Further applications of penalization and sparsity

Patrick Breheny

May 6

Introduction

In our final lecture, we will explore some interesting ways of applying/extending the penalties we have learned about so far in this course to four other statistical methods:

- Additive models
- Principal components analysis
- Models with interactions
- Graphical models

Basis functions

- Suppose for the moment that we have just a single feature x and we are interested in estimating $\mathbb{E}(y|x) = f(x)$
- A common approach for extending the linear model $f(x) = x\beta$ is to augment x with additional, known functions of x:

$$f(x) = \sum_{m=1}^{M} \beta_m h_m(x),$$

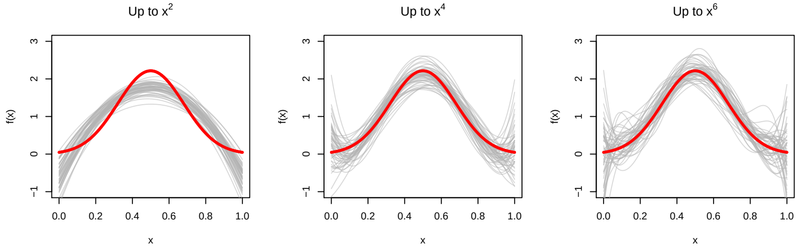
  where the $\{h_m\}$ are called *basis functions*
- Because the basis functions $\{h_m\}$ are prespecified and the model is linear in the new variables, ordinary least squares approaches can be used (at least in low-dimensional settings)

Problems with polynomial regression

- This idea is not new to you, as you have certainly worked with polynomial terms before
- However, polynomial terms introduce undesirable side effects: each observation affects the entire curve, even for x values far from the observation
- Not only does this introduce bias, but it also results in extremely high variance near the edges of the range of x
- As Hastie *et al.* (2009) put it, "tweaking the coefficients to achieve a functional form in one region can cause the function to flap about madly in remote regions"

Problems with polynomial regression (cont'd)

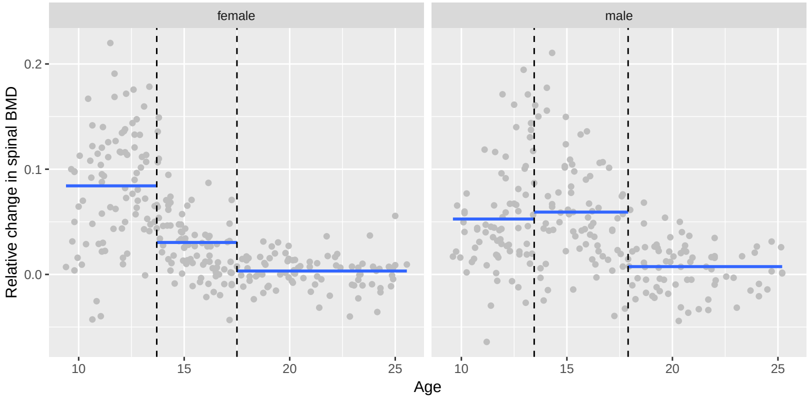
To illustrate this, consider the following simulated example (gray lines are models fit to 100 observations arising from the true f, colored red):



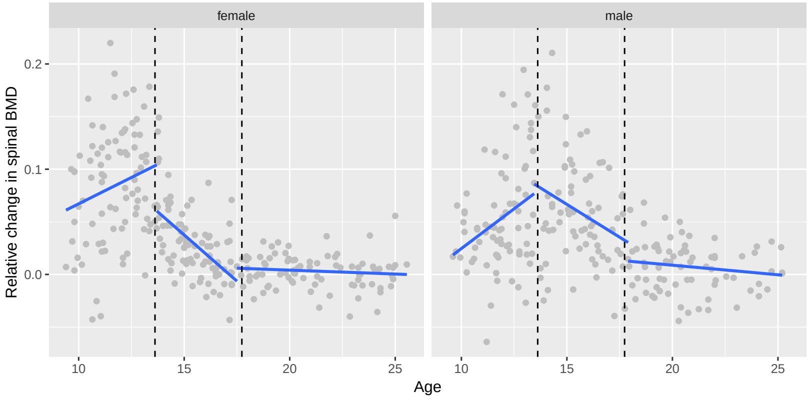
Splines

- For this reason, *local* basis functions, which ensure that a given observation affects only the nearby fit, not the fit of the entire line, are often preferred
- We will focus on a specific type of local bases called *splines*, which are just piecewise polynomials joined together to make a single smooth curve
- To understand splines, we will gradually build up a piecewise model, starting at the simplest one: the piecewise constant model
- First, we partition the range of x into $K+1$ intervals by choosing K points $\{\xi_k\}_{k=1}^K$ called *knots*

