STA314 Summer 2023

Week 4: SVM Classifier & Decision tree

Ziang Zhang

Department of Statistics, University of Toronto

Overview

Hyperplane

In a p-dimensional (feature) space, a **hyperplane** is defined as:

$$\{\boldsymbol{x} \in \mathbb{R}^p : \beta_0 + \boldsymbol{\beta}^T \boldsymbol{x} = 0\},$$

where $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)^T$.

When $p = 1$, the hyperplane is a point

$$\{x \in \mathbb{R} : \beta_0 + \beta_1 x = 0\} = \{-\beta_0/\beta_1\},$$

When $p = 2$, the hyperplane is a line

$$\{\mathbf{x} \in \mathbb{R}^2 : \beta_0+\beta_1 x_1+\beta_2 x_2 = 0\} = \{(x_1, x_2) \in \mathbb{R}^2 : x_2 = -\frac{\beta_0}{\beta_2} - \frac{\beta_1}{\beta_2} x_1\}.$$

Hyperplane

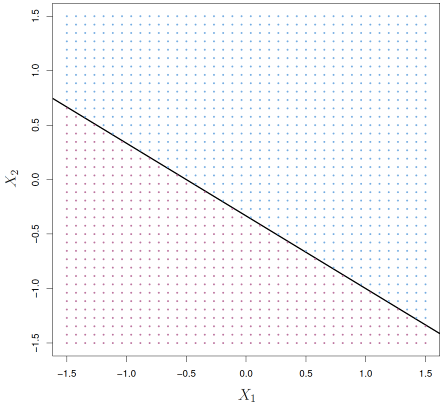

FIGURE 9.1. *The hyperplane* $1 + 2X_1 + 3X_2 = 0$ *is shown. The blue region is the set of points for which* $1 + 2X_1 + 3X_2 > 0$*, and the purple region is the set of points for which* $1 + 2X_1 + 3X_2 < 0$.

Classification using a hyperplane

The hyperplane separates $\mathbb{R}^p$ into two halves:

- $\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x} > 0$
- $\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x} < 0$

We can construct a classifier based on the hyperplane:

- $y(\boldsymbol{x}_i) = 1$ if $\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i > 0$
- $y(\boldsymbol{x}_i) = -1$ if $\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i < 0$

Which implies that $y(\boldsymbol{x}_i)(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i) > 0 \; \forall i \in [n]$.

The problem is then to find the best-separating plane $(\beta_0, \boldsymbol{\beta})$.

Separating hyperplane

For now, assume we have a training set $\{y_i, \boldsymbol{x}_i\}_{i=1}^n$ that can be **linearly separated**.
That means there exists a set of β_0 and $\boldsymbol{\beta}$ such that:

$$y_i(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i) > 0, \text{ for } i = 1, ..., n.$$

However, if a such separating hyperplane exists, there must exist **infinitely** many other separating hyperplanes for this dataset... (*why?*)

Separating hyperplane

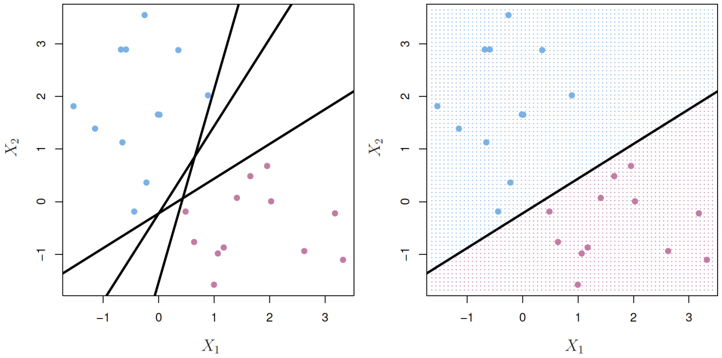

FIGURE 9.2. Left: *There are two classes of observations, shown in blue and in purple, each of which has measurements on two variables. Three separating hyperplanes, out of many possible, are shown in black.* Right: *A separating hyperplane is shown in black. The blue and purple grid indicates the decision rule made by a classifier based on this separating hyperplane: a test observation that falls in the blue portion of the grid will be assigned to the blue class, and a test observation that falls into the purple portion of the grid will be assigned to the purple class.*

The Maximal Margin Classifier: Overview

Given a set of linearly separable training data, the **Maximal Margin Classifier** uses a separating hyperplane in the feature space with the *maximum margin*.

