Week3

Anyi Guo

26/12/2018

Week 3

Regression with Trees

Pros: better interpretability, better performance for non-linear settings

Stop splitting when the leaves are pure ### Measures of impurity 1. Misclassification Error: * 0 = perfect purity * 0.5 = no purity

2. Gini index:

- 0 = perfect purity
- 0.5 = no purity

3. Deviance/information gain:

- 0 = perfect purity
- 1 = no purity

Example: Iris Data

```
data(iris)
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
names(iris)
```

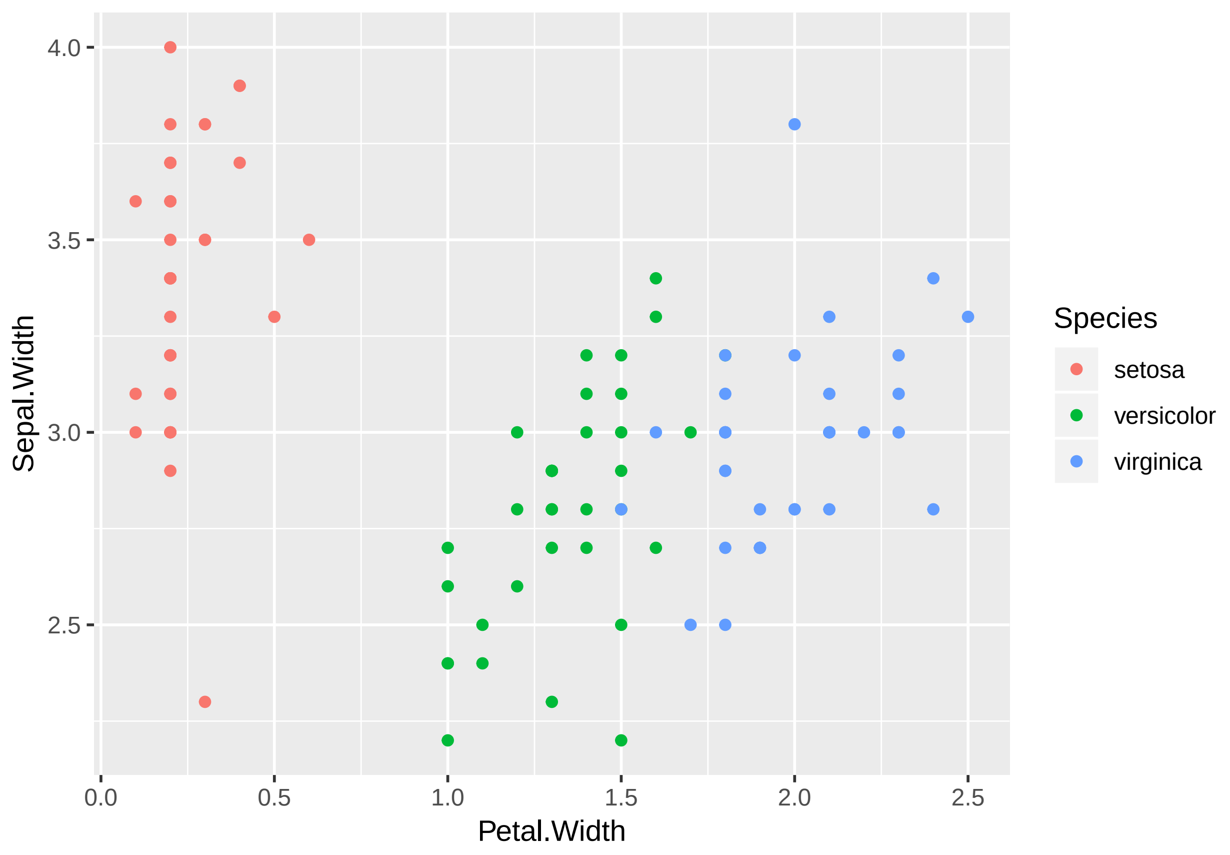
```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
inTrain<-createDataPartition(y=iris$Species,p=0.7,list=FALSE)
training<-iris[inTrain,]
testing<-iris[-inTrain,]

# plot the Iris petal widths/species
qplot(Petal.Width,Sepal.Width,col=Species, data=training)
```