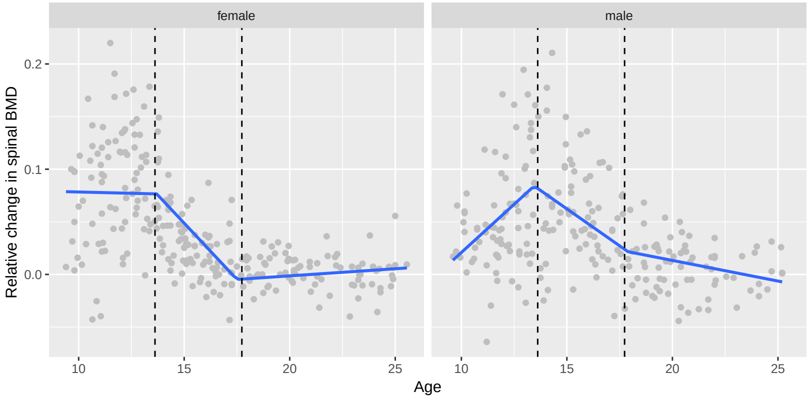
The piecewise constant model (cont'd)



The piecewise linear model



The continuous piecewise linear model



Basis functions for piecewise continuous models

These constraints can be incorporated directly into the basis functions:

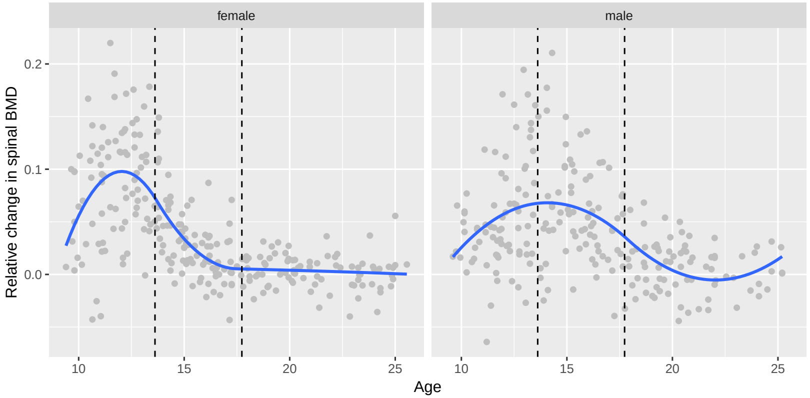
$$h_1(x) = 1, \quad h_2(x) = x, \quad h_3(x) = (x - \xi_1)_+, \quad h_4(x) = (x - \xi_2)_+,$$

where $(\cdot)_+$ denotes the positive portion of its argument:

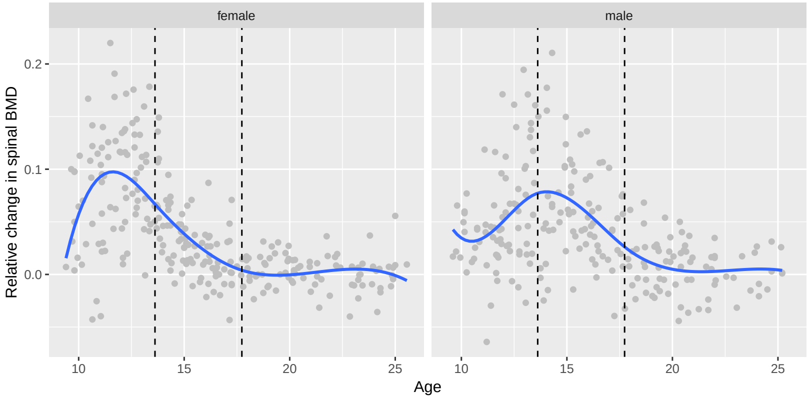
$$r_+ = \begin{cases} r & \text{if } r \geq 0 \\ 0 & \text{if } r < 0 \end{cases}$$