- The *margin* is defined as the smallest distance from the observations (data points) to the hyperplane.
- This is the simplest case of the support vector machine (SVM) classifier, which we will discuss later.

The Maximal Margin Classifier: Construction

Mathematically, the optimization problem can be written as:

$$\begin{aligned} \text{maximize}_{\beta_0, \boldsymbol{\beta}, M} \quad & M \geq 0 \\ \text{subject to} \quad & ||\boldsymbol{\beta}||^2 = \sum_{i=1}^{p} \beta_i^2 = 1 \\ & y_i(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i) \geq M \quad \forall i \in [n] \end{aligned}$$

This optimization problem ensures that:

- All observations lie on the correct side of the hyperplane.
- M denotes the smallest distance from the observations to the hyperplane, which is the margin.
- With the constraint on $||\boldsymbol{\beta}||^2 = 1$, $y_i(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i)$ is the distance from $\boldsymbol{x}_i \in \mathbb{R}^p$ to the hyperplane $\{\boldsymbol{x} \in \mathbb{R}^p : \beta_0 + \boldsymbol{\beta}^T \boldsymbol{x} = 0\}$.
- The optimization can be done efficiently, but its details are outside of the scope.

The Maximal Margin Classifier: Example

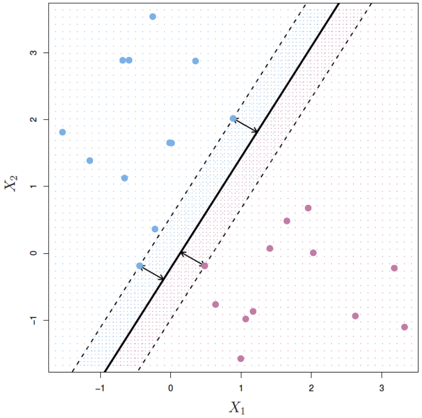

FIGURE 9.3. *There are two classes of observations, shown in blue and in purple. The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the support vectors, and the distance from those points to the hyperplane is indicated by arrows. The purple and blue grid indicates the decision rule made by a classifier based on this separating hyperplane.*

Non-separable data

- Unfortunately, for most interesting dataset, the outcome $\mathbf{y}$ will not be linearly separable.
- In other words, there **does not exist any separating hyperplane** in the training data.
- This implies the optimization problem before has **no solution** with $M > 0$.

Non-separable data

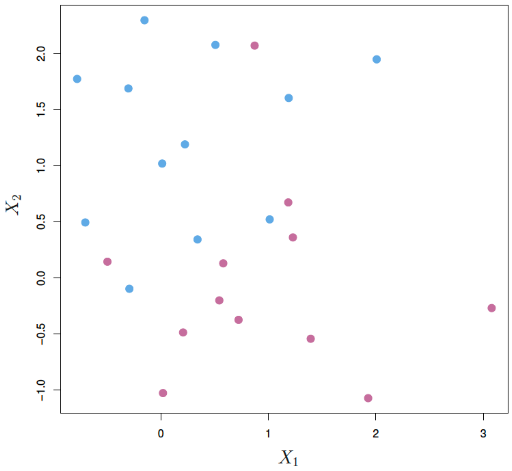

FIGURE 9.4. *There are two classes of observations, shown in blue and in purple. In this case, the two classes are not separable by a hyperplane, and so the maximal margin classifier cannot be used.*

Support Vector Classifier

The **support vector classifier**(SVC) can be viewed as a generalization of the maximal marginal classifier, that accommodate non-separable data.

- Similar to the maximal marginal classifier, SVC also constructs a hyperplane with a set of margins.
- Rather than restrict **all** the observations in the correct side of the margin and hyperplane. SVC allows **some** observations to be on the incorrect side of the margin, or even the hyperplane.
- Unlike the maximal marginal classifier, the margin of the SVC is *soft*. It is also called the *soft margin classifier*.

