دانشگاه صنعتی اصفهان

دانشکده علوم ریاضی

عنوان:

روش های درخت محور

پروژه کارشناسی آمار

به کوشش:

عارفه نایبی

استاد راهنما:

دکتر مریم کلکین نما

۱۴۰۳

فهرست مطالب

مقدمه

این پروژه به بررسی روش‌های مبتنی بر درخت برای رگرسیون و طبقه‌بندی می‌پردازد. این روش‌ها شامل تقسیم فضای پیش‌بینی‌کننده به نواحی ساده هستند که به ما این امکان را می‌دهند تا پیش‌بینی‌هایی بر اساس میانگین یا مد مشاهدات آموزشی در آن نواحی انجام دهیم. چون مجموعه قوانین تقسیم‌بندی که برای تقسیم فضای پیش‌بینی‌کننده استفاده می‌شود، می‌تواند در یک درخت خلاصه شود، این رویکردها به عنوان ***روش‌های درخت تصمیم*** شناخته می‌شوند. اگرچه روش‌های مبتنی بر درخت کاربردی و تفسیرپذیر هستند، اما معمولاً دقت پیش‌بینی آن‌ها به اندازه‌ای که تکنیک‌های یادگیری نظارت شده پیشرفته‌تر دارند، نیست. مطالب این بخش از فصل هشتم کتاب An Introduction to Statistical Learning گرداوری شده است.

۱. اصول درخت‌های تصمیم

درخت‌های تصمیم می‌توانند برای هر دو نوع مسأله رگرسیون و طبقه‌بندی مورد استفاده قرار گیرند. در ابتدا به مسایل رگرسیون می‌پردازیم و از داده‌های Hitters برای پیش‌بینی حقوق بازیکنان بیسبال بر اساس سال‌های تجربه و ضربات آن‌ها استفاده می‌کنیم. درخت رگرسیون ساخته‌شده از این داده‌ها نشان می‌دهد که چگونه فضای پیش‌بینی‌کننده به نواحی مختلف تقسیم می‌شود، به طوری که هر ناحیه به یک پیش‌بینی مشخص برای حقوق مربوط می‌شود.

- از آنجا که دستوراتی که این نواحی را ایجاد می کنند را میتوان در قالب یک نمودار مشابه درخت نمایش داد، به این مدل روش ها، روش های درخت تصمیم (Decision Tree) گویند.
- مزیت این روش ها، ساده بودن و تغییر پذیری بالای آنهاست.
- ایراد این روش ها از لحاظ دقت در پیش بینی معمولا قابل مقایسه با روش های دیگر مانند رگرسیون خطی ، شیکه عصبی و.. نیست.
- با این وجود روش هایی مانند Bagging , Random Forest , Boosting توانایی درخت های تصمیم در پیش بینی را افزایش می دهند. در واقع این روش ها، چندین درخت تصمیم ایجاد می کنند و

سپس نتایج آنها را برای رسیدن به یک نتیجه بهتر، ترکیب می کنند. با این حال این امر خود از توانایی تغییر پذیری درخت های تصمیم کم می کند.

۱.۱ درخت های رگرسیونی

در ابتدا برای روشن شدن موضوع به مثال زیر توجه کنید.

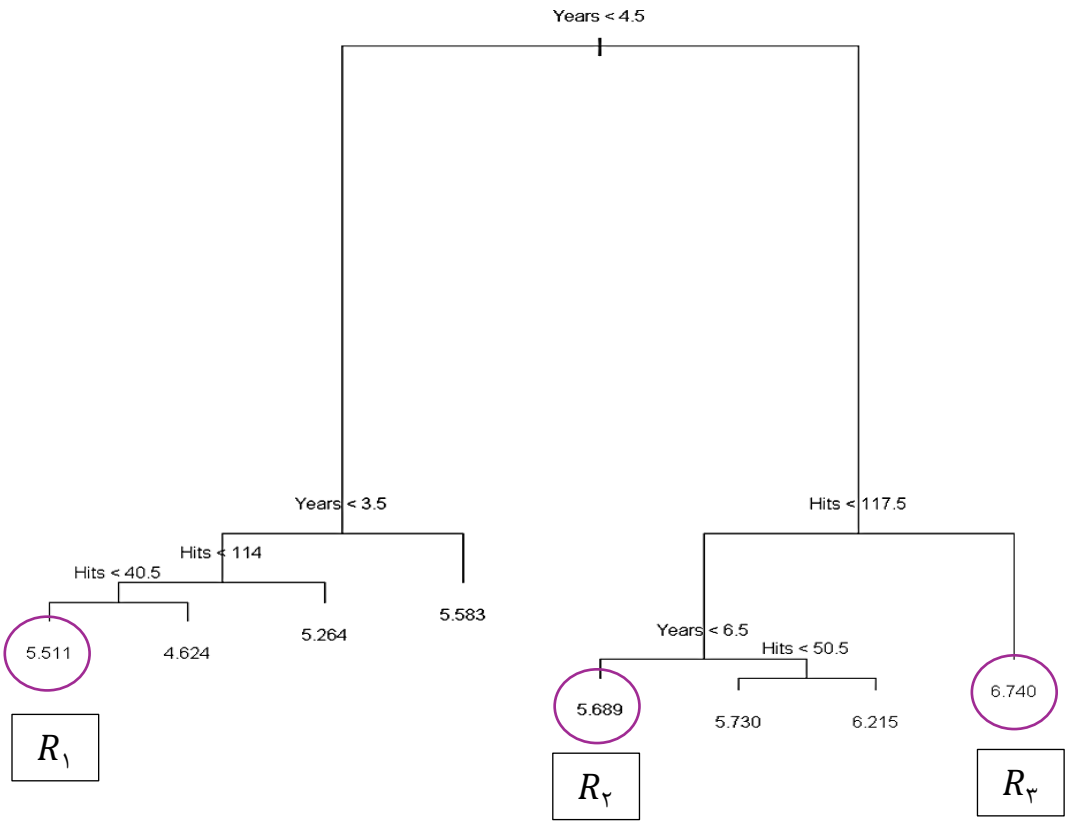
مثال) داده های Hitters data در پکیج ISLR در نرم فزار R

Salary : درآمد سالانه بازیکنان بیس بال

Years : تعداد سال های بازی در لیگ اصلی

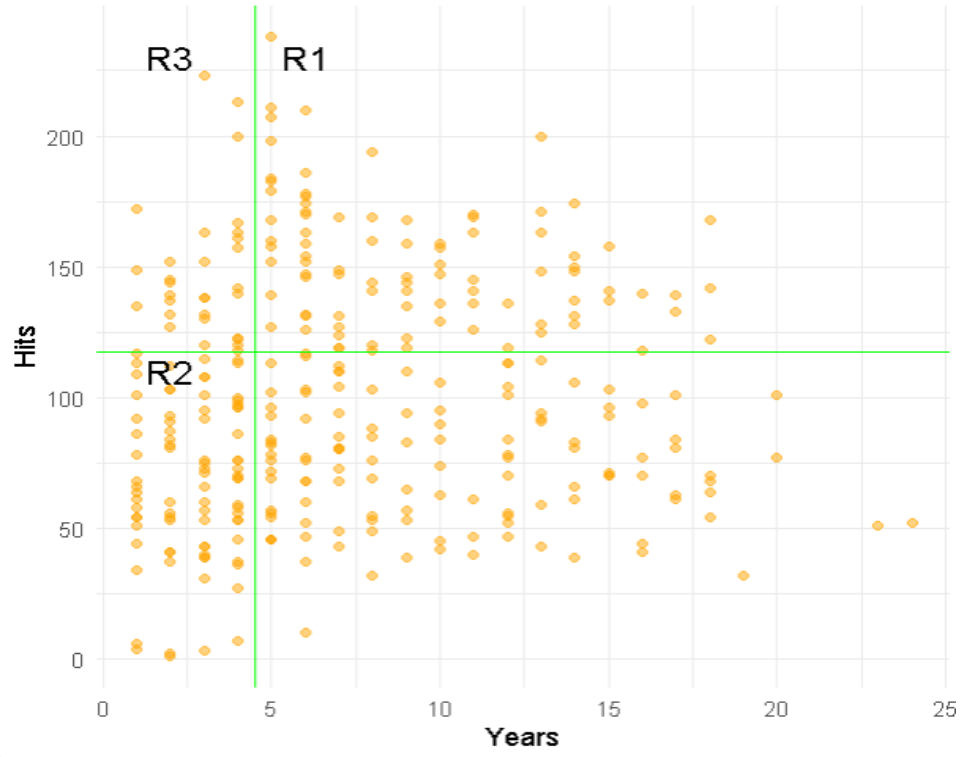
Hits : تعداد گل ها در سال گذشته

```
library(tree)
library(ISLR)
attach(Hitters)
tree_model <- tree(log(Salary) ~ Years + Hits, data = Hitters)
plot(tree_model)
text(tree_model, pretty = 0, cex = 0.6)
```



- R_1 و R_2 و R_3 : Terminal nodes یا leaves یا برگ ها
- Years < ۵,٤ و Hits > ۱۱۷,۵ : interval nodes
- بخش هایی از درخت که نود ها را به هم ارتباط می دهند: Branches

```
library(ggplot2)
ggplot(Hitters, aes(x = Years, y = Hits)) +
  geom_point(color = "orange", alpha = 0.5) +
  geom_hline(yintercept = 117.5, color = "green") +
  geom_vline(xintercept = 4.5, color = "green") +
  annotate("text", x = 2, y = 230, label = "R3", size = 5) +
  annotate("text", x = 2, y = 110, label = "R2", size = 5) +
  annotate("text", x = 6, y = 230, label = "R1", size = 5) +
  annotate("text", x = 2, y = 238, label = "R1", size = 5) +
  labs(x = "Years", y = "Hits") +
  theme_minimal()
```



پیش بینی:

- میانگین پاسخ در هر ناحیه R_1 تا R_3 را برای پیش بینی به کار میبریم.

تفسیر:

- متغییر Years مهمترین متغییر در تعیین درامد بازیکنان است و یازیکنان با سابقه کمتر درامد کمتری دارند.
- اگر بازیکن تجربه کمی داشته باشد، تعداد گل های زده توسط وی تاثیر چندانی در درآمدش ندارد.
- برای بازیکنی که پنج سال یا بیشتر در لیگ های اصلی بوده اند، تعداد گل های زده در سال گذشته، تاثیر زیادی بر درآمد آنها دارد و بازیکنانی که گل بیشتر زده اند، درآمد بیشتری داشته اند. (اثر متقابل)

برخلاف مدل های رگرسیون خطی، نمیتوان در تفسیر ضرایب تاثیر هر یک گل اضافه یا هر یک سال بازی بیشتر را نشان داد.

سوال اصلی: نواحی R_1 تا R_J برای n مشاهده از p متغیر، چگونه تشکیل می شوند؟

این نواحی را به گونه ای تعیین می کنیم که RSS زیر می نیمم شود:

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

$\hat{y}_{R_j}$: میانگین پاسخ در ناحیه R_j

در عمل (از نظر محاسباتی) بررسی تمام نرم افزار های ممکن متغییر های توضیحی به J ناحیه، امکان پذیر نیست.
برای رفع این شکل از رویکردی به نام جداسازی دودویی بازگشتی به کار برده میشوند.

recursive binary splitting

در این رویکرد، در هر مرحله جداسازی، بهترین جداسازی انجام میشود بدون توجه به اینکه چه چیزی میتواند در قدم های بعدی به جدا سازی های بهتری منجر شود.

برای اجرای این الگوریتم، ابتدا پیش گو کننده X_j و آستانه s را طوری انتخاب میکنیم که نواحی

$$R_1(j,s) = \{X | X_j < s\}$$
$$R_2(j,s) = \{X | X_j \geq s\}$$

بیشترین کاهش را در RSS ایجاد کنند.

بنابراین j و s ای را انتخاب میکنیم که عبارت زیر را می نیمم کند.

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

پیدا کردن j و s ای که عبارت بالا را مینیمم کند از لحاظ محاسباتی سریع تر است به خصوص وقتی p زیاد باشد.

سپس فرآیند را تکرار می‌کنیم تا بهترین پیش‌بینی‌کننده و بهترین نقطه تقسیم را پیدا کنیم و داده‌ها را برای کمینه‌سازی RSS در هر یک از نواحی حاصل تقسیم کنیم. به این ترتیب، تعداد نواحی افزایش می‌یابد تا زمانی که به یک معیار توقف برسیم؛ به عنوان مثال، ممکن است ادامه دهیم تا هیچ ناحیه‌ای بیش از پنج مشاهده نداشته باشد.

بعد از ایجاد نواحی، برای یک مشاهده آزمایشی خاص، پاسخ را با استفاده از میانگین مشاهدات آموزشی در ناحیه‌ای که آن مشاهده به آن تعلق دارد، پیش‌بینی می کنیم.

۱.۱.۱ هرس درخت

معمولا روشی که در بالا ارائه شد منجر به درخت بزرگی میشود که بسیار محتمل است که به داده ها بیش برازش (overfit) شود و درخت بسیار پیچیده شود.

معمولا درخت کوچکتر و ساده تر با نواحی R_1 تا R_J که J بزرگ نباشد، میتواند منجر به واریانس کمتر و تفسیر پذیری بیشتر شود، هرچند، مقدار اریبی ایجاد میکند و پیش بینی را برای مشاهدات موجود در مجموعه داده آموزشی، کمی ضعیف تر میکند.

یک راه برای جلوگیری از بیش برازش شدن، این است که در هر مرحله، تشکیل ناحیه جدید (ایجاد شاخه جدید) زمانی صورت گیرد که کاهش در RSS از یک حدی بیشتر باشد و در واقع نواحی(شاخه های) ارزشمند، اجازه تشکیل شدن داشته باشند.

ولی این روش، چندان مناسب نیست، چون ناحیه ای که ارزش چندانی برای تشکیل شدن در یک مرحله ندارد، میتواند منجر به جداسازی بسیار خوبی در مراحل بعدی شود ولی این روش آن را در نظر نمیگیرد.

یک راه بهتر این است که درخت بزرگی تشکیل دهیم به اسم درخت T_0 تا تمامی نواحی مناسب ایجاد شود، سپس شروع به هرس آن نماییم و به یک زیر درخت برسیم.

