)^2}{2s^2})$
- Sigmoidal $\phi_j(x) = \sigma(\frac{x-\mu_j}{s})$, etc.

Note there will be dependent dimension in basis functions because we manually increase the dimensions. So if in high dimension space, we need to reduce the redundant information. In parametric models, we will be concerned with estimating the weights θ and choosing the model order m.

1.2 Non-parametric models: Locally weighted linear regression

The above linear regression based model treats all the training data equally. However, it will make more sense if we weight the training data according to their distance between the query point, because the data closed to the query point contains more useful information about the query point. The objective function for linear regression model is $J(\theta) = \frac{1}{2}\sum_{i=1}^{T}(\mathbf{x}_i^T\theta - y_i)^2$ which treats each data equally. Instead, we can modify the objective funtion to add a weight for each data point according to its distance to the query point:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{T} w_i(\mathbf{x}_i^T\theta - y_i)^2 \tag{2}$$

where $w_i = \exp(-\frac{(\mathbf{x}_i - \mathbf{x})^2}{2\tau^2})$, $\mathbf{x}$ is the query point for which we'd like to know its corresponding y. The weight function w_i ensures that higher weight will be given to the data points that close to the query point and lower weight to the data points that are far away to the query point. The weight function w_i also requires that we have to save all the training data points and use them when we estimate the corresponding y of each query point $\mathbf{x}$. This model is called locally weighted linear regression model. This model is actually a non-parametric method, however (unweighted) linear regression is a parametric model. There is a similar method kernel SVM which is also a non-parametric algorithm.

1.3 Parametric Algorithms vs. Non-parametric Algorithms

Parametric model assumes all data can be represented using a fixed, finite number of parameters (like the θ in the linear regression model). Once we have fixed the parameter θ and stored them, we no longer need the training data to make future predictions and we can just throw them away.

In contrast, the number of parameters in non-parametric models can grow with the training sample size. Like in the weighted linear regression, we need to store all the training data and use them for future predictions.

1.4 Probabilistic Interpretation of Regression

Assume that each data point y_k is generated using the following recipe:

$$y_k = \theta^T \mathbf{x}_k + \mathcal{N}(0, \sigma^2) \tag{3}$$

This can lead us to $P(y_i|x_i) \sim \mathcal{N}(\theta^T x_i, \sigma^2)$. Thus we can have that $\mathbb{E}(Y|X = \mathbf{x}) = \theta^T \mathbf{x}$ is a function of X. The estimator of y is just $\mathbb{E}(Y|X = \mathbf{x})$. Note that the estimation of θ is $(X^T X)^{-1} X^T y$.

In the non-parametric regression model, we concern about the estimation of the regression function:

$$m(\mathbf{x}) = \mathbb{E}(Y|X = \mathbf{x}) \tag{4}$$

from a training set $\{(\mathbf{x}^{(i)}, y^{(i)}) : \mathbf{x}^{(i)} \in \mathbb{R}^p, y^{(i)} \in \mathbb{R}, i = 1, ...n\}$. Note that the regression function $m(\mathbf{x})$ is the estimator of y corresponding to $\mathbf{x}$.

1.5 Kernel Smoother and Linear Smoother

Kerner smoother is a kind of linear smoother. A popular non-parametric regression estimator is the Nadaraya-Watson kernel estimator:

$$\hat{m}(\mathbf{x}) = \frac{\sum_{i=1}^{n} y^{(i)} K(\frac{||\mathbf{x} - \mathbf{x}^{(i)}||}{h})}{\sum_{i=1}^{n} K(\frac{||\mathbf{x} - \mathbf{x}^{(i)}||}{h})} \tag{5}$$

where $K(\cdot)$ is the smoothing kernel function and h is the bandwidth. The smoothing kernel function should statisfy the following properties.

1. $\int K(x)dx = 1$

2. $\int xK(x)dx = 0$

3. $\sigma_K^2 \equiv \int x^2 K(x)dx > 0$

Note that the choice of bandwidth h is more important than the type of kernel function $K(\cdot)$. Small h leads to the rough estimates and large h leads to the smoother estimates.

