ch7 Basis Functions and Piecewise Regression

*Polynomial Regression 多項式回歸

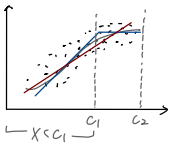
[Linear regression: $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$
Polynomial regression: $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \cdots + \beta_d x_i^d + \varepsilon_i$ △Usually don't use degree > 4 ∵ overly flexible

① Flexible: useful when not linear
② Simple and Interpretability
③ control over complexity: high degree → fit complex pattern
[overfitting → small dataset
[underfitting → simple model

*Step Function: define regions which different models will be fit

→ creating thresholds: $c_1, c_2, \cdots, c_K$ → intervals / C_i as indicator variables (1 or 0)

$y_i = \beta_0 + \beta_1 C_1 x_i + \beta_2 C_2 x_i + \cdots + \beta_K C_K x_i + \varepsilon_i$
$C_1 = 1$ if $c_0 < x < c_1$
$C_2 = 1$ if $c_1 < x < c_2$
⋮
$C_K = 1$ if $c_{K-1} < x < c_K$



*Regression Splines

① Knot selection (K knots)
② piecewise polynomial fitting → Cubic splines
$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \varepsilon_i$
- Add truncated power basis to knot

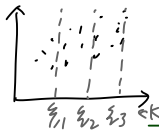
$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \sum_{k=1}^{K} \beta_{3+k} (x_i - \xi_k)_+^3 + \varepsilon_i$

$h(x, \xi) \equiv (x - \xi)_+^3 = \begin{cases} (x-\xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise} \end{cases}$

△ if number of knots ≅ n, model will overfit → Use Cross Validation

③ add constraints → natural cubic splines (smoother, boundry constraint to be linear)

*Smoothing splines: want RSS↓ + smooth function $\hat{f}$



$RSS = \sum_{i=1}^{n} (y_i - g(x_i))^2$
Loss
→ $RSS + \lambda \int g''(t)^2 dt$ penalty
tuning parameter (select: CV, AIC, BIC)

[λ small: fit data closely
λ large: smoother curve

ch8 Tree-based method

- Recursive split (partition) data to minimize ① sum of RSS ② sum of misclassification in new regions
→ branch, leaves, root node

- Steps
① split data
② Fit full tree → overfitting (down to 1 observ.)
③ Prune tree
④ Validate pruned tree: for each trees, use k-fold CV to estimate MSE
⑤ Average MSE for each k-folds
⑥ Select best tree: lowest MSE

*Pruning
- reduce overfitting by simplifying model
- Usually post-pruning: explore data thoroughly before decide which branch are redundant

Advantage	Disadvantage
Interpretable Close to human reasoning Nice Graph No need to create dummy variables	Not good at predicting (regression) high variance (sensitive to small change in data)

*Regression Tree (greedy algo) Quantitative

→ Select region R_k, predictor x_j, splitting point s → splitting R_k with criterion $x_j < s$ (max RSS↓)
- Stop splitting Rule: each region with 5↓ observations
- Overfitting problem if 1 point 1 region.

1. weakest link pruning $\frac{RSS(T_1) - RSS(T_0)}{|T_0| - |T_1|}$ ↓ (T_0: original tree)
2. cost complexity pruning $\left(\sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|\right)$ ↓ (choose by CV)

*Classification Tree Qualitative

→ minimize Loss Function
For growing tree
1. 0-1 loss / misclassification rate $\sum_{m=1}^{|T|} \sum_{i \in R_m} 1(y_i \neq \hat{y}_{R_m})$
For pruning tree
2. Gini Index $\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
3. Cross-entropy $-\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$

[$\hat{p}_{mk}$: proportion of class k with R_m
[q_m: proportion of samples in R_m.

*Bagging, Random Forest & Boosting

Bagging	Random Forest	Boosting	BART
variance↓ high correlated bias ↓x	variance ↓↓ correlated ↓ bias ↑ (random)	variance ↓ overfitting bias ↓ slow to train	capture complex non-linear relation slow to train

- Tree → High variance: small changes / partitions in data can have a big effect on fitted model

① Bagging (Bootstrap AGGregatING): variance↓ (subset with replacement)
→ Bootstraping: results in similar datasets → highly correlated trees $\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$

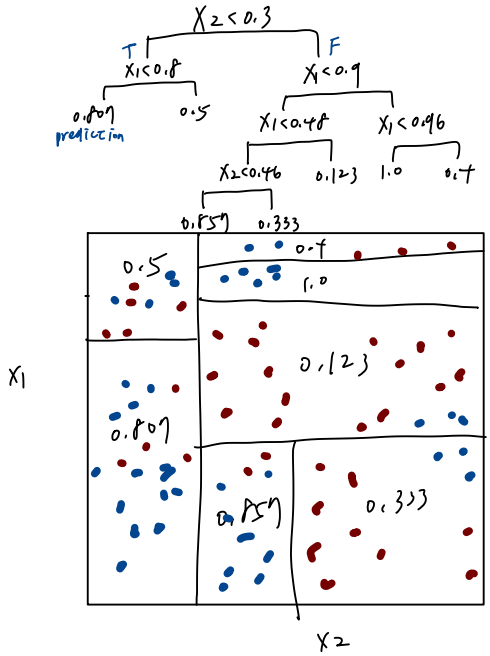
② Random forest
→ Bagging + restrict number of predictors, the algo can see when it splits (randomly choose m≤p predictors)
△ there are 1~2 predictors dominate splitting
∴ Allowing models to be fit without dominant variables: the set of B trees is decorrelated
averaging B trees (predictions), $\hat{f}_{rf}(X)$ will have lower variance