Construction of the SVC

Besides the original parameters $\beta_0, \boldsymbol{\beta}$ and M, SVC introduces the additional parameters $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)$ and C:

$$\begin{aligned} \text{maximize}_{\beta_0, \boldsymbol{\beta}, M, \boldsymbol{\epsilon}} \quad & M \geq 0 \\ \text{subject to} \quad & ||\boldsymbol{\beta}||^2 = \sum_{i=1}^{p} \beta_i^2 = 1 \\ & y_i(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i) \geq M(1 - \epsilon_i), \quad \forall i \in [n] \\ & \epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq C. \end{aligned}$$

- Each variable ϵ_i is a *slack variable* that allows the ith observation to be on the wrong side of the margin (if $\epsilon_i > 0$), or even the wrong side of the hyperplane (if $\epsilon_i > 1$).
- C is a positive tuning parameter, called the *budget*.

Alternative expression of the SVC

The previous optimization problem can be equivalently rewritten as:

$$\text{minimize}_{\beta_0,\boldsymbol{\beta}} \quad \left[\sum_{i=1}^{n} \max\left[0, 1 - y_i f(\mathbf{x}_i)\right] + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}\right].$$

- The derivation detail is beyond the scope
- $f(\mathbf{x}_i) = \beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i$
- λ is a penalty parameter, that is proportional to C.
- This loss function called the (regularized) hinge loss

Illustration

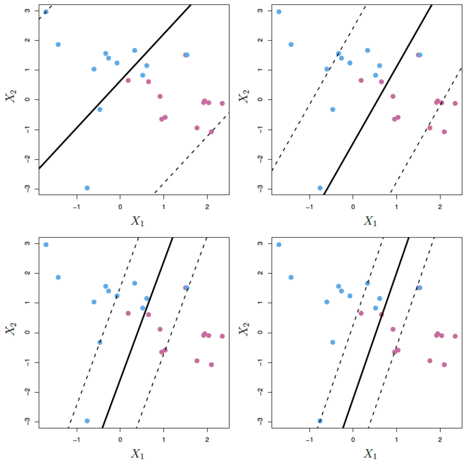

FIGURE 9.7. *A support vector classifier was fit using four different values of the tuning parameter C in (9.12)–(9.15). The largest value of C was used in the top left panel, and smaller values were used in the top right, bottom left, and bottom right panels. When C is large, then there is a high tolerance for observations being on the wrong side of the margin, and so the margin will be large. As C decreases, the tolerance for observations being on the wrong side of the margin decreases, and the margin narrows.*

Bias variance tradeoff in SVC

The hyperparameter C controls the bias-variance trade-off.

- As C decreases, the SVC becomes highly fit to the training data, which lowers the bias but increases the variance.
- As C increases, the margin is wider and more violations are allowed, which increases the bias but lowers the variance.
- If $C = 0$, SVC reduces back to the maximal marginal classifier.

Properties of the SVC

The optimization problem of the SVC only depends on a few *support vectors* (SVs):

- The SVs are those observations that either lie on the margin or violate the margin.
- For observations on the correct side of the margin, changing their positions within the correct side will *not change* the classifier.
- In other words, the hyperplane parameters $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}$ are computed using the SVs only.

An alternative view of the SVC

When the budget C is large:

- The margin will be wider, with more support vectors to determine the hyperplane.
- This reduces the variance of the hyperplane.
- However, there will be more support vectors that violate the margin or the hyperplane.
- This likely increases the bias.

Illustration of SVs

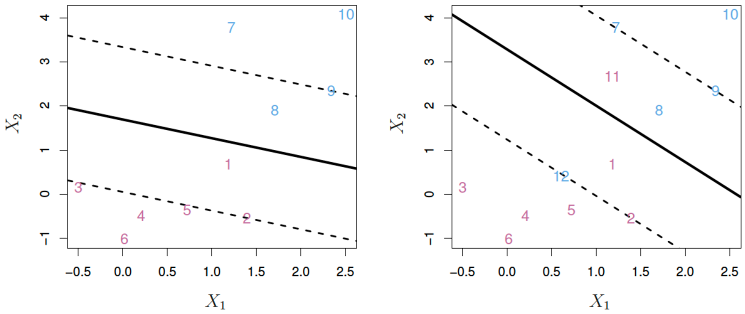

FIGURE 9.6. Left: *A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines.* Purple observations: *Observations* 3, 4, 5, *and* 6 *are on the correct side of the margin, observation* 2 *is on the margin, and observation 1 is on the wrong side of the margin.* Blue observations: *Observations* 7 *and* 10 *are on the correct side of the margin, observation* 9 *is on the margin, and observation* 8 *is on the wrong side of the margin. No observations are on the wrong side of the hyperplane.* Right: *Same as left panel with two additional points,* 11 *and* 12*. These two observations are on the wrong side of the hyperplane and the wrong side of the margin.*

Non-linear decision boundary

- The SVC classifies a new observation $\mathbf{x}_{new}$ by the sign of $f(\mathbf{x}_{new}) = \hat{\beta}_0 + \mathbf{x}_{new}^T \hat{\boldsymbol{\beta}}$.
- In other words, the classifier has a linear decision boundary.
- But in real data, such linear decision boundary is often not flexible enough.