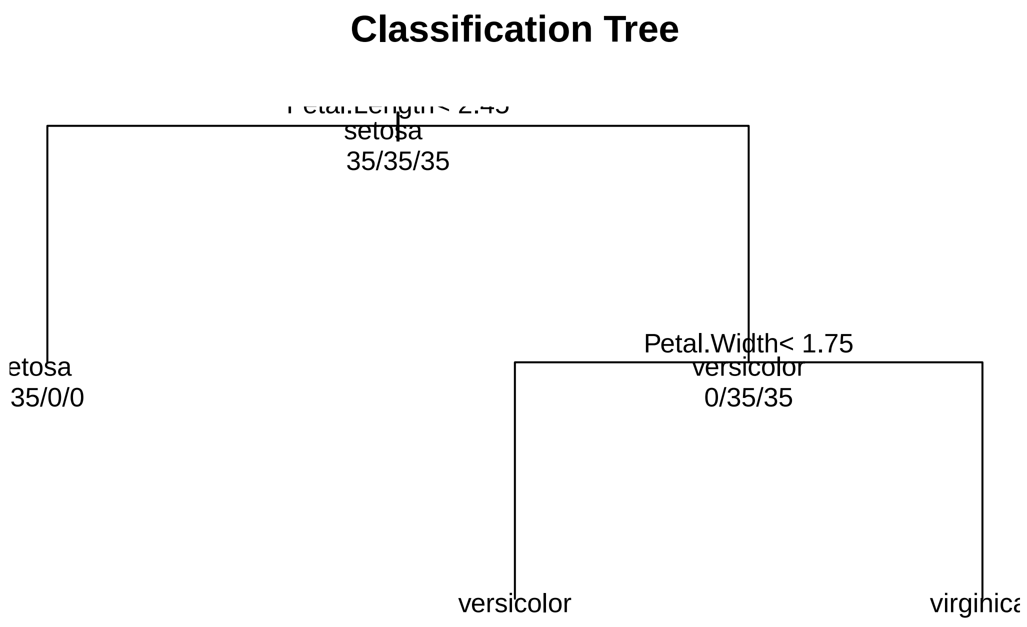


Train the model

```
#rpart is R's package for doing regressions
modFit<-train(Species~.,method="rpart",data=training)
print(modFit$finalModel)
```

```
## n= 105
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 105 70 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 35  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 70 35 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.75 37  3 versicolor (0.00000000 0.91891892 0.08108108) *
##     7) Petal.Width>=1.75 33  1 virginica (0.00000000 0.03030303 0.96969697) *
```

```
# plot tree
plot(modFit$finalModel,uniform=TRUE,main="Classification Tree")
text(modFit$finalModel,use.n=TRUE,all=TRUE,cex=0.8)
```