هدف رسیدن به زیر درختی است که منجر به کمترین مقدار MSE برای داده های آزمایشی شود. درواقع باید هربار یک زیر درختی تشکیل داده و سپس مقدار MSE آزمایشی آن را محاسبه کرد ولی این کار یعنی هرس کردن T_0 به تمام زیر درخت های ممکن، کار بسیار زمان بری است.

برای رفع این مشکل، روشی را در نظر می گیریم که تنها مجموعه کوچکی از زیر درخت ها را برای بررسی در نظر می گیرند.

۲.۱.۱ هرس خفیف ترین ساقه

برای هر تعداد $\alpha \geq 0$ که پارامتر تنظیم (tuning parameter) نام دارد، زیر درخت $T_0 \subset T$ ای وجود دارد که :

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

تا جایی که ممکن است کوچک باشد.

نکته: $\hat{y}_{R_m}$ هارا از قبل، از درخت T_0 در اختیار هستند.

$|T|$: تعداد نواحی ایجاد شده توسط درخت T .

α : تنظیم کننده سطح پیچیدگی درخت و میزان برازش آن به داده های نمونه (آزمایشی) است

هر چه α بیشتر شود، هزینه پیچیده شدن درخت بالا می رود و معمولا درخت به سمت ساده تر شدن می رود.

α	MSE
α_1	MSE_{α_1}
α_2	MSE_{α_2}
⋮	⋮
α_{min}	$MSE_{\alpha_{min}}$
⋮	⋮
α_m	MSE_{α_m}

۲.۱ درخت های طبقه بندی

مشابه درخت های رگرسیونی هستند فقط به جای میانگین پاسخ در هر ناحیه R_J ، مقداری از متغیر پاسخ که در آن ناحیه، فراوانی بیشتری دارد را در نظر میگیریم.

با این حال در تشکیل نواحی R_1 تا R_J ، به جای RSS از نرخ خطا استفاده می کنیم. در واقع مبتنی از داده های نمونه(آموزشی) در آن ناحیه که به کلاس با فراوانی بیشتر تعلق ندارند:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

به این رابطه، classification error rate می گویند.

$\hat{P}_{m_k}$ نسبتی از داده های نمونه (آموزشی) در این m ناحیه است که به کلاس k ام تعلق دارد.

به جای نرخ خطا از دو معیار زیر هم استفاده می شود:

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

به این رابطه، Gini index می گویند.

اگر تمام $\hat{P}_{m_k}$ ها نزدیک ۰ یا ۱ باشند، مقدار این شاخص بسیار کوچک می شود و از آن به عنوان درجه خلوص ناحیه m ام (node purity) نام می برند که اگر کوچک باشد یعنی اکثر مشاهدات آن ناحیه از یک طبقه آمده اند.

همچنین معیار آنتروپی تقاطع cross-entropy :

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

استنباطی مشابه شاخص جینی دارد و این دو بسیار شبیه به هم عمل می کنند.

برای درخت های طبقه بندی، هرسه معیار را میتوان به کار برد اما اگر صحت پیش بینی درخت هرس شده نهایی، هدف اصلی باشد، معیار نرخ خطای طبقه بندی مناسب تر است.

۳.۱ مقایسه درخت تصمیم با مدل خطی:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j, \qquad f(X) = \sum_{m=1}^{M} c_m \cdot 1_{(X \in R_m)}$$

که c_m میانگین پاسخ در ناحیه R_m است.

- اگر ارتباط میان پاسخ و متغیر های توضیحی، به خوبی توسط یک مدل خطی تقریب زده شود، مدل خطی پیش بینی خوبی ارائه می دهد.
- اگر این ارتباط به شدت غیرخطی و پیچیده باشد، در این صورت درخت تصمیم ارجحیت دارد.
- با معیار هایی نظیر MSE آزمایشی می توان دو مدل را مقایسه کرد.
- از مزایای خوب درخت های تصمیم این است که برای متغیر های توضیحی کیفی (معمولا با سطوح بالا)، نیازی به معرفی متغیر نشانگر ندارد.
- با این حال درخت های تصمیم عمدتا برای پیش بینی از صحت لازم نسبت به سایر رویکرد های رگرسیونی، برخوردار نیستند.

۲. Bagging ، جنگل های تصادفی، Boosting

۱.۲ Bagging :

معمولا درخت های تصمیمی که هرس نشده اند، واریانس زیادی دارند. روش بگینگ یک رویکرد کلی در روش های یادگیری است که میتواند منجر به کاهش واریانس $\hat{f}$ شود. در اینجا این رویکرد را در مسئله درخت های تصمیم به کار می بریم.

مثال : اگر

$$z_1, \dots, z_n \overset{ind}{\sim} var(z_i) = \sigma^2 \rightarrow var(\bar{z}) = \frac{\sigma^2}{n}$$

استفاده از $\bar{z}$ واریانس کمتری را نسبت به هر یک از Z_i ها به شرط استقلال Z_i ها، ایجاد می کند.

پس یک راه برای کاهش واریانش درخت های تصمیم، در نظر گرفتن مجموعه داده های آموزشی مختلف، برازش درخت تصمبم و به به دست اوردن $\hat{f}^b$ روی هر یک ا ز i , (b = ۱,..,B) مجموعه آموزشی موجود:

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

با این حال، این کار عمدتا امکان پذیر نیست و دسترسی به مجموعه داده های آموزشی مختلف نداریم. پس با استفاده از بوت استرپ، B نمونه $z_1, \dots, z_B$ را از مجموعه آموزشی در دست تولید می کنیم. روی هر یک درختی بدون هرس کردن برازش می دهیم. این درخت ها معمولا اریبی کم ولی واریانس زیادی دارند که با متوسط گیری واریانس آنها کاهش می یابد.

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

سوال: اگر Y یک متغیر رسته ای باشد، Bagging به چه صورت عمل می کند؟

معمولا به این مسئله کمتر پرداخته شده ولی یک رویکرد این است که برای مشاهده جدید در مجموعه داده آزمایشی، می توان تعیین کرد که هریک از B درختی که با Bagging تولید می شود، چه رسته ای را برای پاسخ آن مشاهده، تعیین می کند. سپس در نهایت آن رسته ای که در بیشتر درخت ها انتخاب شده، برای آن در نظر گرفته می شود.

تعیین اهمیت متغیر های توضیحی:

با Bagging به دلیل اینکه دیگر نمی توان نتایج را در قالب یک درخت نشان داد، تفسیر پذیری مدل کاهش می یابد، هرچند توان پیش بینی آن افزایش داشته است.

با این حال برای تعیین اهمیت متغییر ها، میتوان مقدار کلی کاهش در RSS (رگرسیون) یا شاخص جینی (طبقه بندی) در جداسازی های مشخصه های مختلف درخت ها، با متوسط گیری روی نتایج B درخت ایجاد می کند را در نظر گرفت.

۲.۲ جنگل های تصادفی

مشابه با Bagging ، درخت های تصمیم مختلفی روی نمونه های آموزشی بوت استرپ شده تشکیل می دهد ولی هربار درخت مورد نظر بر روی نمونه تصادفی m تایی از p متغیر توضیحی انجام می شود. معمولا $m \approx \sqrt{P}$.

این مسئله باعث می شود که درخت های نا همبسته ایجاد شوند:

در نظر بگیرید یک متغیر توضیحی خیلی قوی وجود داشته باشد و سایر متغیر های توضیحی، تقریبا اهمیت متوسطی داشته باشند. بنابراین اکثر درخت ها، تقریبا این متغیر قوی را در اولین لایه درخت خواهند داست. به همین ترتیب اکثر درخت ها، تقریبا یک شکل به نظر میرسند.

بنابراین پیش بینی بر اساس نتایج این درخت ها وابستگی زیادی به یکدیگر پیدا می کنند. متاسفانه متوسط گیری روی تعداد زیادی کمیت با وابستگی بالا، باعث کاهش شدید در واریانس نمی شود.

در جنگل های تصادفی، سایر متغیر های توضیحی، شانس حضور در لایه اول را پیدا می کنند. این امر باعث ناهمبسته شدن درخت ها می شوند.

اگر m = p ، جنگل تصادفی همان Bagging می شود.

۳.۲ Boosting :

مشابه با Bagging، Boosting هم یک رویکرد کلی در روش های یادگیری آماری است.

در Boosting، درخت ها یه صورت دنباله ای رشد می کنند. هر درخت بر اساس اطلاعاتی که از درخت قبلی به دست آمده است، رشد می کند.

Boosting، از نمونه گیری بوت استرپ استفاده نمی کند و به جای آن هر درخت بر روی نسخه تغیر یافته ای از داده های آموزشی برازش می شود.

الگوریتم:

۱) $\forall_i \ r_i = y_i \ \ , \hat{f}(x) = ۰$

۲) برای $b = ۱,\ldots,B$ این مراحل را تکرار میکنیم:

a. درخت $\hat{f}^b$ را با $\alpha + ۱$ ناحیه تشکیل می دهیم بر مجموعه داده آموزشی (X, r).

b. $\tilde{f}_{new}(x) = \tilde{f}_{old}(x) + \lambda \tilde{f}^b(x)$

c. $\forall i \ \ r_{i_{new}} = r_{i_{old}} - \lambda \tilde{f}^b(x_i)$

۳) $\tilde{f}(x) = \sum_{b=۱}^{B} \lambda \, \tilde{f}^b(x)$

در واقع رویکرد Boosting، به آهستگی آموزش می بیند به طوری که با داشتن مدل در دست، درخت تصمیم را بر روی مانده های مدل(یعنی بخشی از پاسخ که توسط متغیر قبلی توضیح داده نشده است) برازش می دهد. (استفاده از مانده ها به جای پاسخ)

با در نظر گرفتن درخت های کوچک(d کوچک)، بوستینگ به آهستگی آموزش می بیند و یاد می گیرد در ناحیه ای که خوب عمل نکرده (مانده ها)، بهتر شود.

روش Boosting، برای طبقه بندی به نحو مشابه عمل می کند ولی کمی پیچیده تر است و در اینجا مورد بحث قرار نمی گیرد.

سه پارامتر در Boosting اهمیت دارند:

۱. تعداد درخت ها (B) : بر خلاف Bagging و جنگل های تصادفی، با افزایش B، بوستینگ می تواند بیش برازش شود.(هرچند این مسئله به آهستگی صورت می گیرد.). برای انتخاب B از MSE آزمایشی به ازای مقادیر مختلف B می توان استفاده کرد.
۲. پارامتر کاهش (λ) :کنترل کننده نرخ یادگیری Boosting است. معمولا مقدار ۰.۰۰۱ یا ۰.۰۱ را می گیرد.
۳. تعداد نواحی در هر درخت(d + ۱) : کنترل کننده پیچیدگی درخت های به کار رفته است. اغلب d = ۱ به خوبی کار می کند. یعنی هر درخت یک کنده است که از یک شکاف تشکیل شده است و فقط ۲ ناحیه ایجاد می کند.

٤.۲ خلاصه‌ای از روش‌های مجموعه درختان

ما اکنون چهار روش برای تنظیم یک مجموعه از درختان را مشاهده کرده‌ایم: Bagging ، جنگل‌های تصادفی، و Boosting.

- Bagging یک تکنیک برای ترکیب چند درخت تصمیم‌گیری به منظور بهبود پیش‌بینی‌ها است که درختان به‌طور جداگانه بر روی داده‌های مختلف (نمونه‌های تصادفی) آموزش می‌بینند. چون درختان به هم شبیه هستند، ممکن است نتوانند به خوبی به جستجوی فضای مدل بپردازند و در نتیجه ممکن است به

نتایج بهینه محلی برسند که بهترین نتیجه ممکن نیست. به طور کلی، Bagging می‌تواند به دلیل شباهت زیاد درختان، در پیدا کردن بهترین مدل مشکل داشته باشد.

- جنگل های تصادفی، یک روش پیشرفته‌تر برای ترکیب چند درخت تصمیم‌گیری است که درختان به‌طور جداگانه و بر روی داده‌های مختلف آموزش می‌بینند. هنگام تقسیم‌کردن داده‌ها، از ویژگی‌های تصادفی استفاده می‌شود و این کار باعث می‌شود که درختان کمتر به هم وابسته باشند و به همین دلیل، مدل بهتری ایجاد می‌شود و فضای مدل به‌طور کامل‌تری کاوش می‌شود. در نتیجه، جنگل‌های تصادفی معمولاً عملکرد بهتری نسبت به Bagging دارند.
- Boosting یک روش برای بهبود پیش‌بینی‌ها با استفاده از چندین درخت تصمیم‌گیری است که فقط از داده‌های اصلی استفاده می‌شود، نه نمونه‌های تصادفی. درختان یکی پس از دیگری ساخته می‌شوند، هر درخت جدید به خطاها یا سیگنال‌های باقی‌مانده از درختان قبلی توجه می‌کند، درخت جدید قبل از استفاده، کمی کوچک‌تر می‌شود تا از بیش‌برازش جلوگیری کند. در نهایت، Boosting به ما کمک می‌کند تا مدل‌های قوی‌تری بسازیم که خطاهای قبلی را جبران کنند.