Failue of linear classifier

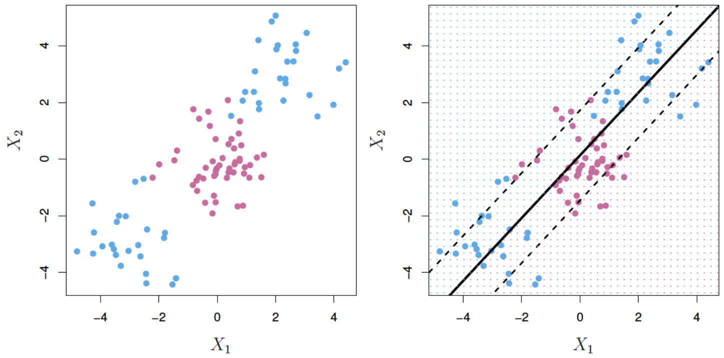

FIGURE 9.8. Left: *The observations fall into two classes, with a non-linear boundary between them.* Right: *The support vector classifier seeks a linear boundary, and consequently performs very poorly.*

Enlarging the feature space

- Like in regression problem, one simple way to deal with nonlinearity is through **the feature mapping.**
- This approach maps each original feature vector $\boldsymbol{x}_i \in \mathbb{R}^p$ to its enlarged version $\tilde{\boldsymbol{x}}_i \in \mathbb{R}^{\tilde{p}}$ where $\tilde{p} \geq p$.
- For example, if $\boldsymbol{x}_i = (x_{i1}, x_{i2}) \in \mathbb{R}^2$, the enlarged feature could be $\tilde{\boldsymbol{x}}_i = (x_{i1}, x_{i2}, x_{i1}^2, x_{i2}^2, x_{i1}x_{i2}) \in \mathbb{R}^5$.
- Then a new linear model can be fitted using the new feature vector:

$$f(\mathbf{x}_{new}) = \tilde{f}(\tilde{\mathbf{x}}_{new}) = \hat{\beta}_0^* + \tilde{\mathbf{x}}_{new}^T \hat{\boldsymbol{\beta}}^*.$$

- f is not linear, but $\tilde{f}$ is.

This approach could work for simple problem, but as $\tilde{p}$ gets too large, it becomes computationally infeasible.

Support Vector Machine (SVM) Classifier

The support vector machine (SVM) classifier is an extension of the SVC, to accommodate non-linear boundary between classes.

Rather than directly expanding the feature vector, SVM method relies on the *kernel trick* that generalizes the notion of the *inner product*.

The resulting model can be computed very efficiently, even when the corresponding enlarged feature vector is infinite-dimensional (i.e. $\tilde{\boldsymbol{x}} \in \mathbb{R}^\infty$).

Review of inner-product

- **Definition**: The inner product of two vectors $\mathbf{x} \in \mathbb{R}^p$ and $\mathbf{y} \in \mathbb{R}^p$ is denoted as $\langle \mathbf{x}, \mathbf{y} \rangle$.
- In $\mathbb{R}^p$, the default inner product is the dot product

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_{i=1}^{p} x_i y_i.$$

- More generally, inner-product is a mapping with the following three **properties**:
  - **Symmetry**: $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$
  - **Linearity**: $\langle a\mathbf{x} + b\mathbf{z}, \mathbf{y} \rangle = a\langle \mathbf{x}, \mathbf{y} \rangle + b\langle \mathbf{z}, \mathbf{y} \rangle$
  - **Positive-definite**: $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ with equality only if $\mathbf{x} = 0$

Kernel function

A **kernel** function $K(.,.)$ is a generalization of the notion of inner-product, that only requires two properties:

1. **Symmetry**: $K(\mathbf{x},\mathbf{y}) = K(\mathbf{y},\mathbf{x})$
2. **Positive-semi-definite**: $\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(\mathbf{x}_i,\mathbf{x}_j) \geq 0$, for any $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^p, c_1, ..., c_n \in \mathbb{R}$.