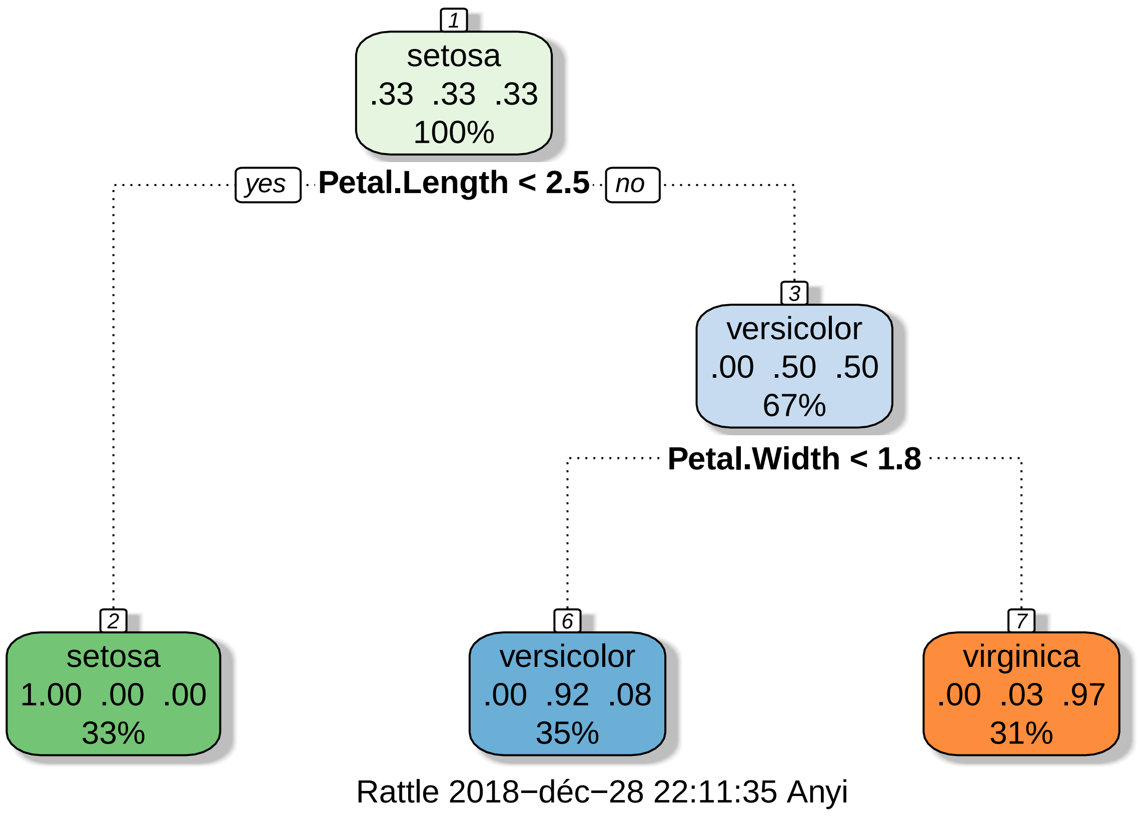


Use the rattle package to make the trees look better

```r
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Entrez 'rattle()' pour secouer, faire vibrer, et faire défiler vos données.
```

```r
fancyRpartPlot(modFit$finalModel)
```



Predict new values

```r
predict(modFit,newdata=testing)
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa    
##  [7] setosa     setosa     setosa     setosa     setosa     setosa    
## [13] setosa     setosa     setosa     versicolor versicolor versicolor
## [19] versicolor versicolor versicolor versicolor versicolor versicolor
## [25] versicolor versicolor versicolor versicolor versicolor versicolor
## [31] virginica  virginica  virginica  virginica  virginica  virginica 
## [37] versicolor virginica  virginica  versicolor virginica  virginica 
## [43] virginica  virginica  virginica 
## Levels: setosa versicolor virginica
```

Notes: Classification trees are non-linear models * They use interaction between variables * Tree can also be used for regression problems (i.e. continuous outcome)

Bagging (Bootstrap aggregating)

What is bagging? 1. Resample cases and recalculate predictions 2. Average or majority vote 3. It produces similar bias, but reduces variance. 4. Bagging is more useful for non-linear functions

Example with the Ozone data from ElemStatLearn package

```r
library(ElemStatLearn)
data(ozone,package="ElemStatLearn")
ozone<-ozone[order(ozone$ozone),]
```

We'll predict temperature based on zone

Bagged loess

```r
ll<-matrix(NA,nrow=10,ncol=155)

#we'll resample the data 10 times (loop 10 times)
for(i in 1:10){
        # each time we'll resample with replacement
        ss<-sample(1:dim(ozone)[1],replace=T)
        # ozone0 is the resampled subset. We'll also reorder the resampled subset with ozone
        ozone0<-ozone[ss,];ozone0<-ozone0[order(ozone0$ozone),]
        # we'll fit a loess line through the resampled subset. span determins how smooth this line woul
        loess0<-loess(temperature~ozone,data=ozone0,span=0.2)
        # for each of the loess curve, we'll predict the outcome for the 155 rows in the original datas
        ll[i,]<-predict(loess0,newdata=data.frame(ozone=1:155))
}
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 14
```