- Note that the degrees of freedom add up: 3 regions $\times$ 2 df/region - 2 constraints $=$ 4 basis functions
- This set of basis functions is an example of what is called the *truncated power basis*; it can be extended to any order of polynomials

Quadratic splines



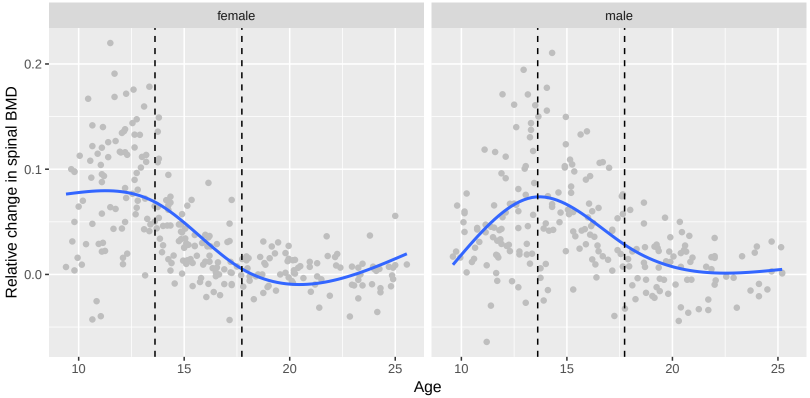
Cubic splines



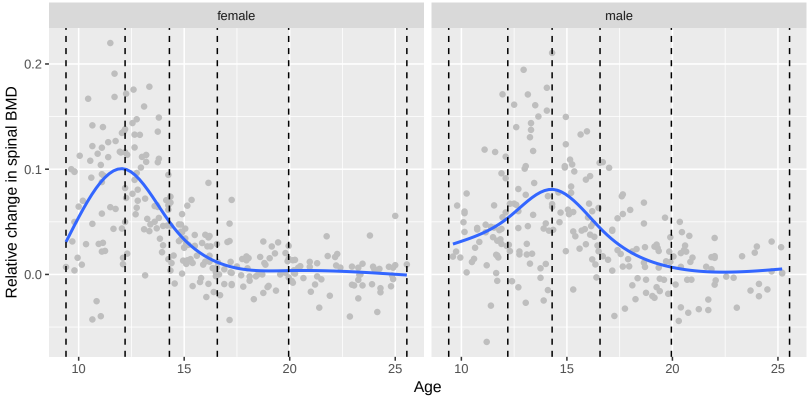
Natural cubic splines

- Polynomial fits tend to be erratic at the boundaries of the data; naturally, cubic splines share the same flaw
- *Natural cubic splines* ameliorate this problem by adding the additional (4) constraints that the function is linear beyond the boundaries of the data

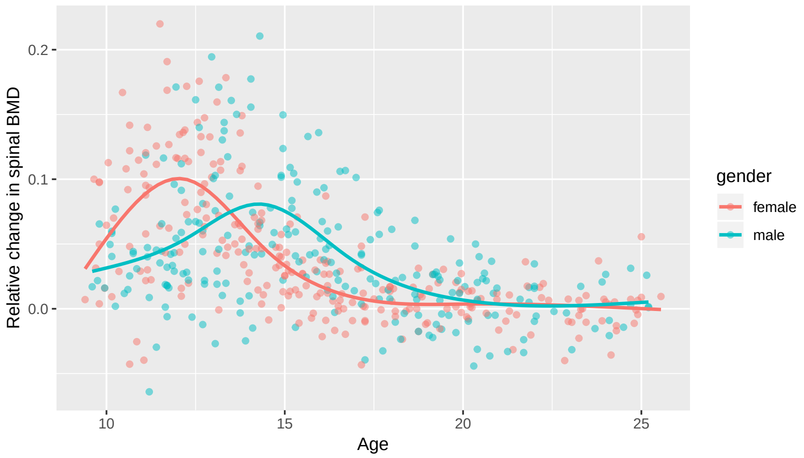
Natural cubic splines (cont'd)



Natural cubic splines, 6 df



Natural cubic splines, 6 df (cont'd)