A *necessary condition* for PSD to hold: $K(\mathbf{x},\mathbf{x}) \geq 0$ for any $\mathbf{x}$.

Some examples of kernel:

- Polynomial kernel of degree $d > 1, d \in \mathbb{Z}^+$:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^d$$

- Radial kernel:

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{y}||^2}{2\sigma^2}\right)$$

SVM Classifier

For now, let's denote $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y}$. The solution of the SVC can always be written as:

$$f(\mathbf{x}) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle,$$

where $\mathcal{S}$ is the set of indices of the *support vectors*.

- The estimated parameters $\{\hat{\beta}_0, \hat{\boldsymbol{\alpha}}\}$ only involve the dot product between feature vectors.
- The SVM classifier replaces all the dot products $\langle \mathbf{x}, \mathbf{y} \rangle$ in f and in $\{\hat{\beta}_0, \hat{\boldsymbol{\alpha}}\}$, with the kernel $K(\mathbf{x}, \mathbf{y})$.

This results in a classifier with form:

$$f(\mathbf{x}) = \tilde{\beta}_0 + \sum_{i \in \tilde{\mathcal{S}}} \tilde{\alpha}_i K(\mathbf{x}_i, \mathbf{x}).$$

How does the kernel trick work?

To understand why the SVM classifier achieves its flexibility through the kernel trick, the *Mercer's theorem* becomes useful.

Theorem (Mercer's theorem)

Let K be any symmetric, positive-semi-definite kernel function, then $K(x, y)$ can be expressed as:

$$K(x, y) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x)\phi_i(y),$$

where $\lambda_i \geq 0$, and ϕ_i are the orthonormal functions corresponding to the λ_i, the eigenvalues of K.

Question: What does $\sum_{i=1}^{\infty} \lambda_i \phi_i(x)\phi_i(y)$ look like?

How does the kernel trick work?

The Mercer's theorem implies that the kernel can be written as the inner (dot) product of some transformed features:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{\Phi}(\mathbf{x}_1), \mathbf{\Phi}(\mathbf{x}_2) \rangle.$$

- By just working with $K(.,.)$ instead of $\langle .,. \rangle$, we *implicitly* map our feature vector $\mathbf{x} \in \mathbb{R}^p$ to $\Phi(\mathbf{x}) \in \mathbb{R}^{\tilde{p}}$
- The dimension $\tilde{p}$ could be much larger (often $\tilde{p} = \infty$).
- We don't need to compute the feature mapping by ourself; we don't even need to know what it is.
- When $\tilde{p} = \infty$, *direct* feature mapping will take forever.

Examples

Suppose $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (\boldsymbol{x}^T \boldsymbol{y})^2 \\ &= (x_1 y_1 + x_2 y_2)^2 \\ &= 2x_1 x_2 y_1 y_2 + (x_1 y_1)^2 + (x_2 y_2)^2 \end{aligned}$$

The corresponding feature mapping, $\Phi(\mathbf{x})$ is:

$$\Phi(\mathbf{x}) = (\sqrt{2}x_1 x_2, x_1^2, x_2^2)^T \in \mathbb{R}^3.$$

For radial kernel, the mapping $\Phi(\mathbf{x})$ is infinite dimensional!

Example:

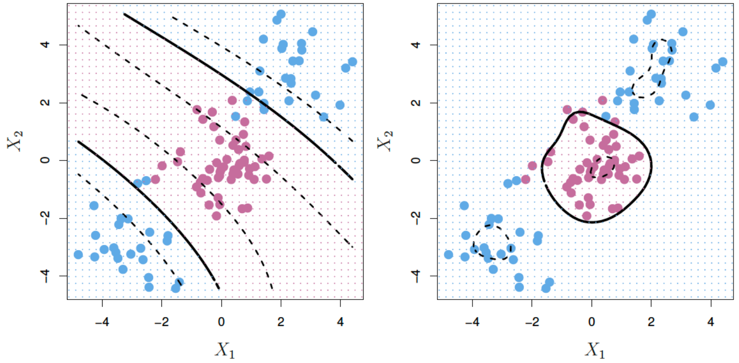

FIGURE 9.9. Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

Main idea of tree-based method:

Tree-based method *segments* or *splits* the feature space into a number of simple regions.

Within a region, the mean (*typical for regression*) or mode (*typical for classification*) of the training outcomes will be used as the prediction.