Linear smoothers are defined as

$$\hat{m}(\mathbf{x}) = \sum_{i=1}^{n} \ell_i(\mathbf{x}) y^{(i)} = \ell(\mathbf{x})^T \mathbf{y} \tag{6}$$

where

$$\ell_i(\mathbf{x}) = \frac{K(\frac{||\mathbf{x}-\mathbf{x}^{(i)}||}{h})}{\sum_{i=1}^{n} K(\frac{||\mathbf{x}-\mathbf{x}^{(i)}||}{h})} \tag{7}$$

where $\ell(\mathbf{x})$ is a vector with each element $\ell_i(\mathbf{x})$ and $\mathbf{y} \in \mathbb{R}^n$. For each query point $\mathbf{x}$, the estimator $(\hat{y})$ is a linear combination of $y^{(i)}$ and the weights are functions of $\mathbf{x}$. In linear regression, we have $\hat{y} = \theta\mathbf{x} = ((X^T X)^{-1}\mathbf{y})^T \mathbf{x}$. It can also be represented using the linear smothers. We have that $\ell(\mathbf{x}) = (X^T X)^{-1}\mathbf{x}$.

An alternative view of the linear smoother is:

$$\hat{\mathbf{y}} = S\mathbf{y} \tag{8}$$

where $\hat{\mathbf{y}}$ is defined as $(\hat{m}(\mathbf{x}^(1)), ..., \hat{m}(\mathbf{x}^(n)))$ be the fitted values of the training examples. $S \in \mathbb{R}^{n \times n}$ is called the smoother matrix with $S_{ij} = \ell_j(\mathbf{x}^{(i)})$. The fitted values are the smoother version of the original values. Compare this with the population setting (population setting basically means the asymtoptic region where $N \to \infty$.), we have the regression function $m(X) = \mathbb{E}(Y|X)$ as:

$$m(X) = PY \tag{9}$$

where $P = \mathbb{E}(\cdot|X)$ is the conditional expectation operator that projects a random variable (it is Y here) onto the linear space of X. S can be viewed as a non-parametric version of P.

2 Additive models

In high dimension case, the data easily become very sparse, and the linear smoothers suffers the curse of dimensionality. Hastie & Tibshirani(1990) proposed the addtive model:

$$m(X_1, ..., X_p) = \alpha + \sum_{j=1}^{p} f_j(X_j) \tag{10}$$

where each f_j is a smooth one-dimensional component function. j is the dimension index. p is the number of dimensions. X_j is a vector of values of jth dimension of all data. We can assume that $\mathbb{E}[f_j(X_j)] = 0$ for each j (because we can normalize the data the 0 mean). The problem can be solved by backfitting algorithm, which is a coordinate descent type of algorithm.

The optimization problem in the population setting is

$$\begin{aligned} J(\alpha, f_1, ..f_p) &= \tfrac{1}{2}\mathbb{E}[(Y - \alpha - \sum_{j=1}^{p} f_j(X_j))^2] \\ &= \tfrac{1}{2}\mathbb{E}[Y^2 + \alpha^2 + (\sum_{j=1}^{p} f_j(X_j))^2 - 2Y\alpha - 2Y\sum_{j=1}^{p} f_j(X_j) + 2\alpha\sum_{j=1}^{p} f_j(X_j)] \end{aligned} \tag{11}$$

Thus we have

$$\begin{aligned} \tfrac{\partial}{\partial\alpha} J(\alpha, f_1, ..f_p) &= \tfrac{\partial}{\partial\alpha}\tfrac{1}{2}\mathbb{E}[Y^2 + \alpha^2 + (\sum_{j=1}^{p} f_j(X_j))^2 - 2Y\alpha - 2Y\sum_{j=1}^{p} f_j(X_j) + 2\alpha\sum_{j=1}^{p} f_j(X_j)] \\ &= \alpha - \mathbb{E}[Y] + \mathbb{E}[\sum_{j=1}^{p} f_j(X_j)] \\ &= \alpha - \mathbb{E}[Y] + 0 \\ &= 0 \end{aligned} \tag{12}$$

Then we have $\hat{\alpha} = \mathbb{E}[Y]$.

$$\begin{aligned} \tfrac{\partial}{\partial f_j} J(\alpha, f_1, ..f_p) &= \tfrac{\partial}{\partial f_j}\tfrac{1}{2}\mathbb{E}[Y^2 + \alpha^2 + (\sum_{j=1}^{p} f_j(X_j))^2 - 2Y\alpha - 2Y\sum_{j=1}^{p} f_j(X_j) + 2\alpha\sum_{j=1}^{p} f_j(X_j)] \\ &= \tfrac{\partial}{\partial f_j}\tfrac{1}{2}\mathbb{E}[(\sum_{j=1}^{p} f_j(X_j))^2] - \mathbb{E}[Y] + \mathbb{E}[\alpha] \\ &= \tfrac{\partial}{\partial f_j}\tfrac{1}{2}\mathbb{E}[(\sum_{k\neq j} f_k(X_k) + f_j(X_j))^2] - \mathbb{E}[Y] + \mathbb{E}[\alpha] \\ &= \tfrac{\partial}{\partial f_j}\tfrac{1}{2}\mathbb{E}[(\sum_{k\neq j} f_k(X_k))^2 + 2f_j(X_j)\sum_{k\neq j} f_k(X_k) + f_j(X_j)^2] - \mathbb{E}[Y] + \mathbb{E}[\alpha] \\ &= \mathbb{E}[f_j(X_j) + \sum_{k\neq j} f_k(X_k)] - \mathbb{E}[Y] + \mathbb{E}[\alpha] \\ &= 0 \end{aligned} \tag{13}$$