Additive models

- When we have multiple features, a natural extension of basis functions is to assume an additive relationship:

$$f(\mathbf{x}) = \sum_j \sum_{m=1}^{M} \beta_{mj} h_{mj}(\mathbf{x}_j);$$

  such models are called *additive models* or *generalized additive models* (GAMs)
- If the number of coefficients is large, we will not wish to use maximum likelihood to estimate them, as we have seen several times in the course
- Furthermore, it is often the case that many potentially useful features are present, but we expect most of them to be unrelated to the outcome

Connection with group lasso

- However, it makes little sense in this scenario to carry out selection at the level of the individual basis functions; we want to select features, and if a feature is selected, we want all of its basis functions in the model
- Representing the problem as a group lasso model, we have

$$Q(\boldsymbol{\beta}|\mathbf{H}, \mathbf{y}) = \|\mathbf{y} - \mathbf{H}\boldsymbol{\beta}\|^2 + \sum_j \|\boldsymbol{\beta}_j\|,$$

  where $\mathbf{H}$ is the expanded design matrix with elements $h_{mj}(x_{ij})$
- This idea was originally proposed by Ravikumar et al. (2009), who named it *sparse additive models* (SPAM)

R code: Setting up splines

- To illustrate how sparse additive models work, let us apply one to the rat eye data; for the sake of simplicity, I'll restrict the analysis to the 857 genes on chromosome 5
- Our first task is setting up the splines; the calculation of the basis functions can be done by the `splines` package:

```
require(splines)
ind <- which(fData$Chr==5)
df <- 3
HH <- sapply(as.data.frame(X[,ind]),
             ns, simplify="array", df=df)
```

this produces an $n \times 3 \times 857$ array, which we can collapse into a matrix with

```
H <- t(apply(HH, 1, rbind))
```

R code: Group lasso

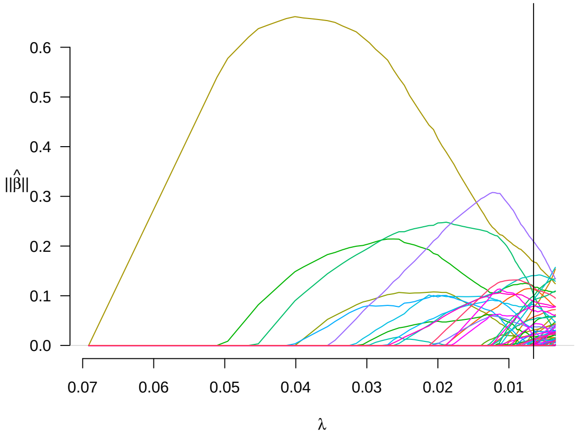
- Now, the code for fitting the group lasso model should look familiar:

```
group <- rep(1:length(ind), each=df)
require(grpreg)
cvfit <- cv.grpreg(H, y, group)
```

- For the sake of illustration, we'll compare the group lasso fit with a group MCP fit (`penalty="grMCP"`)

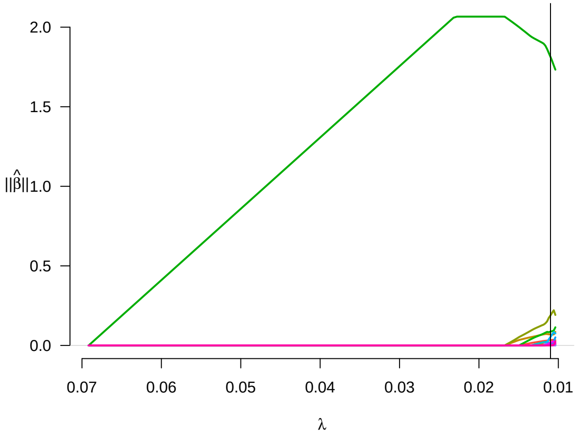
Results: Group Lasso

The group lasso model selects 49 genes, achieving an R^2 of 0.72



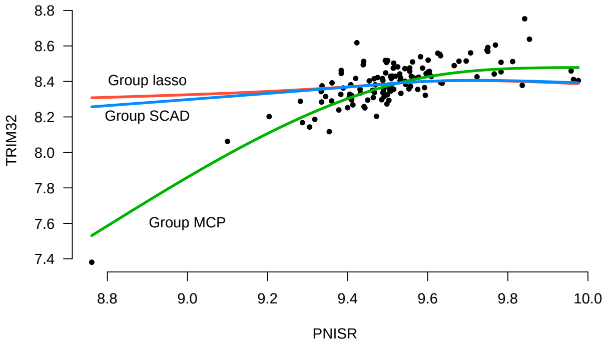
Results: Group MCP

The group MCP model selects just 5 genes, with $R^2 = 0.59$



PNISR

The first gene to enter the model is PNISR; the PNI stands for PNN-interacting, where PNN is a gene that plays a critical role in proper eye development