The set of splitting rules to segment the feature space can be summarized in a tree, hence called the **decision tree method**.

This may sound like the KNN algorithm, but they are different.

Example of regression tree

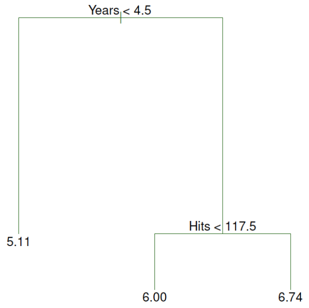

FIGURE 8.1. *For the* Hitters *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to* `Years<4.5`*, and the right-hand branch corresponds to* `Years>=4.5`*. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.*

Example of regression tree

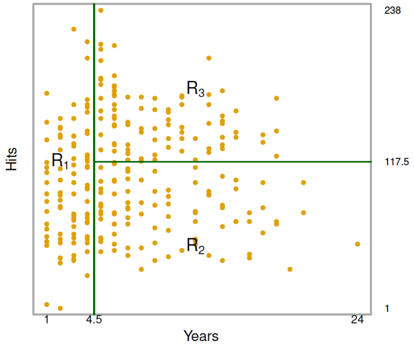

FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

Example continued:

- The tree segments the feature space into three separate regions R_1, R_2 and R_3, such that $R_1 \cup R_2 \cup R_3 = \mathbb{R}^2$.
- The regions can be written as $R_1 = \{\mathbf{x} \in \mathbb{R}^2 : \texttt{year} < 4.5\}$, $R_2 = \{\mathbf{x} \in \mathbb{R}^2 : \texttt{year} \geq 4.5, \texttt{Hits} < 117.5\}$ and $R_3 = \{\mathbf{x} \in \mathbb{R}^2 : \texttt{year} \geq 4.5, \texttt{Hits} \geq 117.5\}$.
- These are called the *leaves* or *terminal nodes* of the tree.
- The points on the tree where the feature space splits, are called *internal nodes* (i.e: $\texttt{Hits} < 117.5$ and $\texttt{Years} < 4.5$).
- Regression tree then predicts based on the training outcomes in each terminal node, for example by:

$$\hat{y}(\mathbf{x}_0) = \frac{\sum_{i=1}^{n} y_i I(\mathbf{x}_i \in R_j)}{\sum_{l=1}^{n} I(\mathbf{x}_l \in R_j)} = \frac{\sum_{i \in R_j} y_i}{\sum_{l=1}^{n} I(\mathbf{x}_l \in R_j)}, \text{ if } \mathbf{x}_0 \in R_j.$$

Example continued:

- The *root node* of the tree is the starting point before its first split (with no parent node, just children node).
- The *depth* of a node is defined as the number of edges needed from the root node to this node.
- The *tree depth* is the **maximum** depth of all the nodes in the tree.
- The *node impurity* measures the dissimilarity between observations in the same node.
- In regression problem, node impurity is typically defined as the within node RSS.

Training the regression tree

The problem now becomes given as set of training observations $\{y_i, \mathbf{x}_i\}_{i=1}^n$, how do you fit the regression tree model?

- Given the number of terminal nodes J. the problem is to find the best partition $\{R_1, \cdots, R_J\}$.
- One natural criterion is the RSS:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

  where $\hat{y}_{R_j}$ is the prediction from terminal node R_j.
- How many partition $\{R_1, \cdots, R_J\}$ do we need to consider?

Training the regression tree

- It is computationally infeasible to consider all the partitions.
- Therefore, we consider a *greedy* algorithm to find an *approximation* to the best partition, called **recursive binary splitting**.
- A greedy algorithm optimizes only the *next step* of your algorithm from your current position. It does not care about the long future.
- Because of its limited vision, such algorithm generally could not find the optimal solution, but it is very fast to run.
- *Question: What's another method that we have learned from this course, which uses the greedy algorithm?*

Recursive binary splitting

Here's how recursive binary splitting works:

1. For a given predictor X_j and a given value s of X_j, we define the pair of half-planes:

$$R_1(j,s) = \{\mathbf{x} | X_j < s\} \quad \text{and} \quad R_2(j,s) = \{\mathbf{x} | X_j \geq s\}$$

2. We then seek the value of j and s that minimize:

$$\sum_{i:\mathbf{x}_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\mathbf{x}_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

3. With the best cutpoint (j, s), partition the data into the two regions and repeat the process for each of them.
4. We do this until a stopping criterion (such as the node impurity, or the number of observations in each terminal node) is reached.