Lab

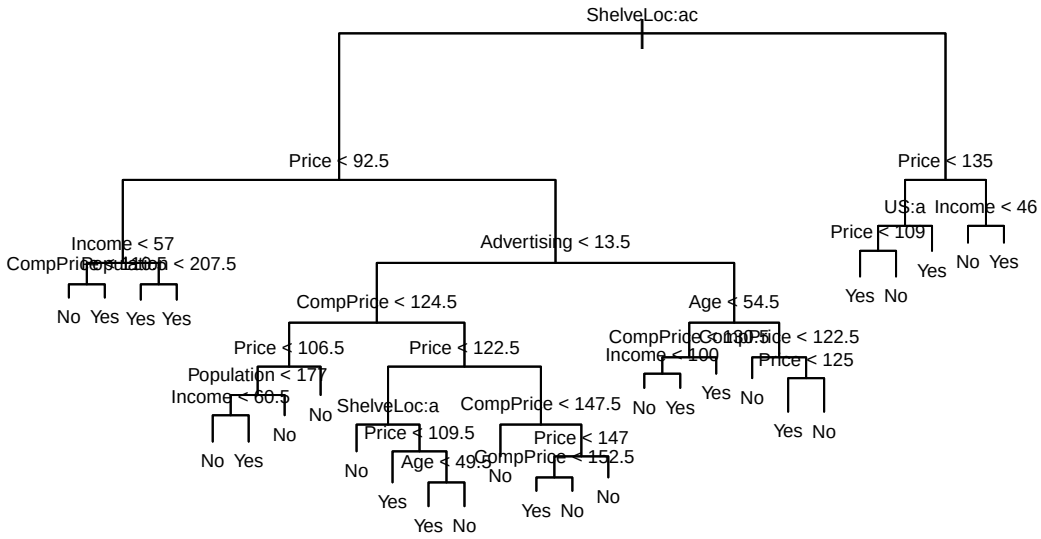
Arefeh Nayebi

2025-02-24

```
library(tree)
library(ISLR)
attach(Carseats)
High = ifelse ( Sales <=8 ," No " ," Yes ")
High <- as.factor(High)
Carseats = data.frame ( Carseats , High )
tree.carseats <- tree(High ~ . - Sales, Carseats)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

```
plot(tree.carseats)
text(tree.carseats, cex = 0.6)
```




```
tree.carseats
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500  No  ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600  No  ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530  Yes  ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220  No  ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000  No  ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730  Yes  ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470  Yes  ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170  Yes  ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941  Yes  ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800  No  ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200  No  ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890  No  ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150  No  ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300  No  ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000  No  ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407  Yes  ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477  No  ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000  No  ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200  No  ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680  Yes  ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702  No  ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930  Yes  ( 0.37500 0.62500 )
```

```
##               170) Price < 109.5 16   7.481  Yes  ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600  No  ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050  Yes  ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702  No  ( 0.90909 0.09091 ) *
##             43) Price > 122.5 77  55.540  No  ( 0.88312 0.11688 )
##               86) CompPrice < 147.5 58  17.400  No  ( 0.96552 0.03448 ) *
##               87) CompPrice > 147.5 19  25.010  No  ( 0.63158 0.36842 )
##                 174) Price < 147 12  16.300  Yes  ( 0.41667 0.58333 )
##                   348) CompPrice < 152.5 7   5.742  Yes  ( 0.14286 0.85714 ) *
##                   349) CompPrice > 152.5 5   5.004  No  ( 0.80000 0.20000 ) *
##                 175) Price > 147 7   0.000  No  ( 1.00000 0.00000 ) *
##         11) Advertising > 13.5 45  61.830  Yes  ( 0.44444 0.55556 )
##           22) Age < 54.5 25  25.020  Yes  ( 0.20000 0.80000 )
##             44) CompPrice < 130.5 14  18.250  Yes  ( 0.35714 0.64286 )
##               88) Income < 100 9  12.370  No  ( 0.55556 0.44444 ) *
##               89) Income > 100 5   0.000  Yes  ( 0.00000 1.00000 ) *
##             45) CompPrice > 130.5 11   0.000  Yes  ( 0.00000 1.00000 ) *
##           23) Age > 54.5 20  22.490  No  ( 0.75000 0.25000 )
##             46) CompPrice < 122.5 10   0.000  No  ( 1.00000 0.00000 ) *
##             47) CompPrice > 122.5 10  13.860  No  ( 0.50000 0.50000 )
##               94) Price < 125 5   0.000  Yes  ( 0.00000 1.00000 ) *
##               95) Price > 125 5   0.000  No  ( 1.00000 0.00000 ) *
##      3) ShelveLoc: Good 85  90.330  Yes  ( 0.22353 0.77647 )
##        6) Price < 135 68  49.260  Yes  ( 0.11765 0.88235 )
##         12) US: No 17  22.070  Yes  ( 0.35294 0.64706 )
##           24) Price < 109 8   0.000  Yes  ( 0.00000 1.00000 ) *
##           25) Price > 109 9  11.460  No  ( 0.66667 0.33333 ) *
##         13) US: Yes 51  16.880  Yes  ( 0.03922 0.96078 ) *
##        7) Price > 135 17  22.070  No  ( 0.64706 0.35294 )
##         14) Income < 46 6   0.000  No  ( 1.00000 0.00000 ) *
##         15) Income > 46 11  15.160  Yes  ( 0.45455 0.54545 ) *
```

```
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train, ]
High.test <- High[-train]
tree.carseats <- tree(High ~ . - Sales, Carseats, subset = train)
tree.pred <- predict(tree.carseats, Carseats.test, type = "class")
table(tree.pred, High.test)
```

```
##          High.test
## tree.pred  No  Yes
##      No   104   33
##      Yes   13   50
```

```
(104 + 50)/200
```

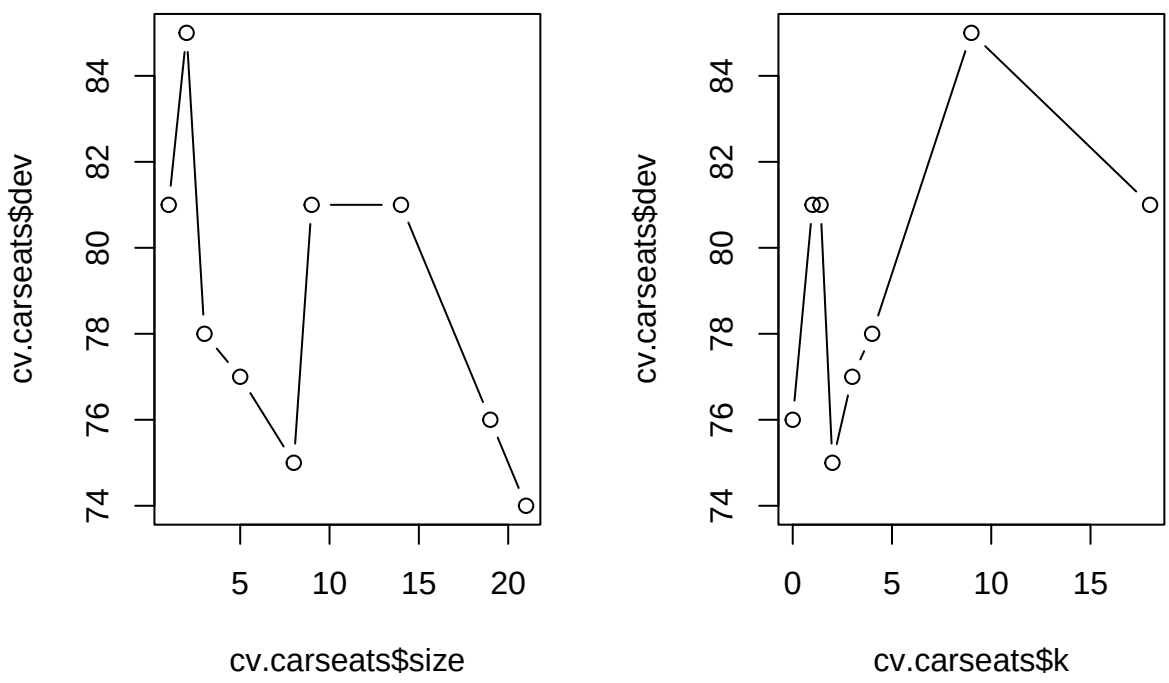
```
## [1] 0.77
```

```
set.seed(3)
cv.carseats <- cv.tree(tree.carseats, FUN = prune.misclass)
names(cv.carseats)
```

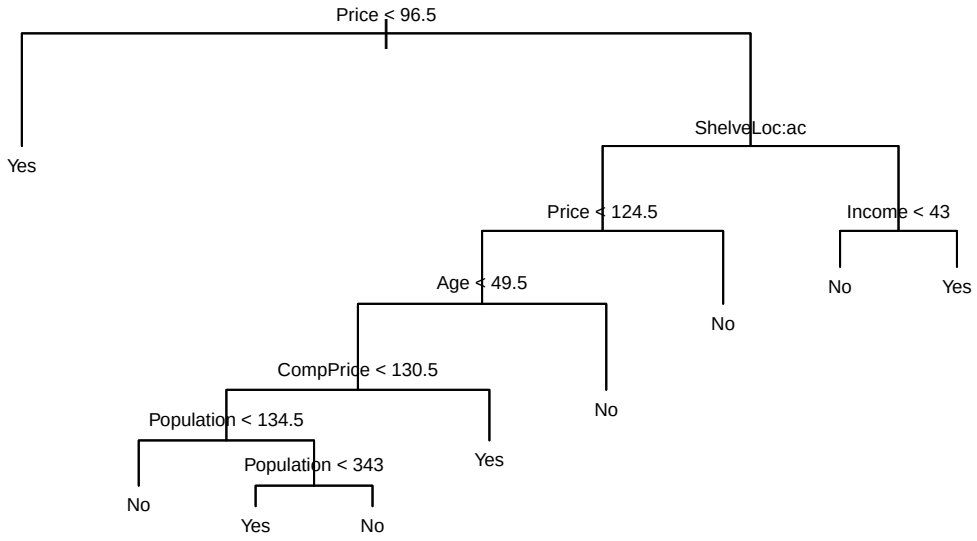
```
## [1] "size"   "dev"    "k"      "method"
```

```
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
```

```
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```




```
library(tree)
prune.carseats <- prune.misclass(tree.carseats, best = 9)
plot(prune.carseats)
text(prune.carseats, cex = 0.6)
```



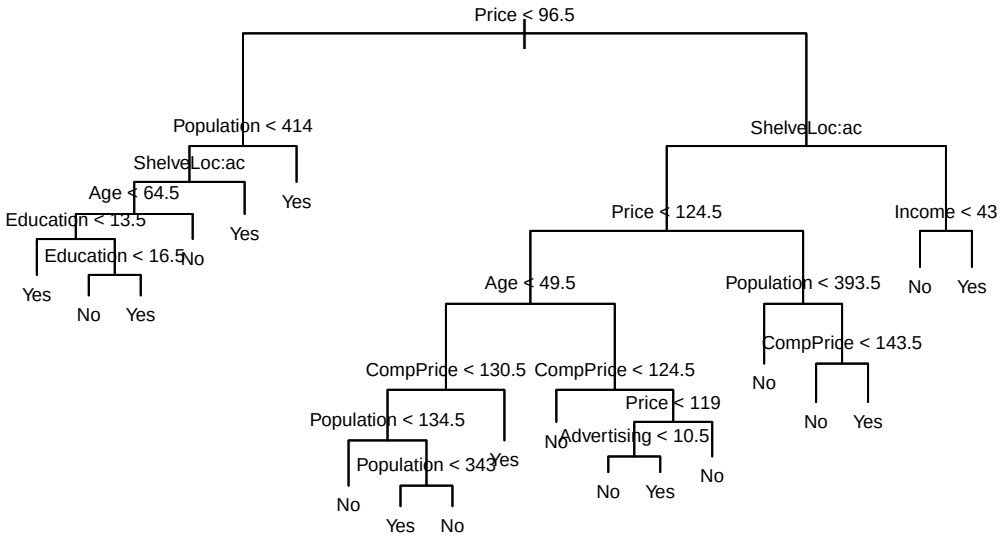

```
tree.pred <- predict(prune.carseats, Carseats.test, type = "class")
table(tree.pred, High.test)
```

```
##          High.test
## tree.pred  No   Yes
##      No     97    25
##      Yes    20    58
```

```
(97 + 58)/200
```

```
## [1] 0.775
```

```
prune.carseats <- prune.misclass(tree.carseats, best = 15)
plot(prune.carseats)
text(prune.carseats, cex = 0.6)
```




```
tree.pred <- predict(prune.carseats, Carseats.test, type = "class")
table(tree.pred, High.test)
```

```
##          High.test
## tree.pred  No   Yes
##      No    102    30
##      Yes    15    53
```

```
(102 + 53)/200
```

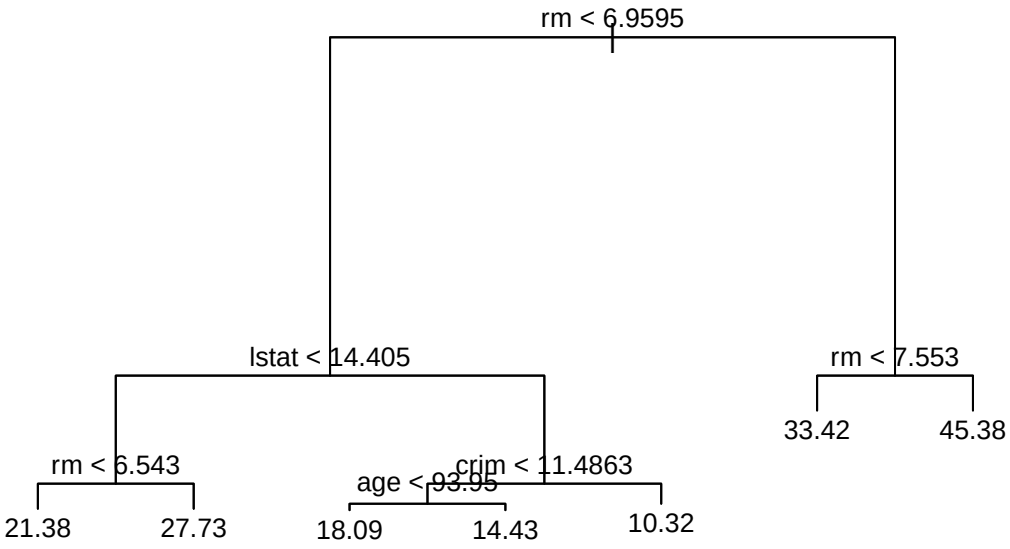
```
## [1] 0.775
```

```
library(MASS)
library(tree)
library(ISLR)
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston <- tree(medv ~ ., Boston, subset = train)
summary(tree.boston)
```

