Lecture 14 - Feb 26

Decision Tree Regressors
Bias-Variance Trade-off

Reading

Elements of Statistical Learning
9.2 Tree-Based Models
7.1-3 Bias-Variance Trade-off

Breiman, L. Bagging Predictors. *Machine Learning* **24**, 123–140 (1996). https://doi.org/10.1023/A:1018054314350

Good approach: grow a large tree T_0 and use cost-complexity pruning

Let $T \subset T_0$ be a subtree from pruning T_0 (collapsing internal nodes)

Index terminal nodes by $m \rightarrow R_m$

$|T|$ = number of terminal nodes

$N_m = |\{x_i | x_i \in R_m\}|$

$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$ ← mean

$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$ ← impurity

And complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

For each α, find $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$

$\alpha \geq 0$ hyperparameter → large α → smaller T_α; small α → larger T_α ($\alpha = 0 \Rightarrow T_0$)

For fixed α, $\exists$ unique T_α minimizing $C_\alpha(T)$

With weakest link pruning, we collapse the internal node producing the smallest per-node increase in $\sum_{m=1}^{|T|} N_m Q_m(T)$ one-by-one

With [illegible]
the smallest per-node increase in $\sum_{m=1}^{|T|}$ [illegible]

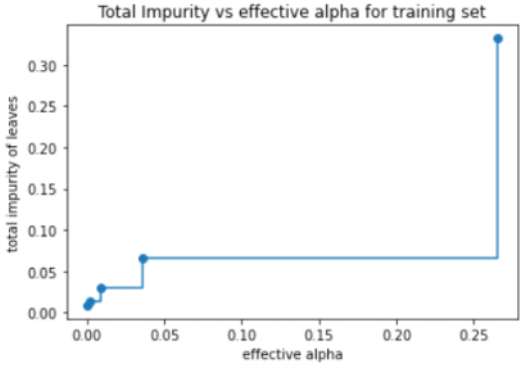
↳ $T_\alpha \in \{$subtrees produced in this process$\}$

Choose α by CV minimizing SSE mean

Pruning with the Iris Dataset

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# create the classifier
clf = DecisionTreeClassifier(random_state=0)

# do cost-complexity pruning
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# plotting the impurities for different alphas
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
```

```
Text(0.5, 1.0, 'Total Impurity vs effective alpha for training set')
```

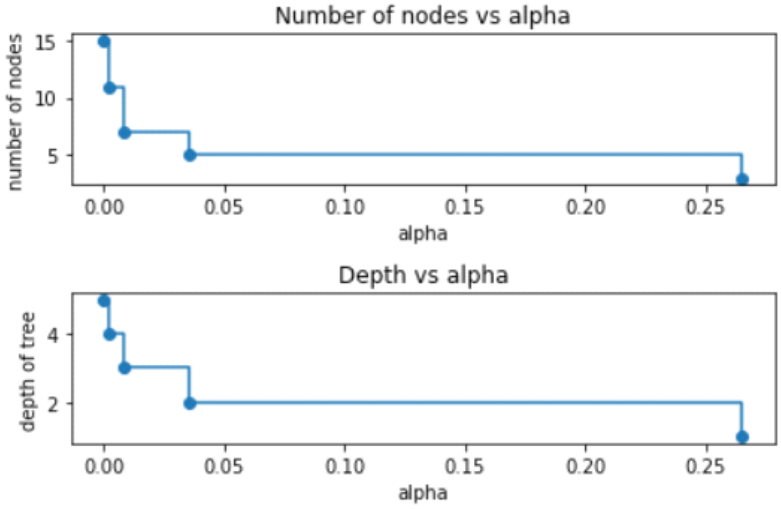


Next, we train a decision tree using the effective alphas. The last value in `ccp_alphas` is the alpha value that prunes the whole tree, leaving the tree, `clfs[-1]`, with one node.