Introduction

- Our second topic for today is the application of penalized regression methods to principal components analysis
- The key idea behind principal components analysis is to reduce the dimension of $\mathbf{X}$ while accounting for as much of the information in $\mathbf{X}$ as possible
- This aim is achieved by transforming to a new set of variables (the principal components) that are linear combinations of the original variables
- The new set of variables have lower dimension and are uncorrelated, both of which can greatly simplify the analysis

Principal components in terms of SVD components

- Suppose that we have standardized $\mathbf{X}$, and let $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ be the singular value decomposition of $\mathbf{X}$
- By convention, the singular values $\{d_j\}$ and their associated vectors $\{\mathbf{u}_j\}$ and $\{\mathbf{v}_j\}$ are ordered, so that $d_1 \geq d_2 \geq \cdots \geq d_p$
- Now, the variables $d_j\mathbf{u}_j$ are called the *principal components* of the original data $\mathbf{X}$, for reasons that we will now describe

Properties of principal components

- First, note that the principal components are linear combinations of the original variables:

$$\mathbf{X}\mathbf{v}_j = d_j\mathbf{u}_j$$

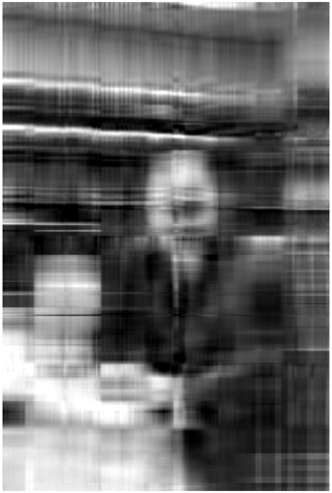
- Furthermore, $\text{Var}(d_1\mathbf{u}_1) \geq \text{Var}(d_2\mathbf{u}_2) \geq \cdots \geq \text{Var}(d_p\mathbf{u}_p)$
- Indeed, out of all possible vectors $\mathbf{z}$ that can be formed from a normalized linear combination of the original explanatory variables (*i.e.*, such that $\mathbf{z} = \mathbf{X}\mathbf{a}$ where $\mathbf{a}^T\mathbf{a} = 1$), the variable with the largest variance is $d_1\mathbf{u}_1$
- Out of all possible normalized linear combinations $\mathbf{z}$, the one that has the largest variance and is orthogonal to the first combination (*i.e.*, such that $\mathbf{z}^T\mathbf{u}_1 = 0$) is $d_2\mathbf{u}_2$, and so on

More terminology

To summarize,

- The vectors $\mathbf{v}_j$ (the columns of $\mathbf{V}$) are the principal component directions, or *loadings*, and they describe the transformation process by which the new variables are created out of the old
- The vectors $\mathbf{u}_j$ (the columns of $\mathbf{U}$) are the normalized principal components (sometimes called the *principal component scores*)
- The singular values d_j are used to rank the principal components in term of importance

Illustration



Sparse principal components

- One downside of principal components is that they can be difficult to interpret: the new variables are linear combinations of the old ones, and if p is large, the linear combination will be complex
- On a related note, suppose an investigator wanted to be able to measure a small number of features, but retain as much of the information in $\mathbf{X}$ as possible
- In both situations, the fact that the principal components are composed of all the original features poses a problem; an appealing extension would be components that are *sparse* with respect to the original features

Principal components as a regression problem

- It turns out that principal components can be written as a regression problem, where the loadings can be found by minimizing

$$\|\mathbf{X} - \mathbf{U}\mathbf{D}\mathbf{V}^T\|_F^2 = \|\mathbf{X} - \mathbf{X}\mathbf{V}\mathbf{V}^T\|_F^2$$

  such that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, where $\|\mathbf{A}\|_F$ is the Frobenius norm defined previously ($\|\mathbf{A}\|_F^2 =$ sum of squares of all elements)