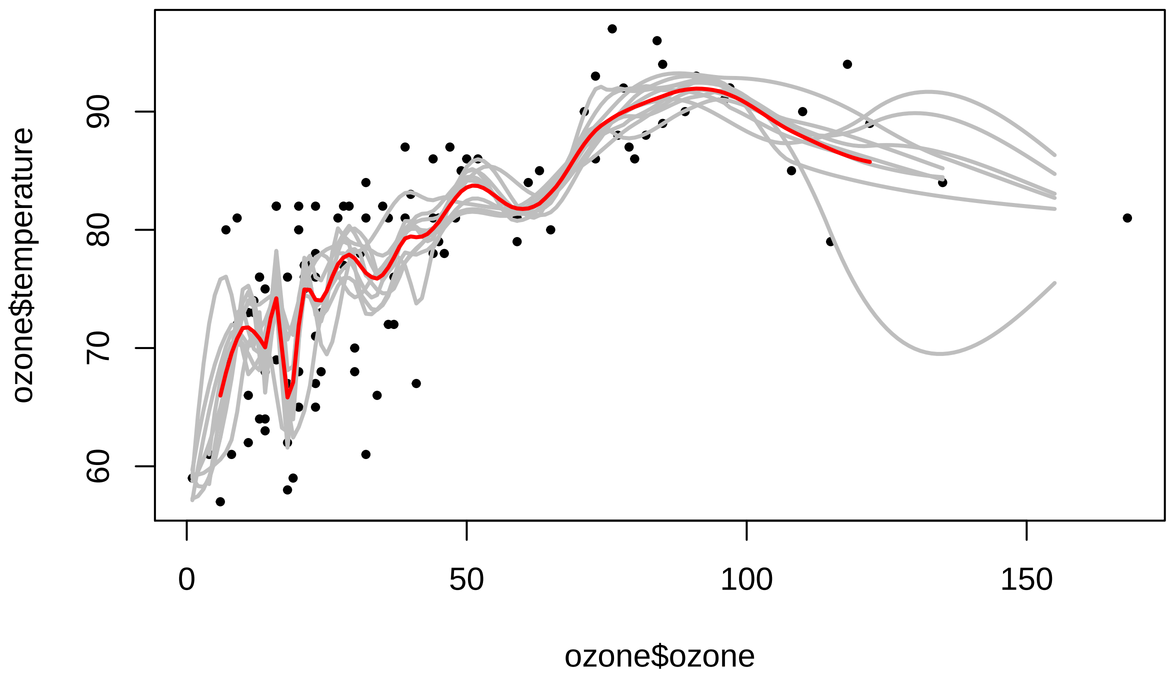
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 2
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 1.296e-16
```

Bagged loess

The red line is the bagged (average) line across the 10 resamples

```r
plot(ozone$ozone,ozone$temperature,pch=19,cex=0.5)
for(i in 1:10){lines(1:155,ll[i,],col="grey",lwd=2)}
lines(1:155,apply(ll,2,mean),col="red",lwd=2)
```



Notes: * Bagging is most useful for non-linear models * Often used with trees & random forests

Random Forests

What is random forests? 1. Bootstrap samples 2. At each split, bootstrap variables 3. Grow multiple trees and vote

Pros: 1. Accuracy

Cons: 1. Speed 2. Interpretability 3. Overfitting

Random Forest on Iris data

```
data(iris)
library(ggplot2)
library(caret)
inTrain<-createDataPartition(y=iris$Species,p=0.7,list=FALSE)
training<-iris[inTrain,]
testing<-iris[-inTrain,]

# build random forest model using caret
modFit<-train(Species~.,model="rf",prox=TRUE,data=training)
```

Getting a single tree

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

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