Complexity of the tree

The above fitting procedure only considers the training performance of the tree.

Unless the size of the tree (such as number of leaves J) is used as a stopping criterion, the fitted tree is likely too large.

Large tree tends to have small training error, but likely overfits the data.

Eventually the size of the tree controls the bias variance tradeoff.

Pruning of the tree

One strategy to deal with overfitting in tree method is through the idea of pruning:

- Grow a very large tree T_0.
- *Prune* the tree *back* to a smaller *subtree.*
- Select the optimal subtree based on its testing performance with the CV method.

It is often too hard to consider all the possible subtrees.

Cost Complexity Pruning

Cost Complexity Pruning, also called *weakest link pruning* gives us an approximation to the optimal subtree.

1. Starts with a huge tree T_0.
2. Rather than considering every possible subtree, we consider a sequence of trees indexed by hyperparameter $\alpha \geq 0$.
3. For each value of α there is a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\mathbf{x}_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

   is minimized.
4. Here $|T|$ denotes the number of terminal nodes of T.

Building a Regression Tree

A regression tree can be trained and pruned as following:

1. Use recursive binary splitting to grow a large tree on the training data, stop when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α that minimizes the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Classification tree

For regression tree, $\hat{y}$ is computed as the mean outcomes of the training data in a terminal node.

A decision tree can also be used for classification problem, where $\hat{y}$ is the *most commonly occurring class* in that node.

The most commonly occurring class is also called the *majority class* or the *dominant class*.

Classification Error

Just as in the regression setting, we use recursive binary splitting to grow a classification tree.

In the classification setting, RSS cannot be used as a criterion for making the binary splits.

A natural alternative to RSS is the *classification error rate*.

The classification error rate is the fraction of the training observations in that node but *not* in the most common class:

$$E = 1 - \max_k \hat{p}_{mk},$$

where $\hat{p}_{mk}$ is the proportion of training observations in the mth region that are from the kth class.

Other Measures for Classification

However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures of impurity are preferable.

- Gini Index (G): a measure of total variance across the K classes.
- Entropy (D): a measure of randomness (uncertainty) of a distribution.

These two approaches are more often used due to its sensitivity to the node purity.

Not just consider the proportion of the most common class, but *also the distribution of the other classes.*

Gini Index

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- G is small when all of $\hat{p}_{mk}$ are close to zero or one.
- A small value of G indicates that a node contains predominantly observations from a single class.

Entropy

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$

- Entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one.
- Like the Gini index, the entropy will take on a small value if the mth node is pure.

Main idea of Ensemble

An *ensemble method* is an approach that combines many simple building block models in order to obtain a single potentially more powerful model.

These simple building block models are sometimes known as *weak learners*, since they may lead to mediocre predictions on their own.

We will cover three types of ensemble method:
Bagging, *Random Forest* and *Boosting*.

Motivation of Bootstrapping

Assume we have B independent datasets from one distribution:

$$\{y_i^{(1)}, x_i^{(1)}\}_{i=1}^n, \cdots, \{y_i^{(B)}, x_i^{(B)}\}_{i=1}^n.$$

- From the each dataset $\{y_i^{(b)}, x_i^{(b)}\}_{i=1}^n$, construct the same type of model and the prediction $\hat{f}^{(b)}(x_0)$.
- Repeat for each dataset, we now have a sample of B $i.i.d$ predictions $\{\hat{f}^{(b)}(x_0)\}_{b=1}^B$.
- Using $\{\hat{f}^{(b)}(x_0)\}_{b=1}^B$, construct a final prediction as:

$$\hat{f}_{avg}(x_0) = \frac{\sum_{b=1}^B \hat{f}^{(b)}(x_0)}{B}.$$

- For classification, $\hat{f}_{avg}(x_0)$ can be the *majority vote*.
- The bias won't change, but the variance will shrink (*why?*)

The main idea of Bootstrapping

If we know the true data generating process, we can simply generate B datasets, and obtain B $i.i.d$ copies of $\hat{f}^{(b)}(x_0)$ to aggregate.

Unfortunately, in real problem, we cannot simulate data from the true generating process hence cannot access to B independent datasets. We only have a single dataset $\{y_i, x_i\}_{i=1}^n$.