```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

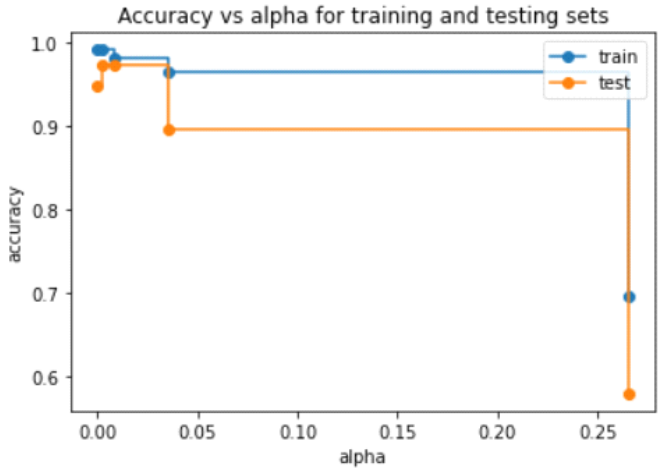
```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.332795493197279
```

```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1)
ax[0].plot(ccp_alphas, node_counts, marker='o', drawstyle="steps-post")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker='o', drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```

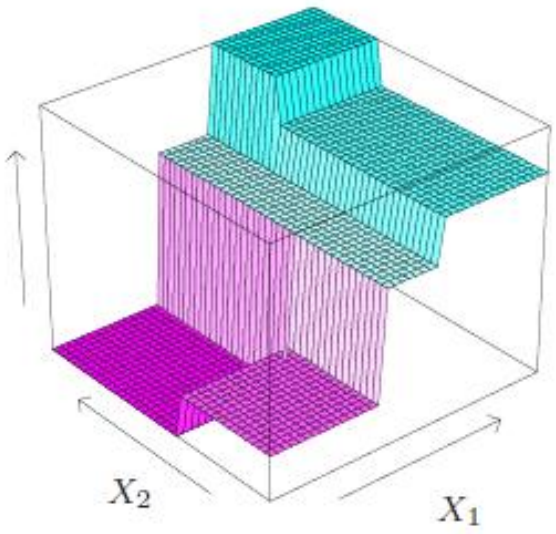


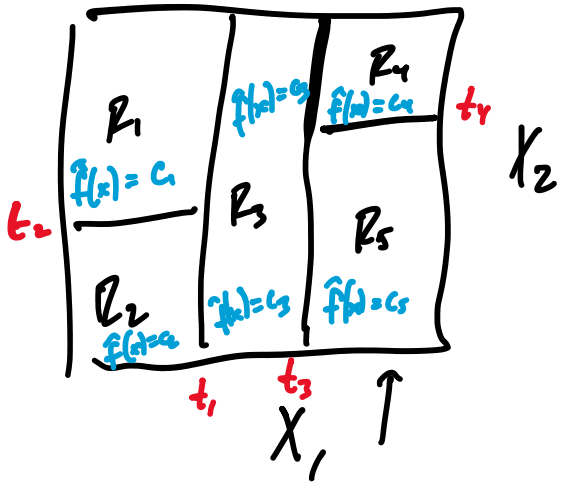
For the remainder of this example, we remove the last element in `clfs` and `ccp_alphas`, because it is the trivial tree with only one node. Here we show that the number of nodes and tree depth decreases as alpha increases.

```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



When `ccp_alpha` is set to zero and keeping the other default parameters of DecisionTreeClassifier, the tree overfits, leading to a 100% training accuracy and 88% testing accuracy. As alpha increases, more of the tree is pruned, thus creating a decision tree that generalizes better. In this example, setting `ccp_alpha = 0.015` maximizes the testing accuracy.