```
## The following object is masked from 'package:rattle':
##
##     importance
```

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

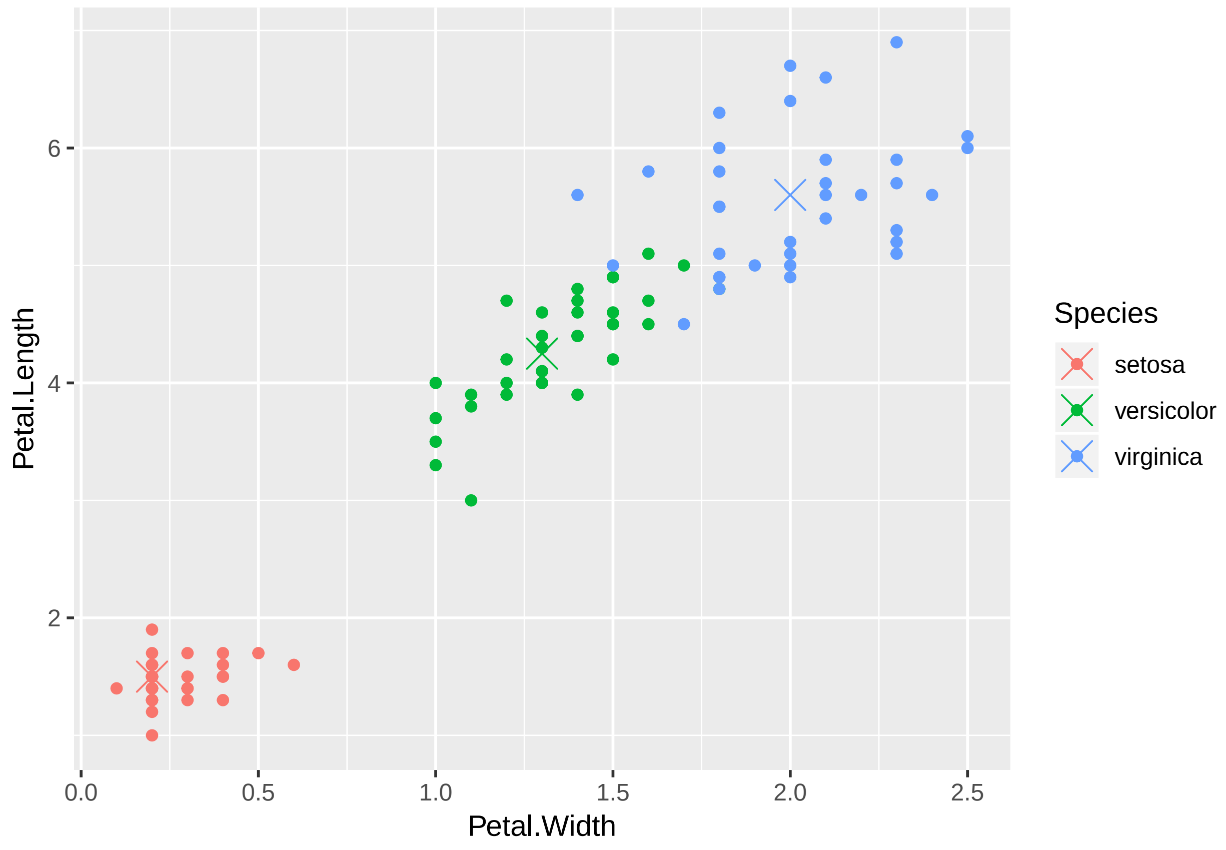
```
getTree(modFit$finalModel,k=2)
```

```
##   left daughter right daughter split var split point status prediction
## 1             2              3         4        1.75      1          0
## 2             4              5         4        0.80      1          0
## 3             0              0         0        0.00     -1          3
## 4             0              0         0        0.00     -1          1
## 5             6              7         1        4.95      1          0
## 6             0              0         0        0.00     -1          3
## 7             8              9         3        5.35      1          0
## 8             0              0         0        0.00     -1          2
## 9             0              0         0        0.00     -1          3
```

Class "centers"

```
irisP<-classCenter(training[,c(3,4)],training$Species,modFit$finalModel$prox)
irisP<-as.data.frame(irisP)
irisP$Species<-rownames(irisP)
p<-qplot(Petal.Width,Petal.Length,col=Species,data=training)