Then we have $f_j = \mathbb{E}[(Y - \alpha - \sum_{k\neq j} f_k)|X_j] := P_j R_j$, where $P_j = \mathbb{E}[\cdot|X_j]$ is the conditional expectation operator onto jth input space, $R_j = Y - \alpha - \sum_{k\neq j} f_k$ is the partial residual.

Replace conditional operator P_j by smoother matrix S_j results in the backfitting algorithm.

Backfitting Algorithm

1. Initialize: $\hat{\alpha} = \sum_{i=1}^{n} y^{(i)}/n$, $\hat{f}_j = 0, j = 1, ...p$
2. Cycle: for $j = 1, ..., p, 1...p....$ (until $J(\cdot)$ doesn't change)

$$\begin{aligned} &\hat{f}_j \leftarrow S_j(\mathbf{y} - \hat{\alpha} - \sum_{k \neq j} \hat{\mathbf{f}}_k) \\ &\text{Centering:} \hat{\mathbf{f}}_j \leftarrow \hat{\mathbf{f}}_j - \frac{1}{n} \sum_{i=1}^{n} \hat{f}_j(x_j^{(i)}) \end{aligned} \tag{14}$$

3 Sparse additive models (SpAM)

The additive model works up to certain dimension. But if the data dimension is much larger than the dataset size $p >> n$, we will need to penalize f_j similar to the way we penalize β_j coefficients in the lasso to restrict the dimensionality of the regressions. This leads to the Sparse Additive Model (SpAM) formulation, which encourage functional sparsity. The optimization problem in the population setting is shown in (15). Note that the data is centered data.

$$\frac{1}{2}\mathbb{E}[(Y - \sum_{j=1}^{p} f_j(X_j))^2] + \lambda \sum_{j=1} p\sqrt{\mathbb{E}[f_j(X_j)^2]} \tag{15}$$

$\sum_{j=1} p\sqrt{\mathbb{E}[f_j(X_j)^2]}$ behaves like an l_1 ball across different componets to encourage functional sparsity. Note that if each component function $f_j(X_j)$ is constrained to have the linear form, the formulation reduces to standard lasso problem.

The optimum is achieved by soft-thresholding step:

$$f_j = [1 - \frac{\lambda}{\sqrt{\mathbb{E}[(P_j R_j)^2]}}] P_j R_j, j = 1, ..., p \tag{16}$$

where $R_j = Y - \sum_{k \neq j} f_k$ is the partial residual. $[\cdot]_+$ is the positive part, which indicates that $f_j = 0$ if only if $\sqrt{\mathbb{E}[(P_j R_j)^2]} \leq \lambda$. As in standard additive models, replace P_j by S_j, the backfitting algorithm for SpAM is:

$$\hat{\mathbf{f}}_j \leftarrow [1 - \frac{\lambda}{\hat{s}_j}]_+ S_j(\mathbf{y} - \sum_{k \neq j} \hat{\mathbf{f}}_k), j = 1, ..., p \tag{17}$$

where $\hat{s}_j = \sqrt{\text{mean}(S_j(\mathbf{y} - \sum_{k \neq j} \hat{\mathbf{f}}_k))}$ is the empirical estimate of $\sqrt{\mathbb{E}[(P_j R_j)^2]}$ today is good

4 Structured sparse additive models (GroupSpAM)

SpAM imposes sparsity on single function and it does not utilize group structure information between covariates, which could be viewed as priori knowledge in many applications. In the parametric setting, it has been shown that if such a group structure exists and is consistent with true sparsity among covariates, the estimator accuracy could be increased by treating the whole group of covariates as a single unit. The GroupSpam achieves functional sparsity at group level by combining the spirit of GroupLasso (Yuan & Lin, 2006) and SpAM. In this sense, GroupSpAM could be viewed as a nonparametric extension of generalized GroupLasso (Friedman et al., 2010).