Model must choose splitting variables + points and then

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}_{\{x \in R_m\}}$$

If we try to minimize SSE, $\sum_{i=1}^{n} (y_i - f(x_i))^2$, then the best $\hat{c}_m$ is

$$\hat{c}_m = \text{mean}\left(\{y_i \mid x_i \in R_m\}\right)$$

Finding the optimal binary partition is infeasible computationally.

Greedy algorithms common.

Consider splitting X_j at v...

$$R_1(j,v) = \{X \mid X_j \le v\} \qquad R_2(j,v) = \{X \mid X_j > v\}$$

$R_1(j,v) = \{X | X_j \le v\}$ $\quad R_2(j,v) = \{X | X_j > v\}$

try to solve

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right]$$

choose c_1 to min SSE in R_1

choose c_2 to min SSE in R_2

choose j, s to minimize minimal SSE in 2 sides

$\hat{c}_1 = \text{mean}(\{y_i | x_i \in R_1(j,s)\})$ is optimal

$\hat{c}_2 = \text{mean}(\{y_i | x_i \in R_2(j,s)\})$ is optimal

- easy to do computationally

Then repeat splitting process in each region

Tree size is a hyperparameter → too high => overfit
too low => underfit

↑ choose adaptively from data

Pros of Decision Trees

Simple to understand/interpret rule sets

↳ "white box" model

Handles numerical + categorical features

Low computational cost

Built-in feature selection based on rulesets

Issues with Decision Trees (ESL)

- Splits not parallel to axes work but harm interpretability
- Good to use trees with missing data with a separate category for "missing" in categorical vars (additional tricks)
  - ↳ finds trends with these
- This was CART implementation. C4.5 + C5.0 are others → diff scheme to deriving rule sets
- Trees have an instability issue. Small changes in data can cause very different splits.

  (Bagging averages many trees)
- Prediction surfaces not smooth for regression

  (MARS helps)
- Learning optimal tree is NP-Complete
- Trees tend to overfit without pruning

7.1 Introduction

The *generalization* performance of a learning method relates to its prediction capability on independent test data. Assessment of this performance is extremely important in practice, since it guides the choice of learning method or model, and gives us a measure of the quality of the ultimately chosen model.