Bootstrapping offers a way to obtain additional datasets from this single dataset, using the idea of *sampling with replacement*.

Bootstrapping Procedure

1. Given a training set of observations $\{(y_i, \mathbf{x}_i)\}_{i=1}^n$.
2. For $b = 1, 2, \ldots, B$, where B is the number of bootstrap samples we wish to create:
    - Draw n observations **with replacement** from the original data to form a bootstrap dataset $\{(y_i^{(b)}, \mathbf{x}_i^{(b)})\}_{i=1}^n$.
3. Now we have B bootstrap datasets

$$\{(y_i^{(b)}, \mathbf{x}_i^{(b)})\}_{i=1}^n, b = 1, 2, \ldots, B.$$

4. A separate model of the same type is fit to each of these B bootstrap samples, and obtain $\{\hat{f}^{(b)}(x)\}_{b=1}^B$.

Bagging

Bootstrap aggregation, or *bagging*, is a general-purpose procedure for reducing the variance.

Bagging Prediction

For a regression problem, the bagging prediction is given by:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{(b)}(x).$$

For classification, we classify based on the majority vote:

$$\hat{f}_{\text{bag}}(x) = \text{majority}\{\hat{f}^{(b)}(x)\}_{b=1}^{B}.$$

where $\hat{f}^{(b)}(x)$ is obtained from the bth bootstrap set.

Question: Do we need to worry if a large B will lead to overfit?

Bagging

- The key to bagging is that the bootstrap introduces variability in the constructed datasets.
- This leads to different models and hence more robust predictions.
- With tree method, you can let each weak learner grow deep without pruning.
- Each individual tree likely overfits, with high variance but low bias.
- Averaging these trees then reduces the variance of the final prediction by an amount.

Random Forest

- However, the individual weak learners constructed in the bagging procedure will still be correlated (*why?*)
- *Random Forests* further enhance the bagging concept by adding a layer of randomness in the feature selection.
- This helps to **decorrelate** between the trees (*why?*), leading to a further reduction in variance.

Random Forest Algorithm

Pseudo-code for Random Forest Algorithm

Given a training set $\{y_i, \mathbf{x}_i\}_{i=1}^n$:

```
1 For $b = 1$ to $B$:
    1 Draw a bootstrap sample of size $n$ from the training data.
    2 Grow a tree $T_b$ to the bootstrapped data, by repeating the following steps:
        1 Select $m$ variables at random from the $p$ variables.
        2 Pick the best variable/split-point among the $m$.
        3 Split the node into two children nodes.
2 Output the random forest: $\{T_b\}_{b=1}^B$.
```

The ensemble prediction with $\{T_b\}_{b=1}^B$ can be done in the same way before.

Random Forest

- Typically, we choose $m \approx \sqrt{p}$.
- Hence at each split, the algorithm *does not allow* most features to be considered.
- Both strong and weak features will get a chance in each tree, which makes the trees less similar.
- When $m = p$, the random forest algorithm reduces to the bagging.

Boosting

Similar to Bagging and Random Forest, Boosting is another type of ensemble method.

The construction of a weak learner in Bagging and Random forest does not depend on the construction of other learners.

Different from Bagging and Random Forest, the weak learners are constructed **sequentially** in Boosting.

Boosting for Regression Trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$:
   1. Fit a tree $\hat{f}^{(b)}$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   2. Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^{b}(x)$$
   3. Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^{(b)}(x_i)$$
3. Output the boosted model:
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^{(b)}(x)$$

Boosting for Regression Trees

There are three tuning parameter in Boosting:

- *The number of trees B.* Unlike bagging and random forests, **boosting can overfit if B is too large**.
- *The shrinkage parameter $\lambda > 0$*, controls the rate at which boosting learns (typically 0.01 or 0.001).
- *The number d of splits in each tree*, which controls the complexity of the boosted ensemble.
- Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split, involving a single variable.

Slow Learning

- The boosting approach aims to learn the training data slowly, before the model overfits.
- Each fitted tree is often very small.
- The shrinkage parameter λ further slows down the learning.
- The construction of the next tree aims to solve the problem remained from the previous tree.

For next week:

- Next week will be our last lecture.
- We will cover the topic of clustering, and some introduction of Bayesian inference.
- Some final exam details will also be announced.
- Again, don't forget to bring the printout for the next quiz.