- In a clever paper, Zou et al. (2006) showed that we can also find the loadings by minimizing

$$\|\mathbf{X} - \mathbf{X}\mathbf{B}\mathbf{A}^T\|_F^2 \qquad \text{such that } \mathbf{A}^T\mathbf{A} = \mathbf{I}$$

  with respect to both $\mathbf{A}$ and $\mathbf{B}$, where $\mathbf{v}_j = \mathbf{b}_j / \|\mathbf{b}_j\|$

Introducing sparsity

- The advantage of this formulation is that it is straightforward to solve for $\mathbf{A}$ and $\mathbf{B}$ separately, treating the other as fixed
- In particular, treating $\mathbf{A}$ as fixed, solving for $\mathbf{b}_j$ is equivalent to minimizing

$$\|\mathbf{X}\mathbf{a}_j + \mathbf{X}\mathbf{b}_j\|^2,$$

which is simply least squares regression with $\tilde{\mathbf{y}} = \mathbf{X}\mathbf{a}_j$

- A natural sparse extension, then, is to add an L_1 penalty: find $\mathbf{b}_j$ by minimizing

$$\|\tilde{\mathbf{y}} - \mathbf{X}\mathbf{b}_j\|^2 + \lambda\|\mathbf{b}_j\|_1$$

Sparse PCA

- The approach proposed by Zou et al., then, was to solve for sparse principal components by minimizing

$$\|\mathbf{X} - \mathbf{X}\mathbf{B}\mathbf{A}^T\|_F^2 + \sum_j \lambda_j \|\mathbf{b}_j\|_1 + \lambda_0 \sum_j \|\mathbf{b}_j\|^2$$

  such that $\mathbf{A}^T\mathbf{A} = \mathbf{I}$, where the extra ridge (elastic net) penalty is introduced to guarantee unique solutions
- As discussed on the previous slide, the problem can be solved by alternating updates for $\mathbf{A}$ and $\mathbf{B}$:
  - Updating $\mathbf{B}$ is equivalent to solving k elastic net problems, where k is the desired number of components
  - We're skipping the details for the update of $\mathbf{A}$, but it amounts to computing the SVD of $\mathbf{X}^T\mathbf{X}\mathbf{B}$

Choice of penalties

- The choice of λ_0 is not particularly important; it is typically just set to some arbitrary small positive value
- The selection of the λ_j parameters is more complex
- One could try out several values of $\{\lambda_j\}$ and attempt to make selections on the basis of the proportion of explained variance in $\mathbf{X}$
- Alternatively, a convenient thing to do in practice is simply to set λ_j at a value such that exactly, say, 3 terms appear in each principal component

WHO-ARI pneumonia data

- As an example of how this works in practice, let's apply the sparse PCA method to our WHO-ARI data set
- This is the kind of data set for which principal components are particularly attractive, as several features measure essentially the same thing; for example, it is not particularly meaningful to isolate the effect of changes in feeding ability while keeping sucking ability constant
- Still, ordinary principal components are difficult to interpret here, as they are linear combinations of all 67 variables

R code

- The sparse PCA approach described here is implemented in the R package `sparsepca`
- As with regular PCA, it is typically preferable to apply sparse PCA to the standardized design matrix (`scale=TRUE`, which is not the default):

```
fit <- spca(X, k=5, 0.1, scale=TRUE) # or use std(X)
V <- fit$loadings
Z <- X %*% V  # Principal components
```

- Here, `K` is the desired number of components and $\lambda = 0.1$; the package unfortunately does not solve the entire path

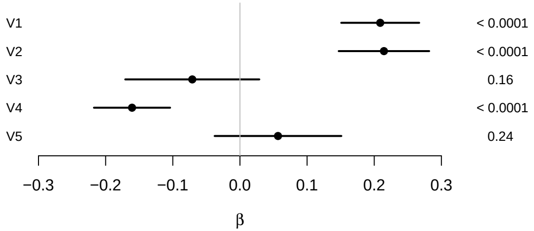
Results: Components

The first five sparse principal components are

- PC1, "Energy": Attentive, eating, not drowsy, quality of crying, amount of movement
- PC2, "Respiratory problems": Respiratory distress, lower chest in-drawing, nasal flaring
- PC3, "Hydration": Skin turgor, dehydrated, sunken fontanelle
- PC4, "Size": Weight, length, head circumference
- PC5, "Agitation": Sleeping less, crying more

Results: Predictive of pneumonia?