7.2 Bias, Variance and Model Complexity

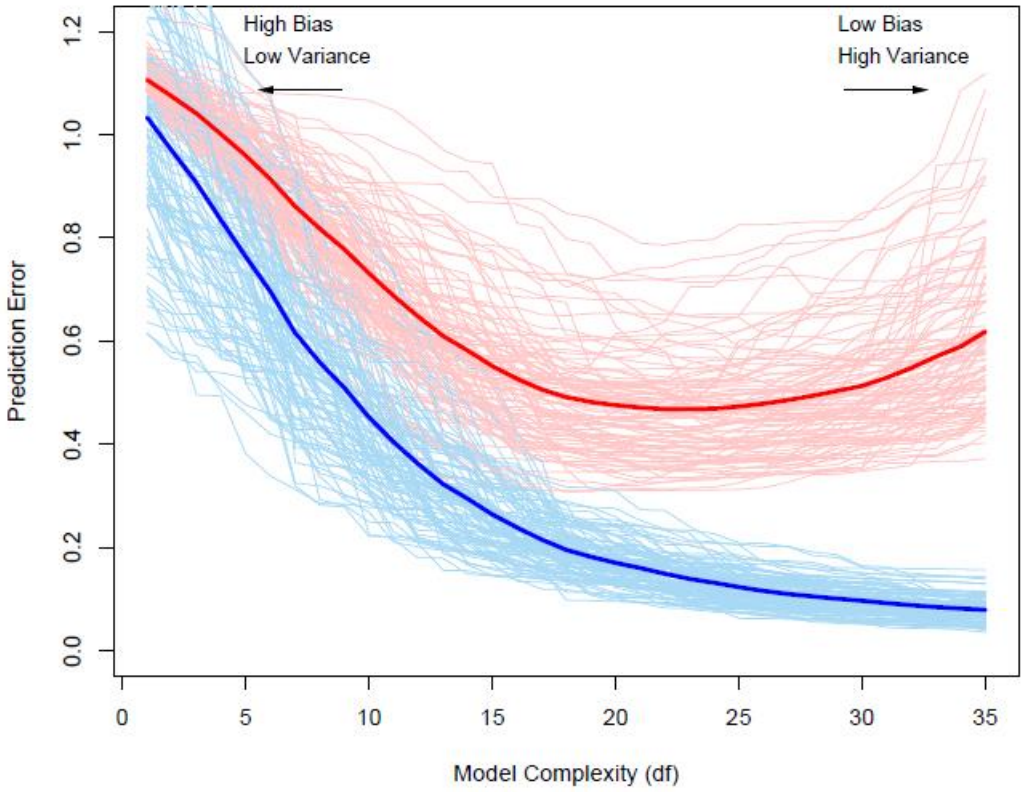


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$.

Test error, also referred to as *generalization error*, is the prediction error over an independent test sample

Test error, also referred to as *generalization error*, is the prediction error over an independent test sample

$$\mathrm{Err}_{\mathcal{T}} = \mathrm{E}[L(Y, \hat{f}(X))|\mathcal{T}] \tag{7.2}$$

where both X and Y are drawn randomly from their joint distribution (population). Here the training set $\mathcal{T}$ is fixed, and test error refers to the error for this specific training set. A related quantity is the expected prediction error (or expected test error)

$$\mathrm{Err} = \mathrm{E}[L(Y, \hat{f}(X))] = \mathrm{E}[\mathrm{Err}_{\mathcal{T}}]. \tag{7.3}$$

Note that this expectation averages over everything that is random, including the randomness in the training set that produced $\hat{f}$.

Figure 7.1 shows the prediction error (light red curves) $\mathrm{Err}_{\mathcal{T}}$ for 100 simulated training sets each of size 50. The lasso (Section 3.4.2) was used to produce the sequence of fits. The solid red curve is the average, and hence an estimate of Err.

Estimation of $\mathrm{Err}_{\mathcal{T}}$ will be our goal, although we will see that Err is more amenable to statistical analysis, and most methods effectively estimate the expected error. It does not seem possible to estimate conditional

error effectively, given only the information in the same training set. Some discussion of this point is given in Section 7.12.

Training error is the average loss over the training sample

$$\overline{\mathrm{err}} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)). \tag{7.4}$$

We would like to know the expected test error of our estimated model $\hat{f}$. As the model becomes more and more complex, it uses the training data more and is able to adapt to more complicated underlying structures. Hence there is a decrease in bias but an increase in variance. There is some intermediate model complexity that gives minimum expected test error.

Unfortunately training error is not a good estimate of the test error, as seen in Figure 7.1. Training error consistently decreases with model complexity, typically dropping to zero if we increase the model complexity enough. However, a model with zero training error is overfit to the training data and will typically generalize poorly.

Classification

Again, test error here is $\text{Err}_{\mathcal{T}} = \text{E}[L(G, \hat{G}(X))|\mathcal{T}]$, the population misclassification error of the classifier trained on $\mathcal{T}$, and Err is the expected misclassification error.

Training error is the sample analogue, for example,

$$\overline{\text{err}} = -\frac{2}{N}\sum_{i=1}^{N} \log \hat{p}_{g_i}(x_i), \tag{7.7}$$

the sample log-likelihood for the model.

The log-likelihood can be used as a loss-function for general response densities, such as the Poisson, gamma, exponential, log-normal and others. If $\Pr_{\theta(X)}(Y)$ is the density of Y, indexed by a parameter $\theta(X)$ that depends on the predictor X, then

$$L(Y, \theta(X)) = -2 \cdot \log \Pr_{\theta(X)}(Y). \tag{7.8}$$

The "-2" in the definition makes the log-likelihood loss for the Gaussian distribution match squared-error loss.