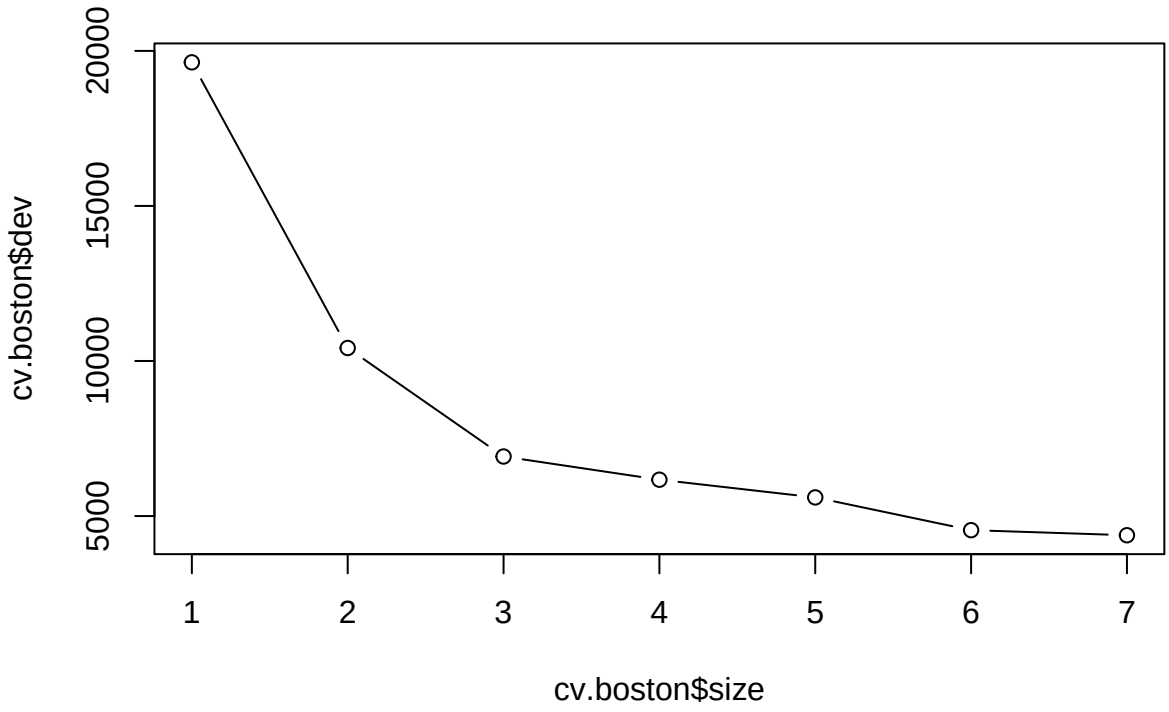
```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

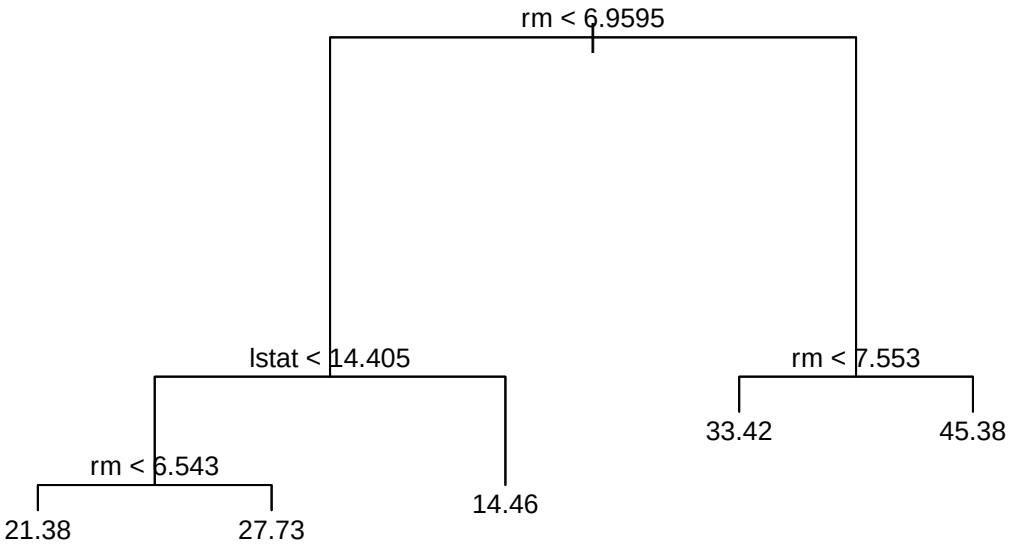
```
plot(tree.boston)
text(tree.boston, cex = 0.8)
```



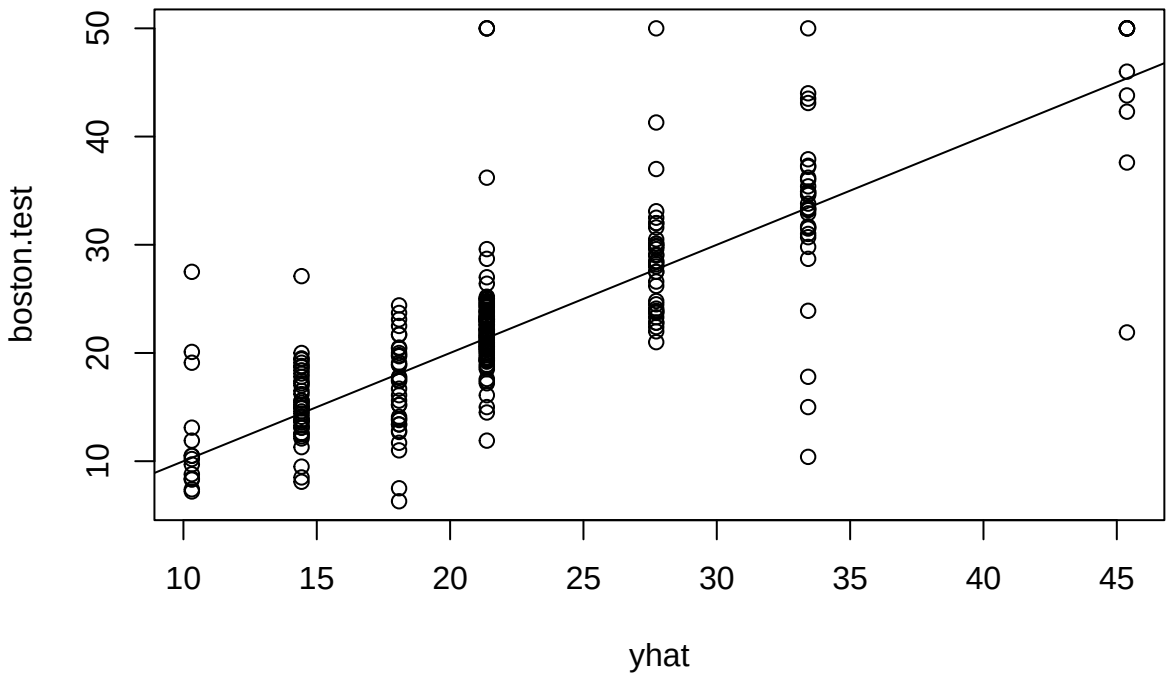

```
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type = "b")
```




```
prune.boston <- prune.tree(tree.boston, best = 5)
plot(prune.boston)
text(prune.boston, cex = 0.8)
```




```
yhat <- predict(tree.boston, newdata = Boston[-train, ])
boston.test <- Boston[-train, "medv"]
plot(yhat, boston.test)
abline(0, 1)
```




```
mean((yhat - boston.test)^2)
```

```
## [1] 35.28688
```

```
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

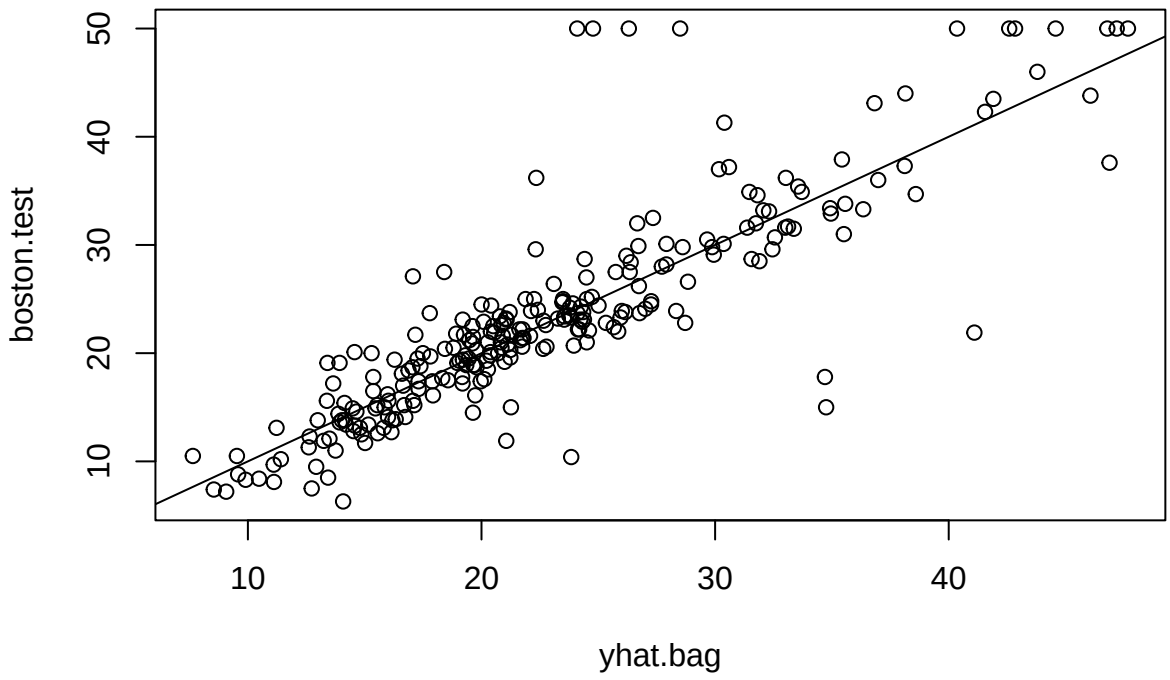
```
library(MASS)
```

```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
bag.boston <- randomForest(medv ~ ., data = Boston, subset = train, mtry = 13, importance = TRUE)
bag.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 11.33119
##                     % Var explained: 85.26
```

```
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
boston.test <- Boston[-train, "medv"]
```

```
plot(yhat.bag, boston.test)
abline(0, 1)
```




```
mean((yhat.bag - boston.test)^2)
```

```
## [1] 23.4579
```

```
bag.boston <- randomForest(medv ~ ., data = Boston, subset = train, mtry = 13, ntree = 25)
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
mean((yhat.bag - boston.test)^2)
```

```
## [1] 22.99145
```

```
set.seed(1)
rf.boston <- randomForest(medv ~ ., data = Boston, subset = train, mtry = 6, importance = TRUE)
yhat.rf <- predict(rf.boston, newdata = Boston[-train, ])
mean((yhat.rf - boston.test)^2)
```

```
## [1] 19.62021
```

```
importance(rf.boston)
```

```
##           %IncMSE IncNodePurity
## crim    16.697017    1076.08786
## zn       3.625784      88.35342
## indus    4.968621     609.53356
## chas     1.061432      52.21793
## nox     13.518179     709.87339
```

```
## rm      32.343305    7857.65451
## age     13.272498     612.21424
## dis      9.032477     714.94674
## rad      2.878434      95.80598
## tax      9.118801     364.92479
## ptratio  8.467062     823.93341
## black    7.579482     275.62272
## lstat   27.129817    6027.63740
```

```
importance(rf.boston)
```

```
##           %IncMSE IncNodePurity
## crim    16.697017    1076.08786
## zn       3.625784      88.35342
## indus    4.968621     609.53356
## chas     1.061432      52.21793
## nox     13.518179     709.87339
## rm      32.343305    7857.65451
## age     13.272498     612.21424
## dis      9.032477     714.94674
## rad      2.878434      95.80598
## tax      9.118801     364.92479
## ptratio  8.467062     823.93341
## black    7.579482     275.62272
## lstat   27.129817    6027.63740
```

```
library(gbm)
```

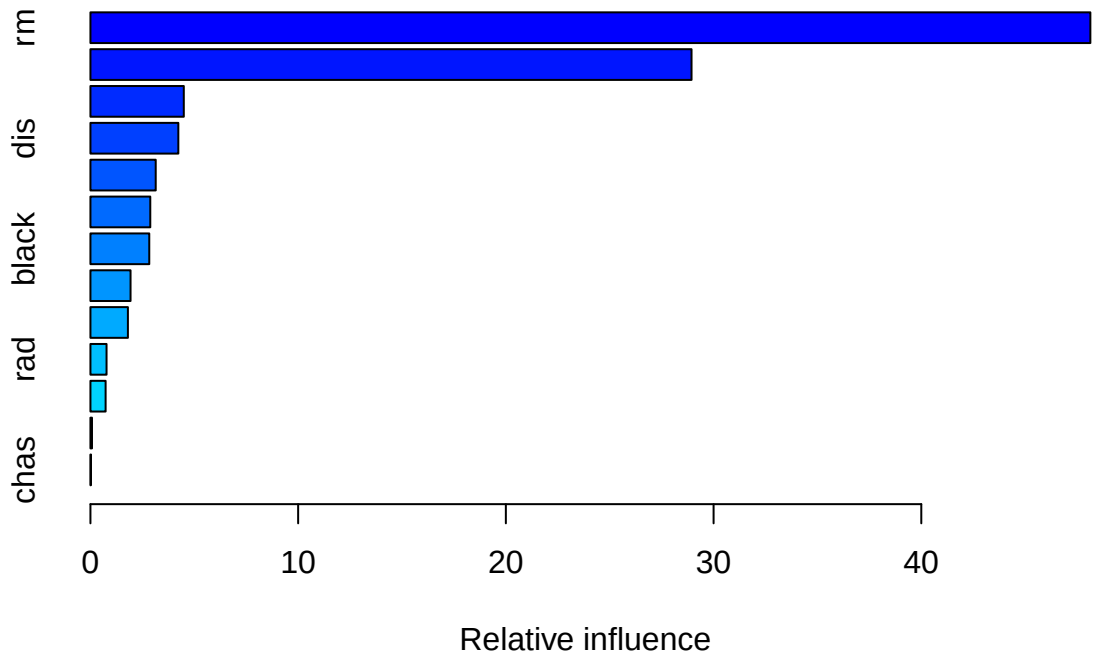
```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```
library(MASS)
library(randomForest)
```