③ Boosting: regularization, bias↓
→ Adding shrinkage tree to every trees sequencially, correcting previous error

$\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^b(x)$ ← shrinkage tree
$r_i = r_i - \lambda \hat{f}^b(x)$ for all i (residual)
$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$ △ Use CV to select B (choose λ↓)

④ Bayesian Additive Regression Tree (BART)
→ grow trees like Bagging and Random Forest, also capture new info like boosting (random predictors)

$f_k^b(X)$ is the prediction at k^{th} regression tree (of K) and b^{th} iteration (of B)
$f_k^b(X) = \sum_{k=1}^{K} \hat{f}_k^b(X)$ for $b = 1, \cdots, B$



ch10 Deep Learning "structure" of the model

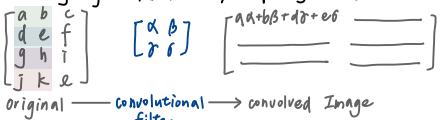
*Multiple Layer Neural Networks

- Input layer → [Layer 1]
- Hidden layer(s)
$A_k^{(1)} = h^{(1)}(X) = g(W_{k0}^{(1)} + \sum_{j=1}^{p} W_{kj}^{(1)} x_i)$ for $k = 1, \cdots, K_1$
[Layer 2]
$A_k^{(2)} = h^{(2)}(X) = g(W_{k0}^{(2)} + \sum_{j=1}^{K_1} W_{kk}^{(2)} x_k^{(1)})$ for $k = 1, \cdots, K_2$
results of activation
- Output layer $E_m = \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} h_l^{(2)}(X)$ last hidden layer
$= \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} A_l^{(2)}$
- Softmax $f_m(X) = P_r(Y = m|X) = \frac{e^{z_m}}{\sum_{l=1}^{K} e^{z_l}}$
Classifying so output is probability ↓ not $f_m(X) = E_m$

→ we need to have a decision rule. Usually, assign to the class with highest probability
To train, we want to find coefficient estimates such that we minimize cross entropy $-\sum_{i=1}^{n} \sum_{m=0}^{9} \log(f_m(x_i))$

*Convolutional Neural Networks (CNN)

① Convolutional Layer: Feature contraction
② Pooling Layer: classification, max pooling = max{each 2x2 subset of image}



original → convolutional filter → convolved Image

△ReLU
$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases}$

*Graphics

- data type [Quantitative : continuous, discrete
  Qualitatative : ordered, nominal (categories)

Other :
- Classification : Confusion Matrix, ROC Curve, Precision-recall curve
- Access model fitting : Residul plot

Histogram	proporcions distribution	hist( )	Quant
Density plot	smoothing histogram	plot(density( ))	Quant
Box plot	statistical info	boxplot( )	Qual + Quant
Rug plot	addicional distribution	plot(density( )) rug( )	Quant
Dot chart	comparision values	dotchart( )	Qual + Quant
Mosiac plot	proporcions across 2 variables	mosiacplot( )	Qual + Qual
Scatter plot	relationship between 2 var.	plot( )	Quant + Quant
Bar plot	frequency of category	barplot( )	Qual

```
abline( )  add line
points( )  add point
lines( )   add line segments
text( )    add texe
plot(type="l", p:points, l:lines, n:nothing
     ylim=c(0,1), range for scale of axis
     xlab="...", x axis label
     main="...", plot title
     col=c( ) vector of colors
     log="xy", use log scale on y/x/xy
par(mfrow=c(x,y))

grid(nx=NULL, ny=NULL, lwd=2)  add grid line
legend("topright", fill=colors, legend=c(A,B,C,D), title="—")

lwd=2, thickness of line
pch=19, dots type
cex=0.5, 50% smaller
lty=2, line type
las=1  style of tick mark labels
)
```

*R code

- Bootstrapping

```
library(boot)
set.seed(8436)   # mean & median estimation
results_mean <- boot(data, bootstrap_mean, R=1000)
results_median <- boot(data, bootstrap_median, R=1000)
#confidence interval
ci_mean <- boot.ci(results_mean, type="perc")
ci_median <- boot.ci(results_median, type="perc")
```

- Basic Neural Network

```
input_shape <- c(32,32,3)
model_basic <- keras_model_sequential( ) %>%
      layer_flatten(input_shape)
      layer_dense(units=512, activation='relu')
      layer_dense(units=10, activation='softmax')

model_basic %>% compile(
      loss='categorical_crossentropy',
      optimizer=optimizer_adam( ),
      metrics=c('acuracy')
)
history_basic <- model_basic %>% fit(
   X_train, y_train,
   epochs=10
   batch_size=64,
   validation_split=0.2
)
evaluation_results_basic <- model_basic %>% evaluate(X_test, y_test)


model_cnn <- keras_model_sequential( ) %>%
      layer_conv_2d(filters=32, kernal_size=c(3,3), activation='relu', input_shape)
      layer_max_pooling_2d(pool_size=c(2,2))
      layer_conv_2d(      =64,
      layer_max_pooling_2d(          )
      layer_conv_2d(      =64,
      layer_max_pooling_2d(          )
      layer_flatten( )
      layer_dense(64, 'relu')
      layer_dense(10,'soft_max')
```