For ease of exposition, for the remainder of this chapter we will use Y and $f(X)$ to represent all of the above situations, since we focus mainly on the quantitative response (squared-error loss) setting. For the other situations, the appropriate translations are obvious.

Bias-Variance Decomposition

As in Chapter 2, if we assume that $Y = f(X) + \varepsilon$ where $\mathrm{E}(\varepsilon) = 0$ and $\mathrm{Var}(\varepsilon) = \sigma_\varepsilon^2$, we can derive an expression for the expected prediction error of a regression fit $\hat{f}(X)$ at an input point $X = x_0$, using squared-error loss:

$$\begin{aligned}
\mathrm{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] \\
&= \sigma_\varepsilon^2 + [\mathrm{E}\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - \mathrm{E}\hat{f}(x_0)]^2 \\
&= \sigma_\varepsilon^2 + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\hat{f}(x_0)) \\
&= \text{Irreducible Error} + \mathrm{Bias}^2 + \text{Variance}. \qquad (7.9)
\end{aligned}$$

The first term is the variance of the target around its true mean $f(x_0)$, and cannot be avoided no matter how well we estimate $f(x_0)$, unless $\sigma_\varepsilon^2 = 0$. The second term is the squared bias, the amount by which the average of our estimate differs from the true mean; the last term is the variance; the expected squared deviation of $\hat{f}(x_0)$ around its mean. Typically the more complex we make the model $\hat{f}$, the lower the (squared) bias but the higher the variance.

For the k-nearest-neighbor regression fit, these expressions have the simple form

$$\begin{aligned}
\mathrm{Err}(x_0) &= E[(Y - \hat{f}_k(x_0))^2 | X = x_0] \\
&= \sigma_\varepsilon^2 + \left[f(x_0) - \frac{1}{k} \sum_{\ell=1}^{k} f(x_{(\ell)}) \right]^2 + \frac{\sigma_\varepsilon^2}{k}. \qquad (7.10)
\end{aligned}$$

Here we assume for simplicity that training inputs x_i are fixed, and the randomness arises from the y_i. The number of neighbors k is inversely related to the model complexity. For small k, the estimate $\hat{f}_k(x)$ can potentially adapt itself better to the underlying $f(x)$. As we increase k, the bias—the squared difference between $f(x_0)$ and the average of $f(x)$ at the k-nearest neighbors—will typically increase, while the variance decreases.

For a linear model fit $\hat{f}_p(x) = x^T \hat{\beta}$, where the parameter vector β with p components is fit by least squares, we have

$$\mathrm{Err}(x_0) = E[(Y - \hat{f}_p(x_0))^2 | X = x_0]$$

$$= \quad \sigma_\varepsilon^2 + [f(x_0) - \mathrm{E}\hat{f}_p(x_0)]^2 + ||\mathbf{h}(x_0)||^2\sigma_\varepsilon^2. \qquad (7.11)$$

Here $\mathbf{h}(x_0) = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}x_0$, the N-vector of linear weights that produce the fit $\hat{f}_p(x_0) = {x_0}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, and hence $\mathrm{Var}[\hat{f}_p(x_0)] = ||\mathbf{h}(x_0)||^2\sigma_\varepsilon^2$. While this variance changes with x_0, its average (with x_0 taken to be each of the sample values x_i) is $(p/N)\sigma_\varepsilon^2$, and hence

$$\frac{1}{N}\sum_{i=1}^{N}\mathrm{Err}(x_i) = \sigma_\varepsilon^2 + \frac{1}{N}\sum_{i=1}^{N}[f(x_i) - \mathrm{E}\hat{f}(x_i)]^2 + \frac{p}{N}\sigma_\varepsilon^2, \qquad (7.12)$$

the *in-sample* error. Here model complexity is directly related to the number of parameters p.