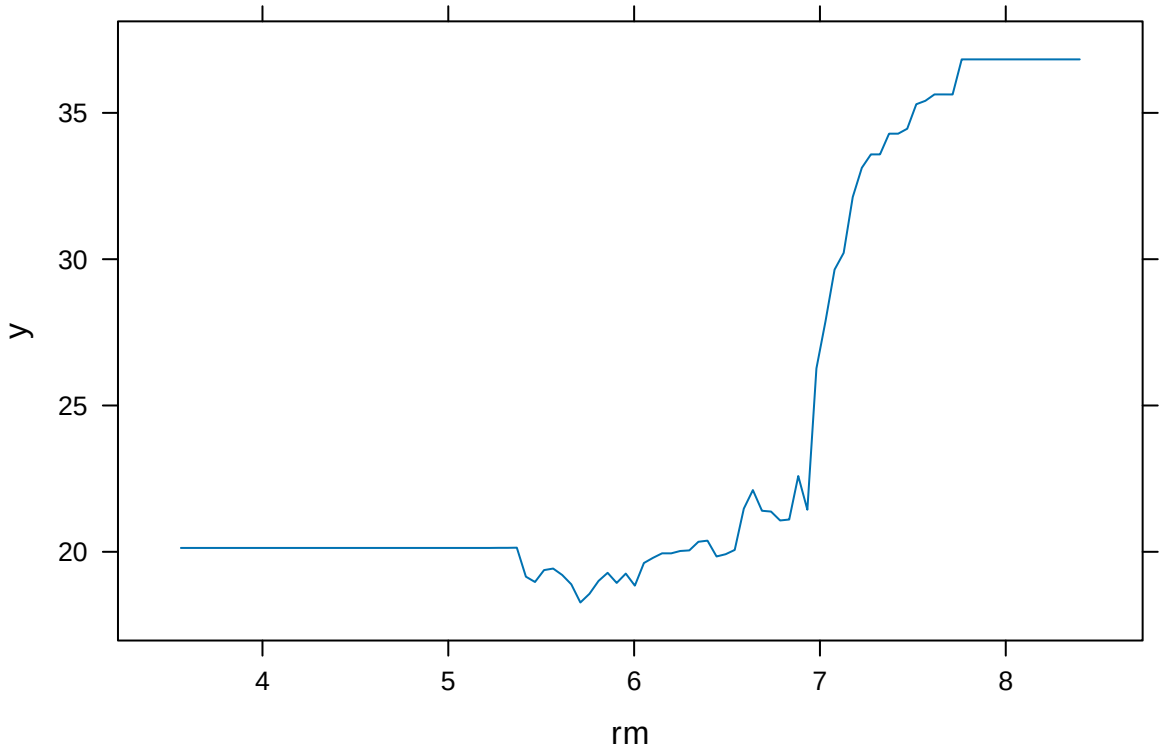
```
set.seed(1)
data(Boston)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
boost.boston <- gbm(medv ~ ., data = Boston[train, ], distribution = "gaussian", n.trees = 5000, intera
```

```
summary(boost.boston)
```

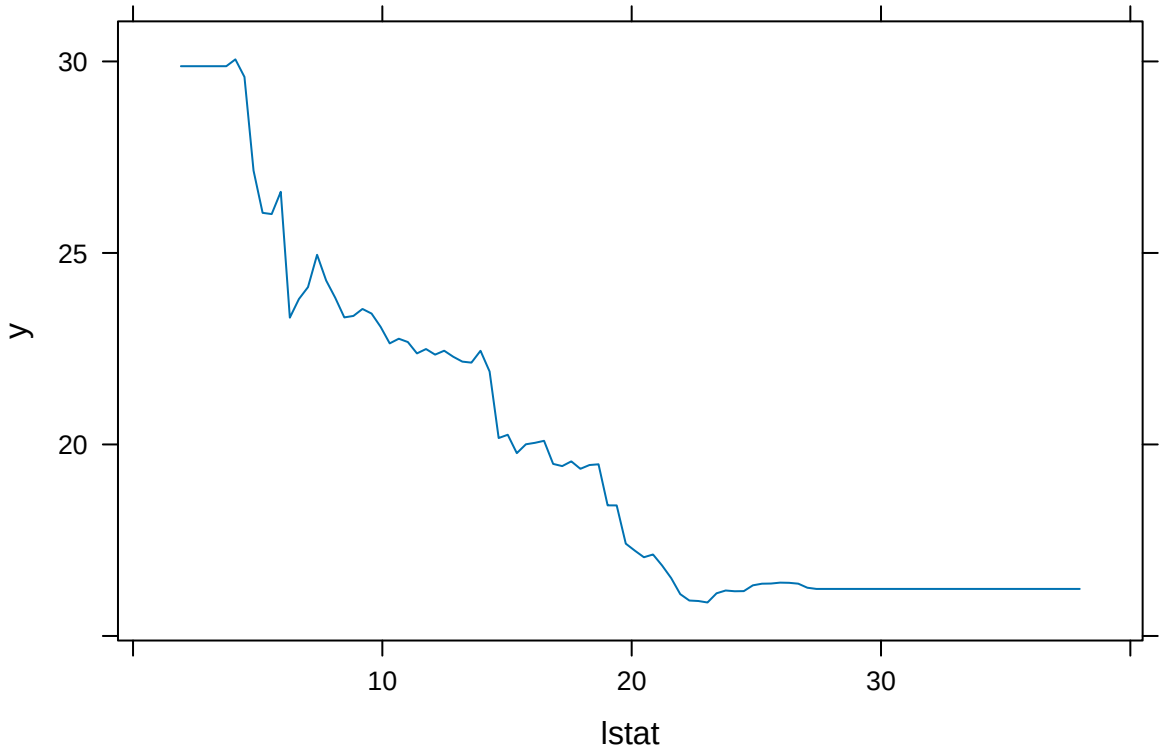



```
##             var     rel.inf
## rm           rm 48.13967682
## lstat     lstat 28.93851185
## crim       crim  4.49413146
## dis         dis  4.23182696
## age         age  3.14221169
## nox         nox  2.88094283
## black     black  2.83238772
## ptratio ptratio  1.93050932
## tax         tax  1.80427054
## rad         rad  0.77569461
## indus     indus  0.73110525
## zn           zn  0.07442923
## chas       chas  0.02430170
```

```
par(mfrow = c(1, 2))
plot(boost.boston, i = "rm")
```




```
plot(boost.boston, i = "lstat")
```




```
yhat.boost <- predict(boost.boston, newdata = Boston[-train, ], n.trees = 5000)
boston.test <- Boston[-train, "medv"]
mean((yhat.boost - boston.test)^2)
```

```
## [1] 19.37033
```

```
boost.boston <- gbm(medv ~ ., data = Boston[train, ], distribution = "gaussian", n.trees = 5000, intera
yhat.boost <- predict(boost.boston, newdata = Boston[-train, ], n.trees = 5000)
boston.test <- Boston[-train, "medv"]
mean((yhat.boost - boston.test)^2)
```

```
## [1] 18.68911
```

Applied

Arefeh Nayebi

2025-02-18

Question 8

In the lab, a classification tree was applied to the `Carseats` data set after converting `Sales` into a qualitative response variable. Now we will seek to predict `Sales` using regression trees and related approaches, treating the response as a quantitative variable.

a. Split the data set into a training set and a test set.

```
library(ggplot2)
library(ISLR2)
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

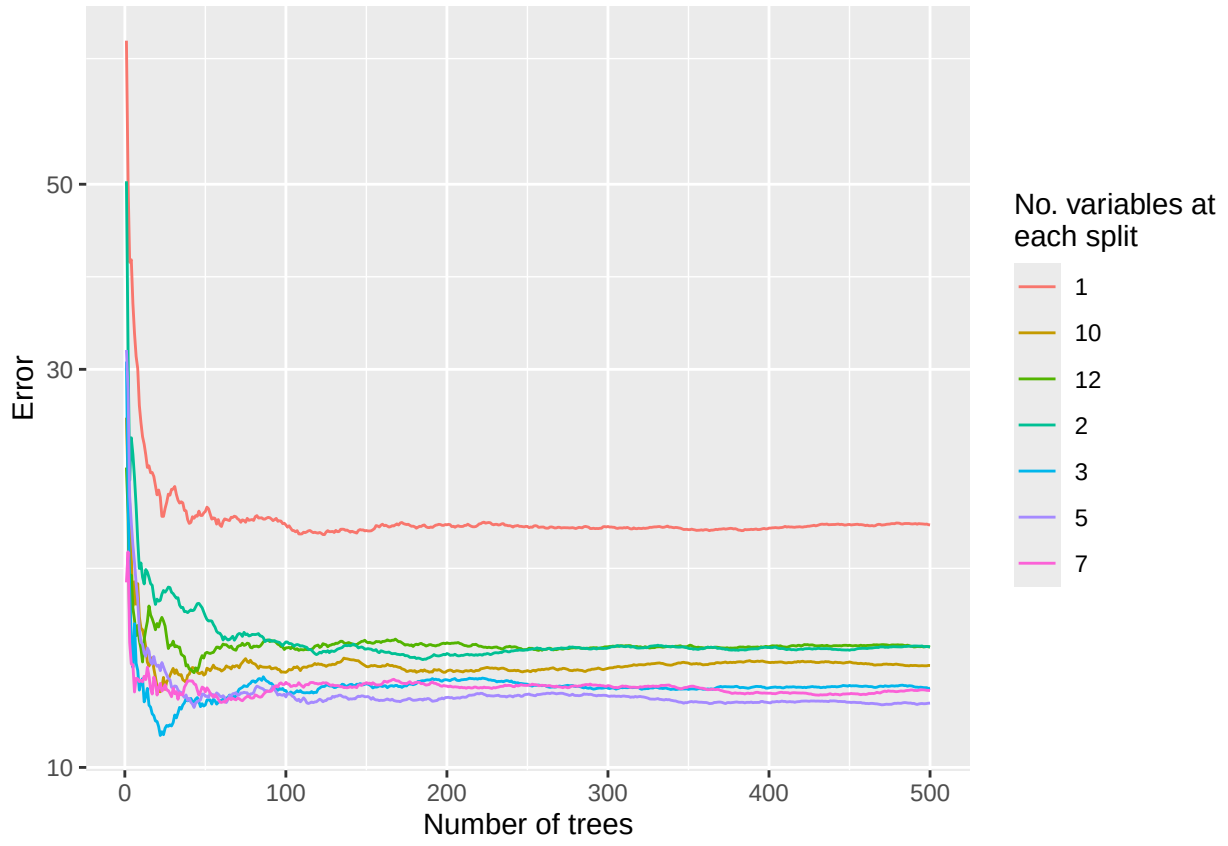
```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.4     v tibble    3.2.1
## v purrr     1.0.4     v tidyr     1.3.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::combine()       masks randomForest::combine()
## x dplyr::filter()        masks stats::filter()
## x dplyr::lag()           masks stats::lag()
## x randomForest::margin() masks ggplot2::margin()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
set.seed(42)

train <- sample(c(TRUE, FALSE), nrow(Boston), replace = TRUE)

rf_err <- function(mtry) {
  randomForest(
    Boston[train, -13],
    y = Boston[train, 13],
    xtest = Boston[!train, -13],
    ytest = Boston[!train, 13],
```

```
    mtry = mtry,
    ntree = 500
  )$test$mse
}
res <- lapply(c(1, 2, 3, 5, 7, 10, 12), rf_err)
names(res) <- c(1, 2, 3, 5, 7, 10, 12)
data.frame(res, check.names = FALSE) |>
  mutate(n = 1:500) |>
  pivot_longer(!n) |>
  ggplot(aes(x = n, y = value, color = name)) +
  geom_line(na.rm = TRUE) +
  xlab("Number of trees") +
  ylab("Error") +
  scale_y_log10() +
  scale_color_discrete(name = "No. variables at\neach split")
```



b. Fit a regression tree to the training set. Plot the tree, and interpret the results. What test error rate do you obtain?

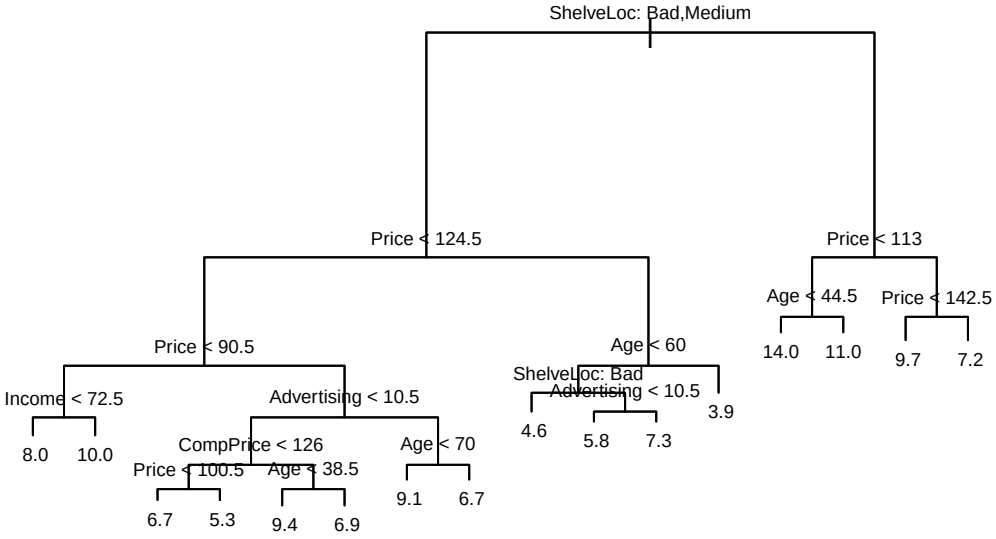
```
set.seed(42)
train <- sample(c(TRUE, FALSE), nrow(Carseats), replace = TRUE)
```

```
library(tree)
tr <- tree(Sales ~ ., data = Carseats[train, ])
summary(tr)
```

```
##
```

```
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats[train, ])
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "Advertising" "CompPrice"
## [6] "Age"
## Number of terminal nodes:  16
## Residual mean deviance:  2.356 = 424.1 / 180
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.54900 -0.82980  0.03075  0.00000  0.89250  4.83100
```