Fitting a linear regression for pneumonia score on these principal components, $\mathbf{XV}$, we have



So energy and respiratory problems clearly increase the likelihood of pneumonia, while size decreases it; it is not clear that hydration and agitation are useful in predicting pneumonia

Motivation

- A topic we have not really discussed thus far is how we can use penalized regression to select variables with potential interactions
- In principle, of course, you could just create a big design matrix with all the main effect and interaction terms included; however, this has two potential drawbacks:
  - We might select interactions without including the corresponding main effect term
  - We would likely end up with lots of false selections among the interaction terms because so many are present

Group lasso setup

- One approach is to set the problem up as a group lasso problem
- To illustrate, let's consider the simplest possible scenario, in which we have two features $\mathbf{x}_1$ and $\mathbf{x}_2$, along with their interaction $\mathbf{x}_{1:2}$
- Now let us construct a 5-column design matrix with columns $(\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_{1:2})$, where we will consider the first and second columns as groups containing just a single element, and the last three belonging to a combined group (i.e., `g=c(1,2,3,3,3)`)

Interpretation and latent variable representation

- The idea behind this approach is that we are parsing the main effect of $\mathbf{x}_1$ into two latent portions: the pure main effect portion and the portion belonging to the interaction group
- Letting γ denote the the coefficients for this expanded design matrix, the main effect for $\mathbf{x}_1$ would then be

$$\begin{aligned}\mathbf{x}_1\gamma_1 + \mathbf{x}_1\gamma_3 &= \mathbf{x}_1(\gamma_1 + \gamma_3) \\ &= \mathbf{x}_1\beta_1\end{aligned}$$

- As a consequence of this setup, if we select $\mathbf{x}_{1:2}$, we are guaranteed to also include its two main effects in the model as well

Simulation example

- To see how well this works, let's simulate some data under the following conditions:
  - $n = 70$, $p = 20$; so 210 potential features
  - $\mathbf{x}_j$, ε all drawn from $\mathrm{N}(0, 1)$
  - $y_i = \mathbf{x}_{i1} + \mathbf{x}_{i4} - \mathbf{x}_{i1}\mathbf{x}_{i2} + \varepsilon_i$
- We'll fit both an ordinary lasso and a group lasso model using the latent variable representation we described earlier (this is implemented in the `R` package `glinternet`), using cross-validation to select λ for both models

Results: Lasso

- The ordinary lasso model selects 11 variables: 2 main effects and 9 interaction terms
- Of note, it does select all the true effects
- However, of the 9 interaction terms, none of them have both main effects selected
- The maximum cross-validated R^2 achieved by the model is 0.80

Results: `glinternet`

- The latent variable group lasso approach selects 5 variables: 4 main effects and 1 interaction (the true interaction)
- By construction, both main effects (for $\mathbf{x}_1$ and $\mathbf{x}_2$) are included in the model for the selected interaction, $\mathbf{x}_{1:2}$
- The model also achieves a slightly higher R^2, 0.81

Introduction

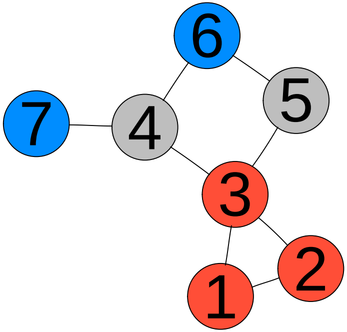
- As a final topic in this course, we will briefly discuss how penalization can be used to estimate network structure in probabilistic graphical models
- This is a big topic that we're only going to scratch the surface of; the main thing we will focus on is that graphs encode conditional independence relationships between variables
- Specifically, if a set of vertices $\mathcal{S}$ separates a graph into two disconnected components $\mathcal{A}$ and $\mathcal{B}$, then the variables in $\mathcal{A}$ are independent of the variables in $\mathcal{B}$ conditional on the variables in $\mathcal{S}$