The test error $\mathrm{Err}(x_0)$ for a ridge regression fit $\hat{f}_\alpha(x_0)$ is identical in form to (7.11), except the linear weights in the variance term are different: $\mathbf{h}(x_0) = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})^T x_0$. The bias term will also be different.

For a linear model family such as ridge regression, we can break down the bias more finely. Let β_* denote the parameters of the best-fitting linear approximation to f:

$$\beta_* = \arg\min_\beta \mathrm{E}\left(f(X) - X^T\beta\right)^2. \tag{7.13}$$

Here the expectation is taken with respect to the distribution of the input variables X. Then we can write the average squared bias as

$$\begin{aligned} \mathrm{E}_{x_0}\left[f(x_0) - \mathrm{E}\hat{f}_\alpha(x_0)\right]^2 &= \mathrm{E}_{x_0}\left[f(x_0) - x_0^T\beta_*\right]^2 + \mathrm{E}_{x_0}\left[x_0^T\beta_* - \mathrm{E}x_0^T\hat{\beta}_\alpha\right]^2 \\ &= \mathrm{Ave}[\text{Model Bias}]^2 + \mathrm{Ave}[\text{Estimation Bias}]^2 \end{aligned} \tag{7.14}$$

The first term on the right-hand side is the average squared *model bias*, the error between the best-fitting linear approximation and the true function. The second term is the average squared *estimation bias*, the error between the average estimate $\mathrm{E}(x_0^T\hat{\beta})$ and the best-fitting linear approximation.

For linear models fit by ordinary least squares, the estimation bias is zero. For restricted fits, such as ridge regression, it is positive, and we trade it off with the benefits of a reduced variance. The model bias can only be reduced by enlarging the class of linear models to a richer collection of models, by including interactions and transformations of the variables in the model.

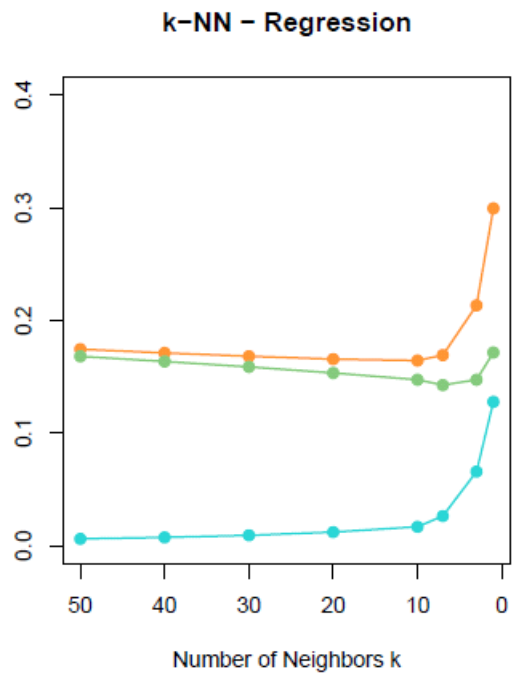
7.3.1 Example: Bias–Variance Tradeoff

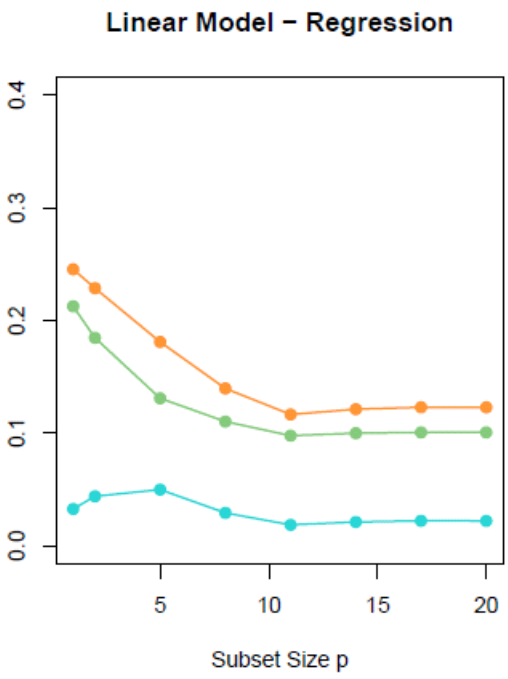
Figure 7.3 shows the bias–variance tradeoff for two simulated examples. There are 80 observations and 20 predictors, uniformly distributed in the hypercube $[0, 1]^{20}$. The situations are as follows:

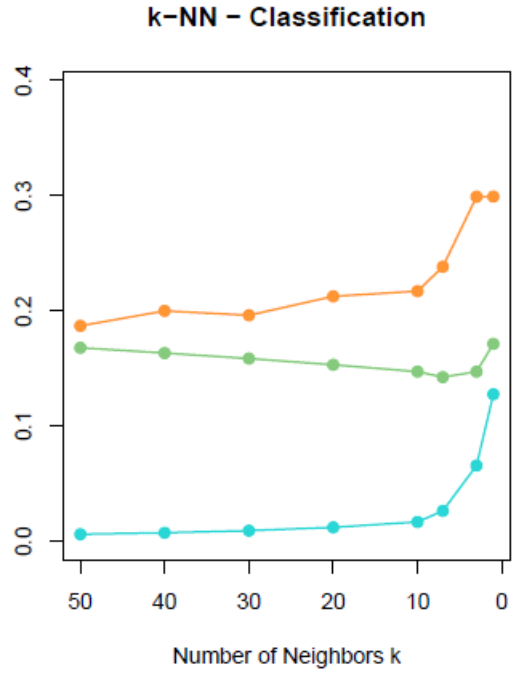
Left panels: Y is 0 if $X_1 \leq 1/2$ and 1 if $X_1 > 1/2$, and we apply k-nearest neighbors.

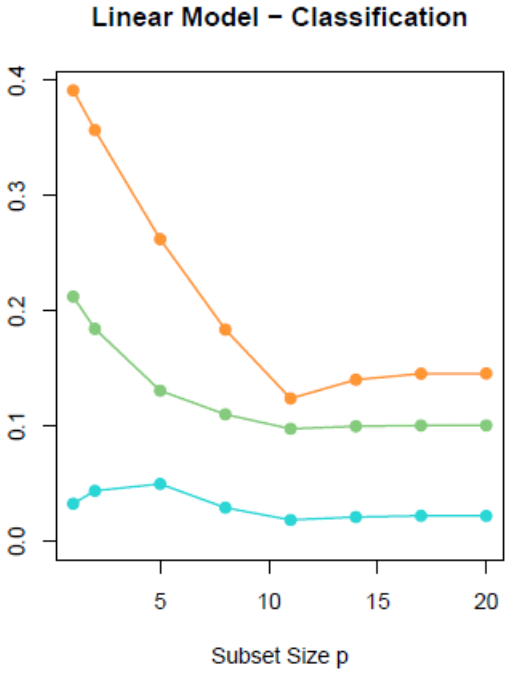
Right panels: Y is 1 if $\sum_{j=1}^{10} X_j$ is greater than 5 and 0 otherwise, and we use best subset linear regression of size p.

The top row is regression with squared error loss; the bottom row is classification with 0–1 loss. The figures show the prediction error (red), squared bias (green) and variance (blue), all computed for a large test sample.


k−NN − Regression
0.0
0.1
0.2
0.3
0.4
50
40
30
20
10
0
Number of Neighbors k


Linear Model − Regression
0.0
0.1
0.2
0.3
0.4
5
10
15
20
Subset Size p


k−NN − Classification
0.0
0.1
0.2
0.3
0.4
50
40
30
20
10
0
Number of Neighbors k


Linear Model − Classification
0.0
0.1
0.2
0.3
0.4
5
10
15
20
Subset Size p