```
plot(tr)
text(tr, pretty = 0, digits = 2, cex = 0.6)
```



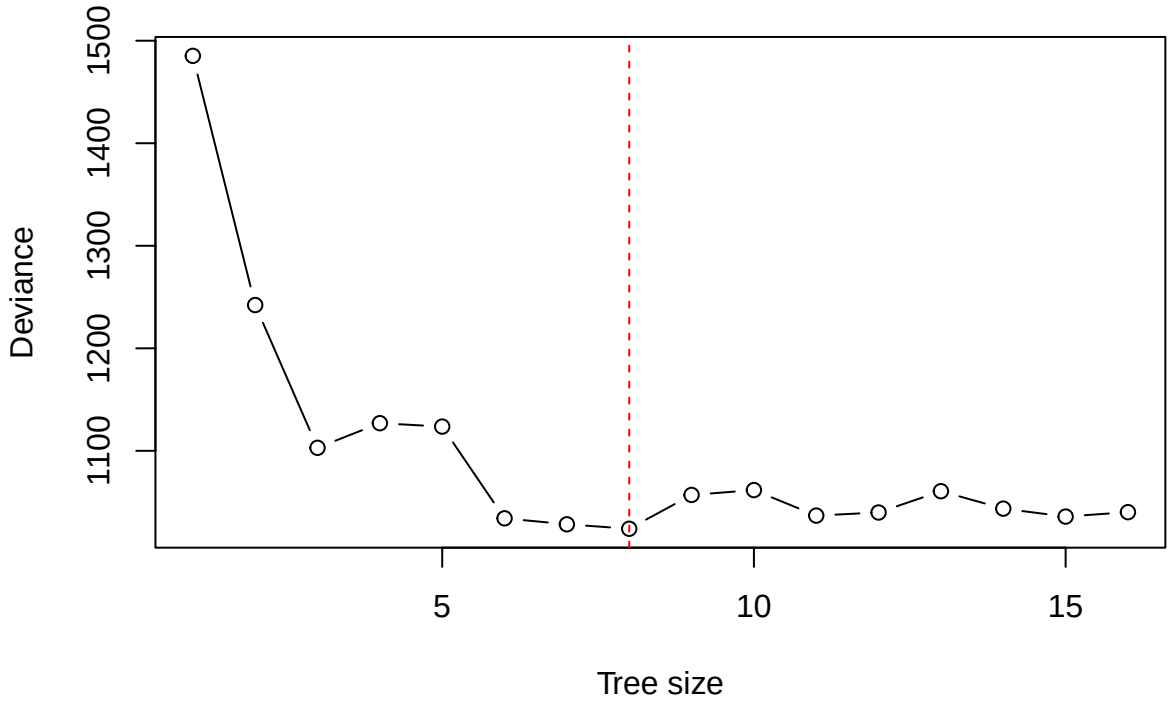

```
carseats_mse <- function(model) {
  p <- predict(model, newdata = Carseats[!train, ])
  mean((p - Carseats[!train, "Sales"])^2)
}
carseats_mse(tr)
```

```
## [1] 4.559764
```

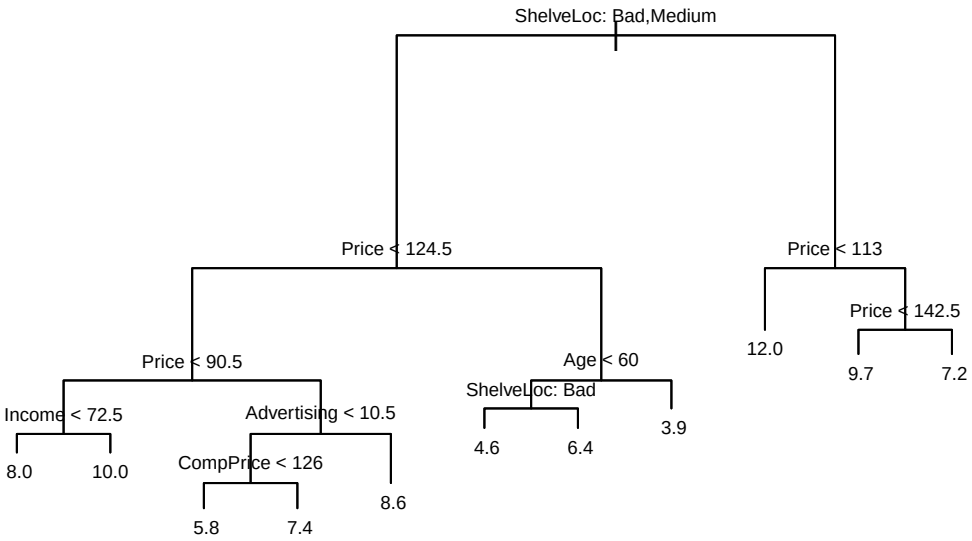
c. Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test error rate?

```
res <- cv.tree(tr)
plot(res$size, res$dev, type = "b", xlab = "Tree size", ylab = "Deviance")
min <- which.min(res$dev)
```

```r
abline(v = res$size[min], lty = 2, col = "red")
```




```r
ptr <- prune.tree(tr, best = 11)
plot(ptr)
text(ptr, pretty = 0, digits = 2, cex = 0.6)
```




```
carseats_mse(ptr)
```

```
## [1] 4.625875
```

d. Use the bagging approach in order to analyze this data. What test error rate do you obtain? Use the `importance()` function to determine which variables are most important.

```
# Here we can use random Forest with mtry = 10 = p (the number of predictor
# variables) to perform bagging
bagged <- randomForest(Sales ~ .,
  data = Carseats[train, ], mtry = 10,
  ntree = 200, importance = TRUE
)
carseats_mse(bagged)
```

```
## [1] 2.762861
```

```
importance(bagged)
```

```
##                %IncMSE IncNodePurity
## CompPrice   11.2608998    104.474222
## Income       5.0953983     73.275066
## Advertising 12.9011190    125.886762
## Population   3.4071044     60.095200
## Price       34.6904380    450.952728
## ShelveLoc   33.7059874    374.808575
## Age          7.9101141    143.652934
## Education   -2.1154997     32.712444
```

```
## Urban        0.9604097      7.029648
## US           3.1336559      6.287048
```

e. Use random forests to analyze this data. What test error rate do you obtain? Use the `importance()` function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
rf <- randomForest(Sales ~ .,
  data = Carseats[train, ], mtry = 3,
  ntree = 500, importance = TRUE
)
carseats_mse(rf)
```

```
## [1] 3.439357
```

```
importance(rf)
```

```
##                %IncMSE IncNodePurity
## CompPrice    8.5717587     122.75189
## Income       2.8955756     116.33951
## Advertising 13.0681194     128.13563
## Population   2.0475415     104.03803
## Price       34.7934136     342.84663
## ShelveLoc   39.0704834     292.56638
## Age          7.7941744     135.69061
## Education    0.8770806      64.67614
## Urban       -0.3301478      13.83594
## US           6.2716539      22.07306
```

f. Now analyze the data using BART, and report your results.

```
library(BART)
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## Loading required package: survival
```

```
# For ease, we'll create a fake "predict" method that just returns
# yhat.test.mean regardless of provided "newdata"
predict.wbart <- function(model, ...) model$yhat.test.mean

bartfit <- gbart(Carseats[train, 2:11], Carseats[train, 1],
  x.test = Carseats[!train, 2:11]
)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 196, 14, 204
## y1,yn: 2.070867, 2.280867
## x1,x[n*p]: 138.000000, 1.000000
## xp1,xp[np*p]: 141.000000, 1.000000
## *****Number of Trees: 200
## *****Number of Cut Points: 58 ... 1
```

```
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.287616,3,0.21118,7.42913
## *****sigma: 1.041218
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,14,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 11s
## trcnt,tecnt: 1000,1000
```

```
carseats_mse(bartfit)
```

```
## [1] 1.631285
```

Question 9

This problem involves the `OJ` data set which is part of the `ISLR2` package.

a. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
set.seed(42)
train <- sample(1:nrow(OJ), 800)
test <- setdiff(1:nrow(OJ), train)
```

b. Fit a tree to the training data, with `Purchase` as the response and the other variables except for `Buy` as predictors. Use the `summary()` function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
tr <- tree(Purchase ~ ., data = OJ[train, ])
summary(tr)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ[train, ])
## Variables actually used in tree construction:
## [1] "LoyalCH"     "SalePriceMM" "PriceDiff"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7392 = 585.5 / 792
## Misclassification error rate: 0.1638 = 131 / 800
```

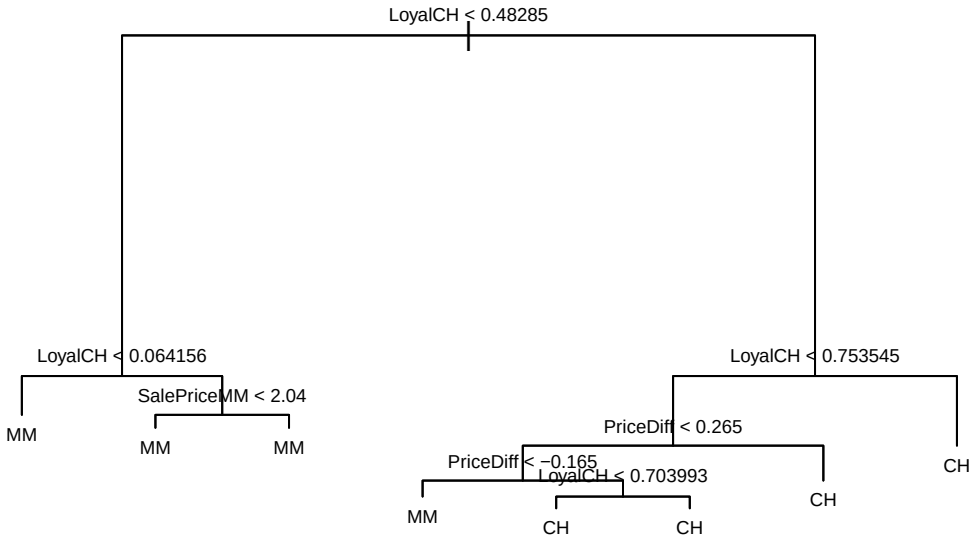
c. Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

```
tr
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1066.00 CH ( 0.61500 0.38500 )
##    2) LoyalCH < 0.48285 285  296.00 MM ( 0.21404 0.78596 )
##      4) LoyalCH < 0.064156 64    0.00 MM ( 0.00000 1.00000 ) *
##      5) LoyalCH > 0.064156 221  260.40 MM ( 0.27602 0.72398 )
##       10) SalePriceMM < 2.04 128  123.50 MM ( 0.18750 0.81250 ) *
##       11) SalePriceMM > 2.04 93  125.00 MM ( 0.39785 0.60215 ) *
##    3) LoyalCH > 0.48285 515  458.10 CH ( 0.83689 0.16311 )
##      6) LoyalCH < 0.753545 230  282.70 CH ( 0.69565 0.30435 )
##       12) PriceDiff < 0.265 149  203.00 CH ( 0.57718 0.42282 )
##         24) PriceDiff < -0.165 32   38.02 MM ( 0.28125 0.71875 ) *
##         25) PriceDiff > -0.165 117  150.30 CH ( 0.65812 0.34188 )
##           50) LoyalCH < 0.703993 105  139.60 CH ( 0.61905 0.38095 ) *
##           51) LoyalCH > 0.703993 12    0.00 CH ( 1.00000 0.00000 ) *
##       13) PriceDiff > 0.265 81   47.66 CH ( 0.91358 0.08642 ) *
##      7) LoyalCH > 0.753545 285  111.70 CH ( 0.95088 0.04912 ) *
```

d. Create a plot of the tree, and interpret the results.

```
plot(tr)
text(tr, pretty = 0, digits = 2, cex = 0.6)
```



> e. Predict the response on the test data, and produce a confusion matrix > comparing the test labels to the predicted test labels. What is the test > error rate?

```
table(predict(tr, OJ[test, ], type = "class"), OJ[test, "Purchase"])
```