Assuming G is a partition of $\{1, ..., p\}$ and the group in G do not overlap. The optimization problem of GroupSpAM is formulated as:

$$min_{f:f_j \in H_j} L(f) + \lambda \Omega_{group}(f) \tag{18}$$

where $L(f)$ is expected square error:

$$L(f) = \frac{1}{2}\mathbb{E}[(Y - \sum_{j=1}^{p} f_j(X_j))^2] \tag{19}$$

and $\Omega_{group}(f)$ is the regularization functional penalty defined as:

$$\begin{array}{rcl} \Omega_{group}(f) & = & \sum_{g \in G} \sqrt{d_g} \|f_g\| \\ & = & \sum_{g \in G} \sqrt{d_g} \sqrt{\sum_{j \in g} \mathbb{E}[f_j^2(X_j)]} \end{array} \tag{20}$$

(Yin et. al, 2012) proved the following necessary and sufficient thresholding condition at the group level:

$$\sqrt{\sum_{j \in g} \mathbb{E}[(P_j R_g)^2]} \leq \lambda \sqrt{d_g} \tag{21}$$

where the $R_g = Y - \sum_{g' \neq g} \sum_{j' \in g'} f_{j'}(X_{j'})$ is the partial residual after removing all functions from group g.

Learning functional f_j could be achieved through Block Coordinate Descent algorithm and its building block is thresholding. The Thresholding algorithm and Block Coordinate Descent algorithm are as follows:

4.1 Algorithm 1 Thresholding

1: Input: Partial residual $\widehat{R_g}$, smoother matrics $\{S_j : j \in g\}$, and tuning parameter λ.

2: Output: $\widehat{f_g} = \{\widehat{f_j} : j \in g\}$.

3: Estimate $P_j R_g$ by smoothing: $\widehat{P_j} = S_j \widehat{R_g}, \forall j \in g$.

4: Estimate $\sqrt{\sum_{j \in g} \mathbb{E}[(P_j R_g^2)]}$ by

$$\widehat{\omega_g} = \sqrt{\frac{1}{n} \sum_{j \in g} \|\widehat{P_j}\|^2} \tag{22}$$

5: if $\widehat{\omega_g} \leq \lambda \sqrt{d_g}$ then 6: Set $\widehat{f_j} = 0, \forall j \in g$ 7: else 8: Estimate $\widehat{f_g}$ by iterating the following fixed point equation over t until convergence

$$\widehat{f_g}^{(t+1)} = (\widehat{J} + \frac{\lambda \sqrt{d_g}}{\|\widehat{f_g}^{(t)}\|/\sqrt{n}} I)^{-1} \widehat{Q} \widehat{R_g} \tag{23}$$

9: end if 10 Center each $\widehat{f_j}$ by subtracting its mean.

4.2 Algorithm 2 Block Coordinate Descent

1: Input: Data $X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n$, a partition G of $\{1, ..., p\}$, and tuning parameter λ.

2: Output: Fitted functions $\widehat{f} = \{\widehat{f}_j : j = 1, ..., p\}$.

3: Initialize $\widehat{f}_j = 0 \forall j$;, pre-compute smoother matrices $S_j, \forall j$.

4: Cycle though group $g \in G$ until convergence:

(1) Compute the partial residual $R_g = Y - \sum_{g' \neq g} \sum_{j' \in g'} f_{j'}(X^{j'})$

(2) $\widehat{f}_g \leftarrow Thresholding(R_g, \widehat{\{S_j\}}_{j \in g}, \lambda)$

In the previous GroupSpAM, it assumes there is no overlap between group partitions. Like structured spare regression, allowing overlap among groups gives more flexibility but optimization could not be decoupled simply. The idea is to decompose each original component function to be a sum of a set of latent functions and then apply the functional group penalty to the decomposed:

$$minimize \quad \frac{1}{2}\mathbb{E}[(Y - \sum_{j=1}^{p} f_j(X_j))^2] + \lambda \sum_{g \in G} \sqrt{|g|}\|h^g\| \tag{24}$$

$$subjectto \quad \sum_{g: j \in g} h_j^g = f_j, j = 1, ..., p \tag{25}$$

5 Closing Comments

By relaxing parametric assumption in LASSO, GroupLASSO, Structured Sparse Regression to a non-parametric form, we come up with SparseAM, GroupSpAM and GroupSpAM with overlap. Among all the algorithm explored above, instead of representing a probability distribution, graph structure serves as a carrier of prior knowledge. Here we are being freqeuntists, and hence we do not see latent variables and latent structures which served a center role in Bayesian learning.