Illustration

For example, the conditional independence statement

$$X_{1:3} \perp\!\!\!\perp X_{6:7} | X_{4:5}$$

implies (and is implied by) the graph



Gaussian graphical models

- For continuous variables, a particularly convenient type of graphical model is to assume the multivariate normal distribution $X \sim \mathrm{N}(, \boldsymbol{\Sigma})$; this is known as the *Gaussian graphical model*
- For Gaussian graphical models, it is typically more convenient to work in terms of the precision matrix $\boldsymbol{\Theta} = \boldsymbol{\Sigma}^{-1}$
- One particularly relevant property of $\boldsymbol{\Theta}$ is that $\theta_{ij} = 0$ implies that there is no edge connecting nodes i and j in the graph depicting the conditional independence relationships

GGM likelihood

- It can be shown that for a Gaussian graphical model, up to a constant, the loss ($-1/n$ times the log-likelihood) is

$$L(\boldsymbol{\Theta}|\mathbf{X}) = \operatorname{tr}(\mathbf{S}\boldsymbol{\Theta}) - \log|\boldsymbol{\Theta}|\,,$$

  where $\mathbf{S} = \frac{1}{n}\sum_i \mathbf{x}_i\mathbf{x}_i^\top$ is the sample covariance matrix
- As you might expect, the maximum likelihood estimator is unstable and inaccurate when p is large relative to n
- Furthermore, even when p is small, the MLE will not produce exact zeros for $\boldsymbol{\Theta}$ (no help for estimating the graph)
- Thus, let us consider the penalized loss:

$$L(\boldsymbol{\Theta}|\mathbf{X}) = \operatorname{tr}(\mathbf{S}\boldsymbol{\Theta}) - \log|\boldsymbol{\Theta}| + \lambda\sum_{j\neq k}|\theta_{jk}|$$

Graphical lasso algorithm

- The matrix of penalized score equations is then

$$\mathbf{S} - \boldsymbol{\Theta}^{-1} + \lambda \boldsymbol{\Psi} \ni \mathbf{0},$$

  where $\Psi_{ij} = \partial \left|\theta_{ij}\right|$

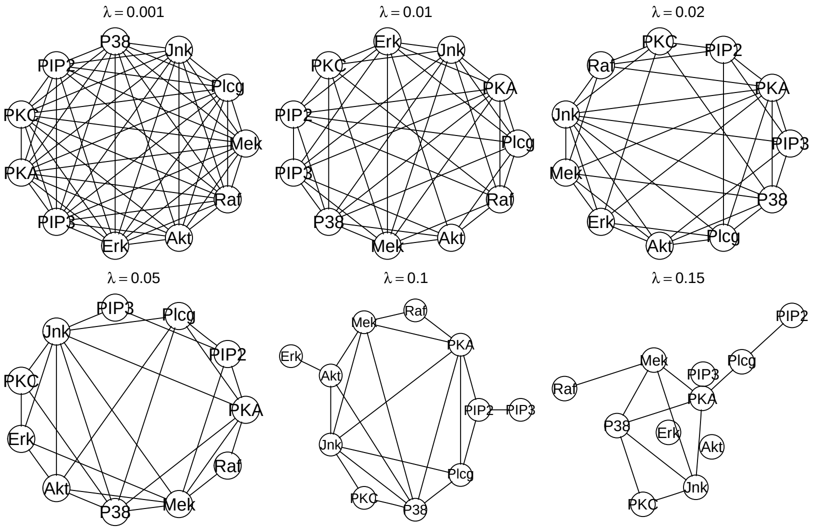
- Partitioning this equation yields

$$-\mathbf{s}_{-j} + \boldsymbol{\Sigma}_{-j,-j}\boldsymbol{\beta} + \lambda \partial \|\boldsymbol{\beta}\|_1 \ni \mathbf{0},$$

  where $\boldsymbol{\beta} = -\boldsymbol{\theta}_{-j}/\theta_{j,j}$

- Thus, we can estimate $\boldsymbol{\Theta}$ by repeatedly solving a slightly modified version of the lasso, in which we essentially iteratively regress each variable on all the others; this algorithm is known as the *graphical lasso*

Example: Protein phosphorylation network


$\lambda = 0.001$
$\lambda = 0.01$
$\lambda = 0.02$
$\lambda = 0.05$
$\lambda = 0.1$
$\lambda = 0.15$
P38
Jnk
PIP2
Plcg
PKC
Mek
PKA
Raf
PIP3
Erk
Akt