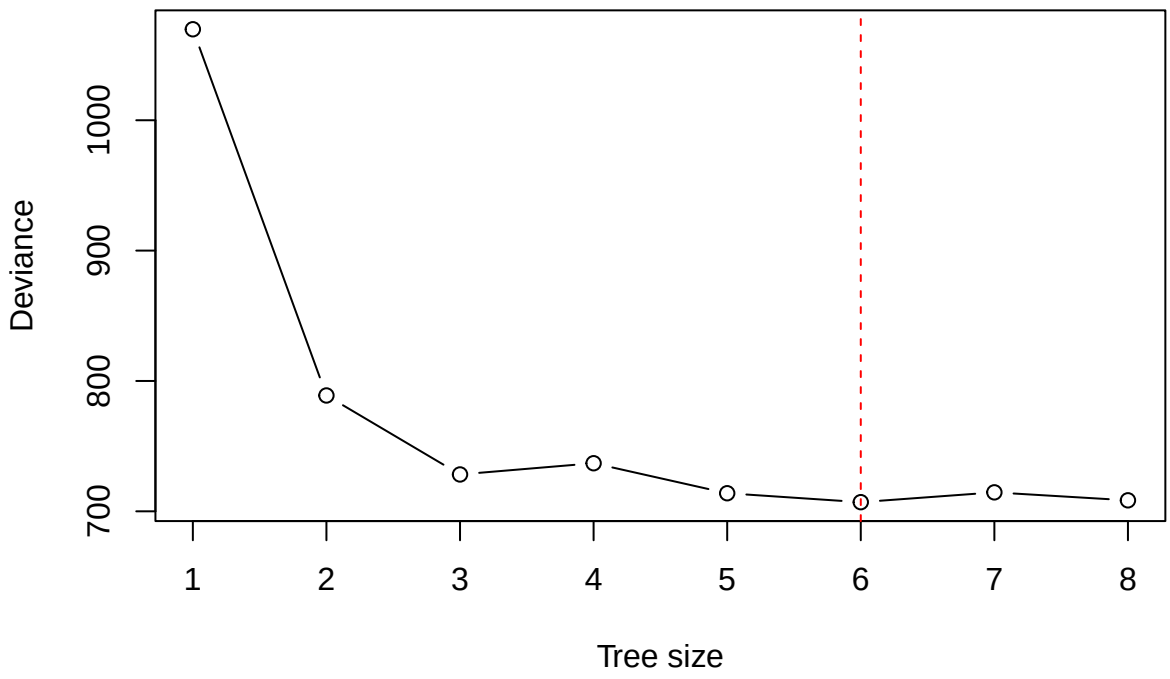
```
##
##       CH  MM
##   CH 125  15
##   MM  36  94
```

f. Apply the `cv.tree()` function to the training set in order to determine the optimal tree size.

```
set.seed(42)
res <- cv.tree(tr)
```

g. Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(res$size, res$dev, type = "b", xlab = "Tree size", ylab = "Deviance")
min <- which.min(res$dev)
abline(v = res$size[min], lty = 2, col = "red")
```



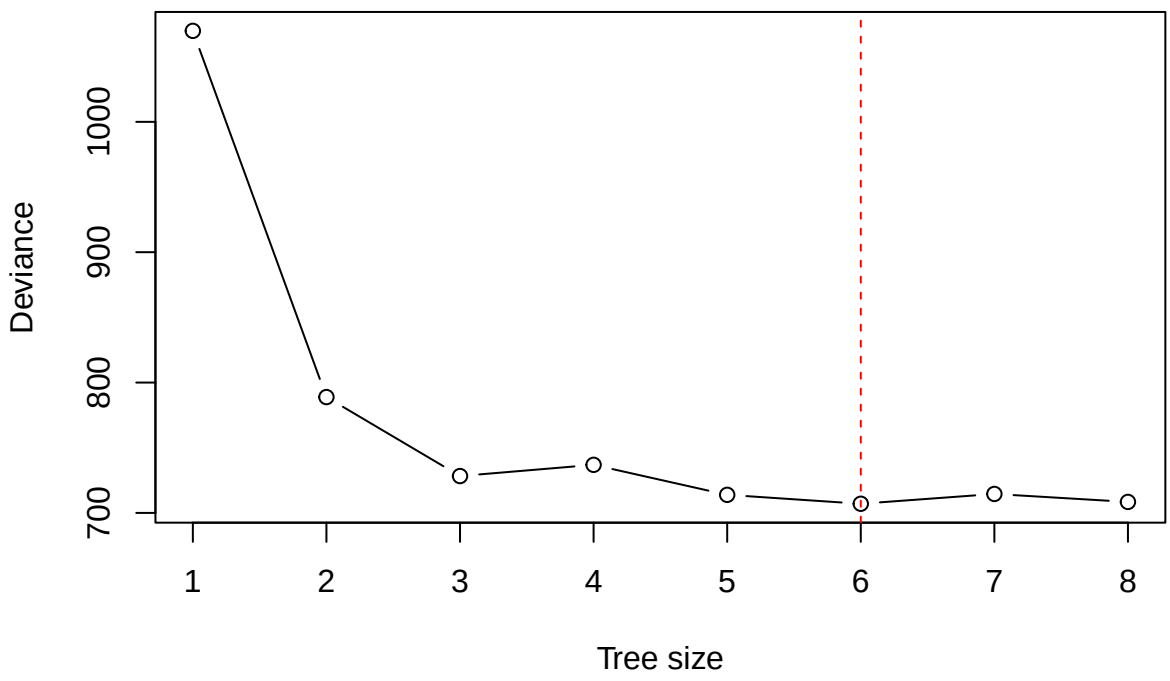
f. Apply the `cv.tree()` function to the training set in order to determine the optimal tree size.

```
set.seed(42)
res <- cv.tree(tr)
```

g. Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(res$size, res$dev, type = "b", xlab = "Tree size", ylab = "Deviance")
min <- which.min(res$dev)
```

```
abline(v = res$size[min], lty = 2, col = "red")
```



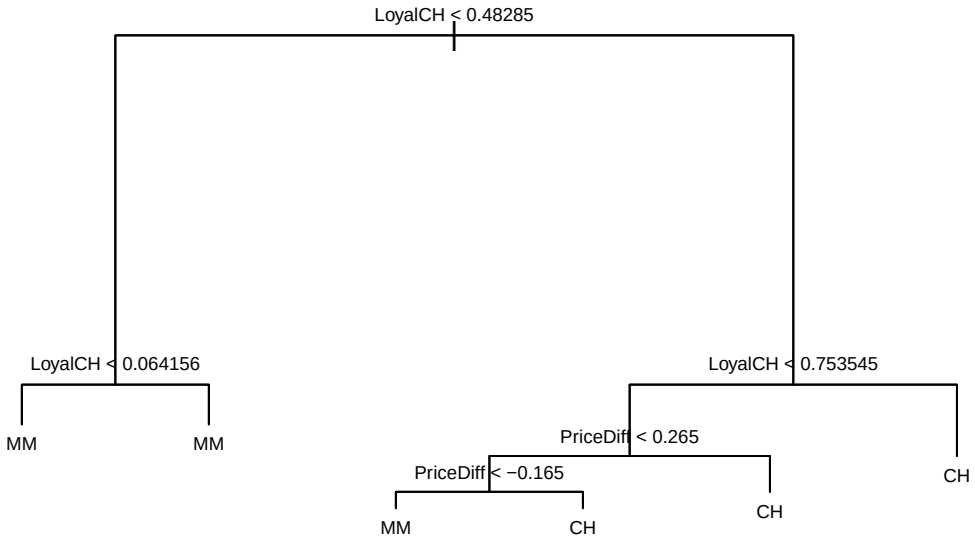
h. Which tree size corresponds to the lowest cross-validated classification error rate?

```
res$size[min]
```

```
## [1] 6
```

i. Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
ptr <- prune.tree(tr, best = res$size[min])
plot(ptr)
text(ptr, pretty = 0, digits = 2, cex = 0.6)
```



j. Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
oj_misclass <- function(model) {
  summary(model)$misclass[1] / summary(model)$misclass[2]
}
oj_misclass(tr)
```

```
## [1] 0.16375
```

```
oj_misclass(ptr)
```

```
## [1] 0.16375
```

k. Compare the test error rates between the pruned and unpruned trees. Which is higher?

```
oj_err <- function(model) {
  p <- predict(model, newdata = OJ[test, ], type = "class")
  mean(p != OJ[test, "Purchase"])
}
oj_err(tr)
```

```
## [1] 0.1888889
```

```
oj_err(ptr)
```

```
## [1] 0.1888889
```

Question 10

We now use boosting to predict `Salary` in the `Hitters` data set.

a. Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

```
dat <- Hitters
dat <- dat[!is.na(dat$Salary), ]
dat$Salary <- log(dat$Salary)
```

b. Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

```
train <- 1:200
test <- setdiff(1:nrow(dat), train)
```

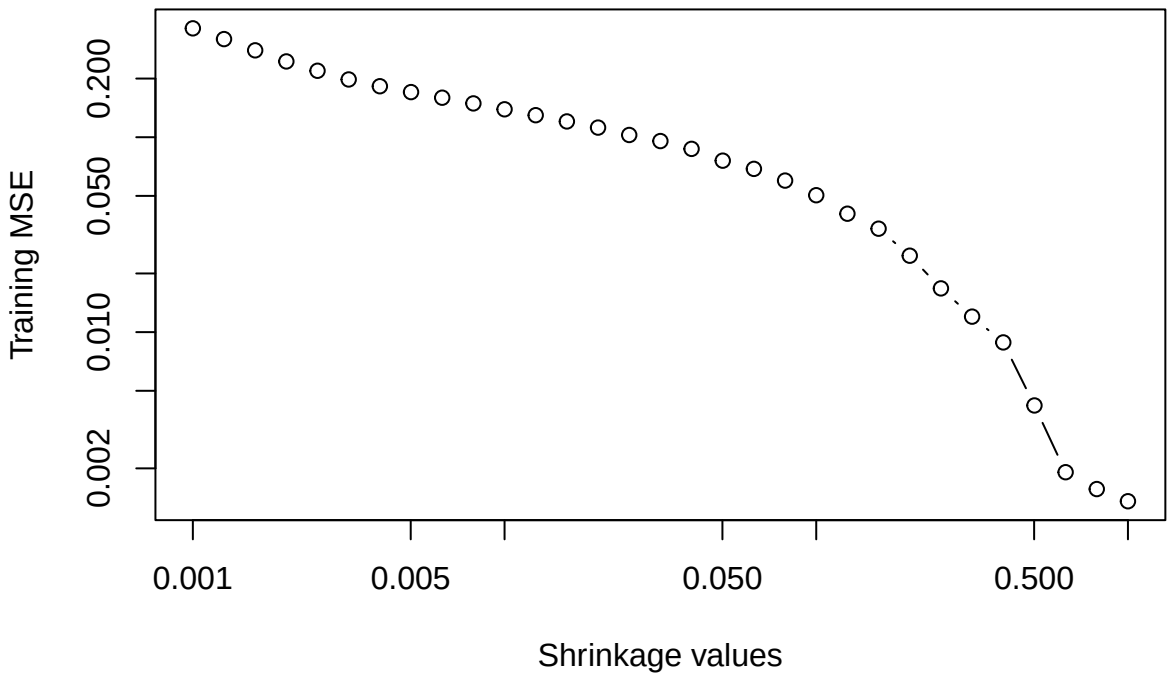
c. Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
library(gbm)
```

```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```
set.seed(42)
lambdas <- 10^seq(-3, 0, by = 0.1)
fits <- lapply(lambdas, function(lam) {
  gbm(Salary ~ .,
    data = dat[train, ], distribution = "gaussian",
    n.trees = 1000, shrinkage = lam
  )
})
errs <- sapply(fits, function(fit) {
  p <- predict(fit, dat[train, ], n.trees = 1000)
  mean((p - dat[train, ]$Salary)^2)
})
plot(lambdas, errs,
  type = "b", xlab = "Shrinkage values",
  ylab = "Training MSE", log = "xy"
)
```

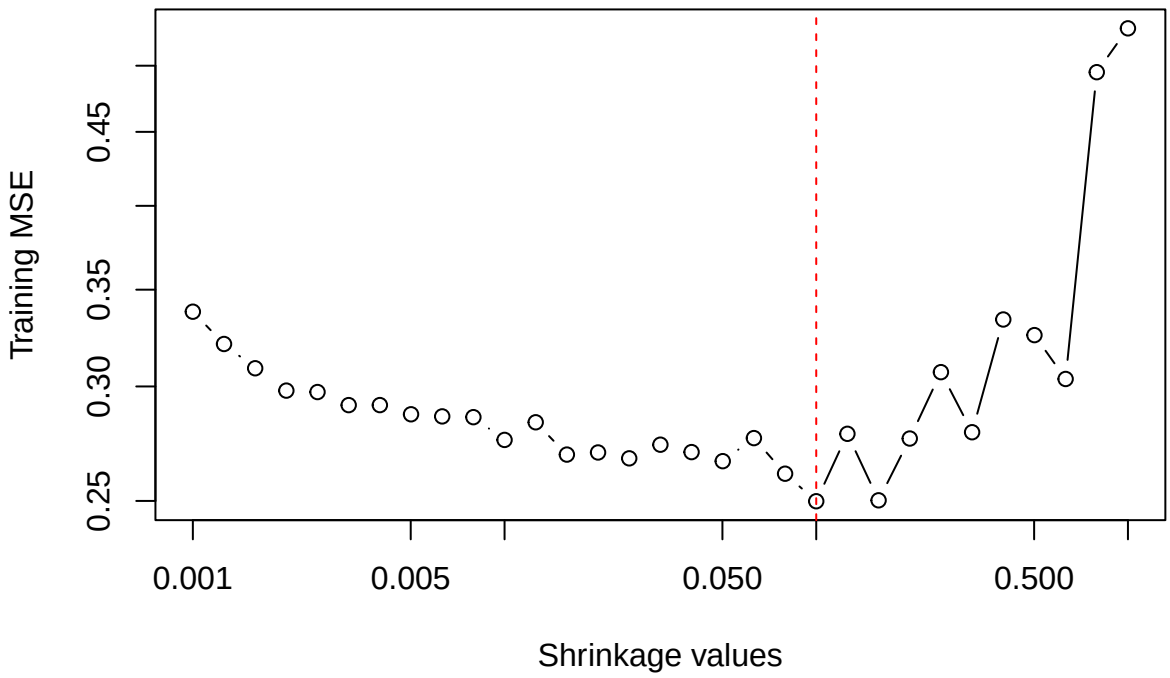


d. Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axi

```
errs <- sapply(fits, function(fit) {
  p <- predict(fit, dat[test, ], n.trees = 1000)
  mean((p - dat[test, ]$Salary)^2)
})
plot(lambdas, errs,
  type = "b", xlab = "Shrinkage values",
  ylab = "Training MSE", log = "xy"
)
min(errs)
```

```
## [1] 0.249881
```

```
abline(v = lambdas[which.min(errs)], lty = 2, col = "red")
```



e. Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

```
fit1 <- lm(Salary ~ ., data = dat[train, ])
mean((predict(fit1, dat[test, ]) - dat[test, "Salary"])^2)
```

```
## [1] 0.4917959
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

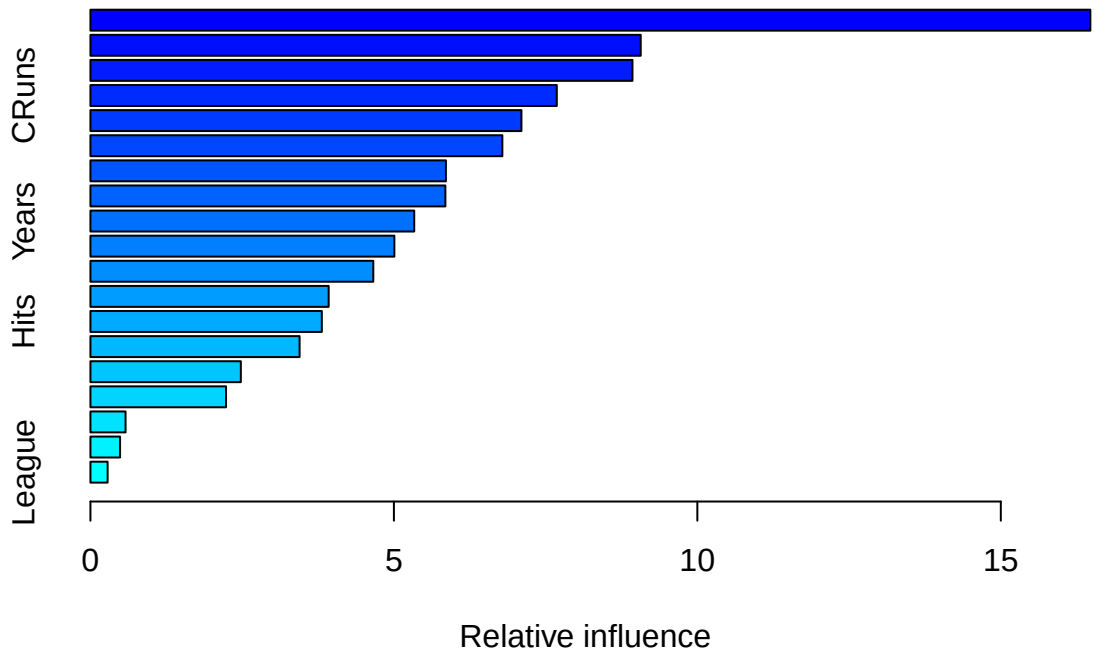
```
## Loaded glmnet 4.1-8
```

```
x <- model.matrix(Salary ~ ., data = dat[train, ])
x.test <- model.matrix(Salary ~ ., data = dat[test, ])
y <- dat[train, "Salary"]
fit2 <- glmnet(x, y, alpha = 1)
mean((predict(fit2, s = 0.1, newx = x.test) - dat[test, "Salary"])^2)
```

```
## [1] 0.4389054
```

f. Which variables appear to be the most important predictors in the boosted model?

```
summary(fits[[which.min(errs)]])
```




```
##                 var    rel.inf
## CAtBat       CAtBat 16.4755242
## CRBI           CRBI  9.0670759
## CHits         CHits  8.9307168
## CRuns         CRuns  7.6839786
## CWalks       CWalks  7.1014886
## PutOuts     PutOuts  6.7869382
## AtBat         AtBat  5.8567916
## Walks         Walks  5.8479836
## Years         Years  5.3349489
## Assists     Assists  5.0076392
## CHmRun       CHmRun  4.6606919
## RBI             RBI  3.9255396
## Hits           Hits  3.8123124
## HmRun         HmRun  3.4462640
## Runs           Runs  2.4779866
## Errors       Errors  2.2341326
## NewLeague NewLeague  0.5788283
## Division   Division  0.4880237
## League       League  0.2831352
```

g. Now apply bagging to the training set. What is the test set MSE for this approach?

```
set.seed(42)
bagged <- randomForest(Salary ~ ., data = dat[train, ], mtry = 19, ntree = 1000)
```

```
mean((predict(bagged, newdata = dat[test, ]) - dat[test, "Salary"])^2)
```

```
## [1] 0.2278813
```

Question 11

This question uses the `Caravan` data set.

a. Create a training set consisting of the first 1,000 observations, and a test set consisting of the remaining observations.

```
train <- 1:1000
test <- setdiff(1:nrow(Caravan), train)
```

b. Fit a boosting model to the training set with `Purchase` as the response and the other variables as predictors. Use 1,000 trees, and a shrinkage value of 0.01. Which predictors appear to be the most important?

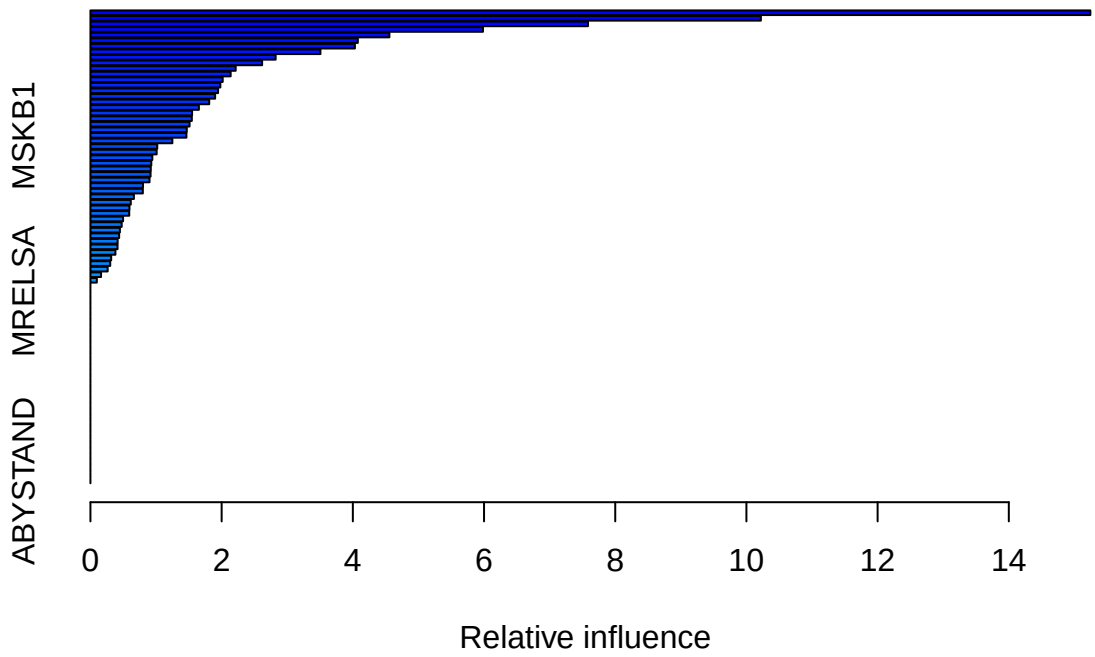
```
set.seed(42)
fit <- gbm(as.numeric(Purchase == "Yes") ~ ., data = Caravan[train, ], n.trees = 1000, shrinkage = 0.01
```

```
## Distribution not specified, assuming bernoulli ...
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution,
## : variable 50: PVRAAUT has no variation.
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution,
## : variable 71: AVRAAUT has no variation.
```

```
head(summary(fit))
```



```
##               var   rel.inf
## PPERSAUT PPERSAUT 15.243041
## MKOOPKLA MKOOPKLA 10.220498
## MOPLHOOG MOPLHOOG  7.584734
## MBERMIDD MBERMIDD  5.983650
## PBRAND     PBRAND  4.557491
## ABRAND     ABRAND  4.076017
```

c. Use the boosting model to predict the response on the test data. Predict that a person will make a purchase if the estimated probability of purchase is greater than 20%. Form a confusion matrix. What fraction of the people predicted to make a purchase do in fact make one? How does this compare with the results obtained from applying KNN or logistic regression to this data set?

```
p <- predict(fit, Caravan[test, ], n.trees = 1000, type = "response")
table(p > 0.2, Caravan[test, "Purchase"] == "Yes")
```

```
##
##         FALSE TRUE
##   FALSE  4415  257
##   TRUE    118   32
```

```
sum(p > 0.2 & Caravan[test, "Purchase"] == "Yes") / sum(p > 0.2)
```

```
## [1] 0.2133333
```

```
# Logistic regression
fit <- glm(Purchase == "Yes" ~ ., data = Caravan[train, ], family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
p <- predict(fit, Caravan[test, ], type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
table(p > 0.2, Caravan[test, "Purchase"] == "Yes")
```

```
##
##         FALSE TRUE
##   FALSE  4183  231
##   TRUE    350   58
```

```
sum(p > 0.2 & Caravan[test, "Purchase"] == "Yes") / sum(p > 0.2)
```

```
## [1] 0.1421569
```

```
library(class)
# KNN
fit <- knn(Caravan[train, -86], Caravan[test, -86], Caravan$Purchase[train])
table(fit, Caravan[test, "Purchase"] == "Yes")
```

```
##
## fit   FALSE TRUE
##   No   4260  263
##   Yes   273   26
```

```
sum(fit == "Yes" & Caravan[test, "Purchase"] == "Yes") / sum(fit == "Yes")
```

```
## [1] 0.08695652
```

Question 12

Apply boosting, bagging, random forests and BART to a data set of your choice. Be sure to fit the models on a training set and to evaluate their performance on a test set. How accurate are the results compared to simple methods like linear or logistic regression? Which of these approaches yields the best performance?

```
library(gam)
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loaded gam 1.22-5
```

```
set.seed(42)
train <- sample(1:nrow(College), 400)
test <- setdiff(1:nrow(College), train)

# Linear regression
lr <- gam(Outstate ~ ., data = College[train, ])

# GAM from chapter 7
gam <- gam(Outstate ~ Private + s(Room.Board, 2) + s(PhD, 2) +
  s(perc.alumni, 2) + s(Expend, 2) + s(Grad.Rate, 2), data = College[train, ])

# Boosting
boosted <- gbm(Outstate ~ ., data = College[train, ], n.trees = 1000, shrinkage = 0.01)
```

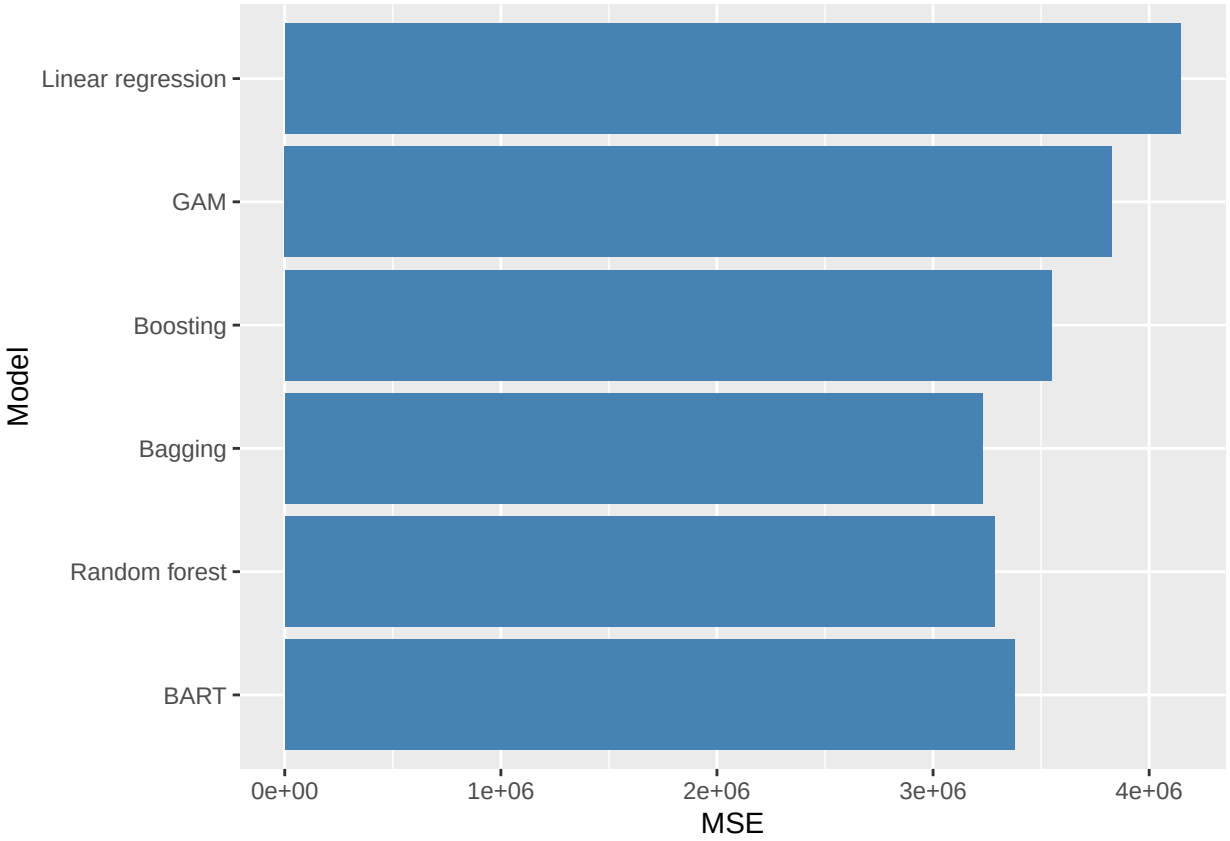
```
## Distribution not specified, assuming gaussian ...
```

```
# Bagging (random forest with mtry = no. predictors)
bagged <- randomForest(Outstate ~ ., data = College[train, ], mtry = 17, ntree = 1000)

# Random forest with mtry = sqrt(no. predictors)
rf <- randomForest(Outstate ~ ., data = College[train, ], mtry = 4, ntree = 1000)

# BART
pred <- setdiff(colnames(College), "Outstate")
bart <- gbart(College[train, pred], College[train, "Outstate"],
  x.test = College[test, pred]
)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 400, 18, 377
## y1,yn: -4030.802500, 77.197500
## x1,x[n*p]: 1.000000, 71.000000
## xp1,xp[np*p]: 0.000000, 99.000000
## *****Number of Trees: 200
## *****Number of Cut Points: 1 ... 75
## *****burn,nd,thin: 100,1000,1
```

```
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,301.581,3,715815,10580.8
## *****sigma: 1916.969943
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,18,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 15s
## trcnt,tecnt: 1000,1000
```

```r
mse <- function(model, ...) {
  pred <- predict(model, College[test, ], ...)
  mean((College$Outstate[test] - pred)^2)
}

res <- c(
  "Linear regression" = mse(lr),
  "GAM" = mse(gam),
  "Boosting" = mse(boosted, n.trees = 1000),
  "Bagging" = mse(bagged),
  "Random forest" = mse(rf),
  "BART" = mse(bart)
)
res <- data.frame("MSE" = res)
res$Model <- factor(row.names(res), levels = rev(row.names(res)))
ggplot(res, aes(Model, MSE)) +
  coord_flip() +
  geom_bar(stat = "identity", fill = "steelblue")
```


Linear regression
GAM
Boosting
Bagging
Random forest
BART
Model
0e+00
1e+06
2e+06
3e+06
4e+06
MSE