# This line plots the three centers
p+geom_point(aes(x=Petal.Width,y=Petal.Length,col=Species),size=5,shape=4,data=irisP)
```

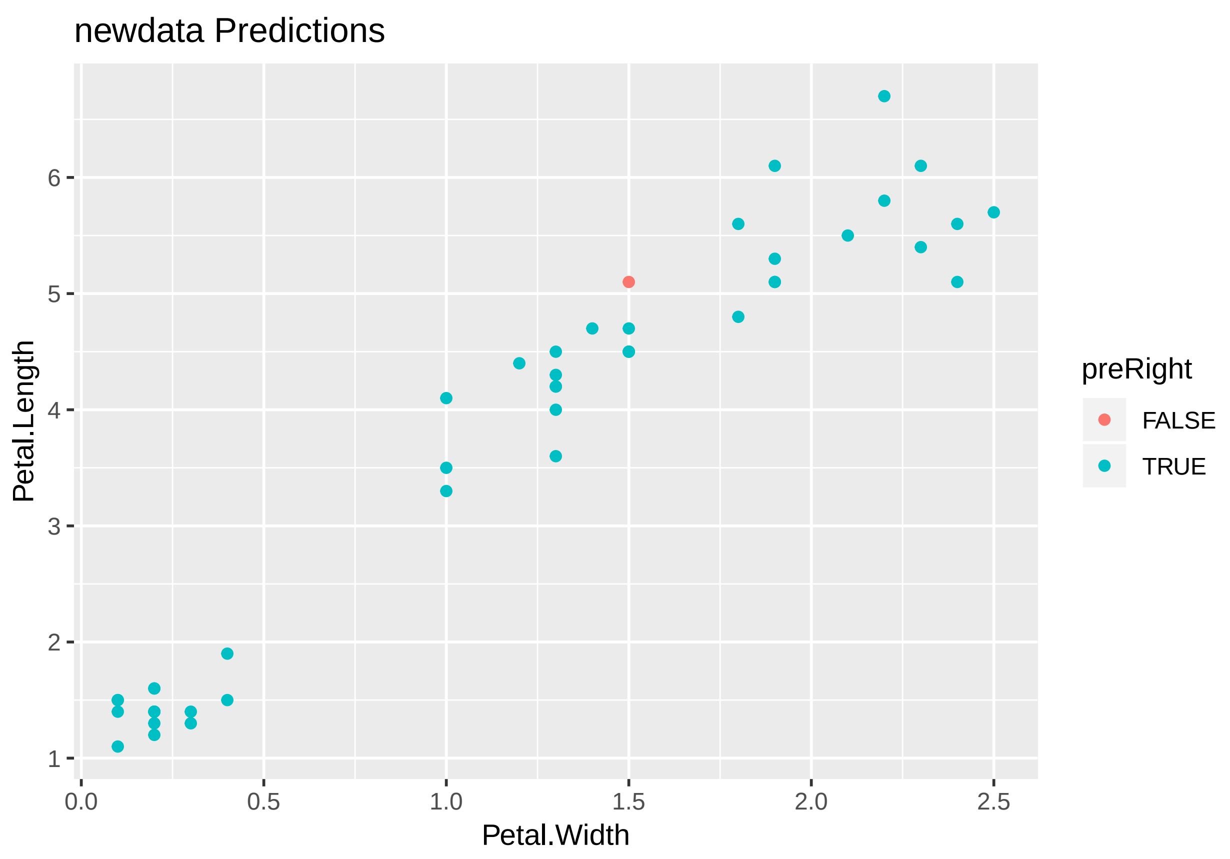


Predicting new values

```
pred<-predict(modFit,testing)
testing$preRight<-pred==testing$Species
table(pred,testing$Species)
```

```
##
## pred         setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         15         1
##   virginica       0          0        14
```

```
qplot(Petal.Width,Petal.Length,col=preRight,data=testing,main="newdata Predictions")
```



Boosting

Boosting and random forest are two of the most accurate out of the box classifiers for prediction analysis.

What is boosting?

1. Take lots of (possibly) weak predictors
2. Weight them and add them up
3. Get a strong predictor

Wage example for boosting

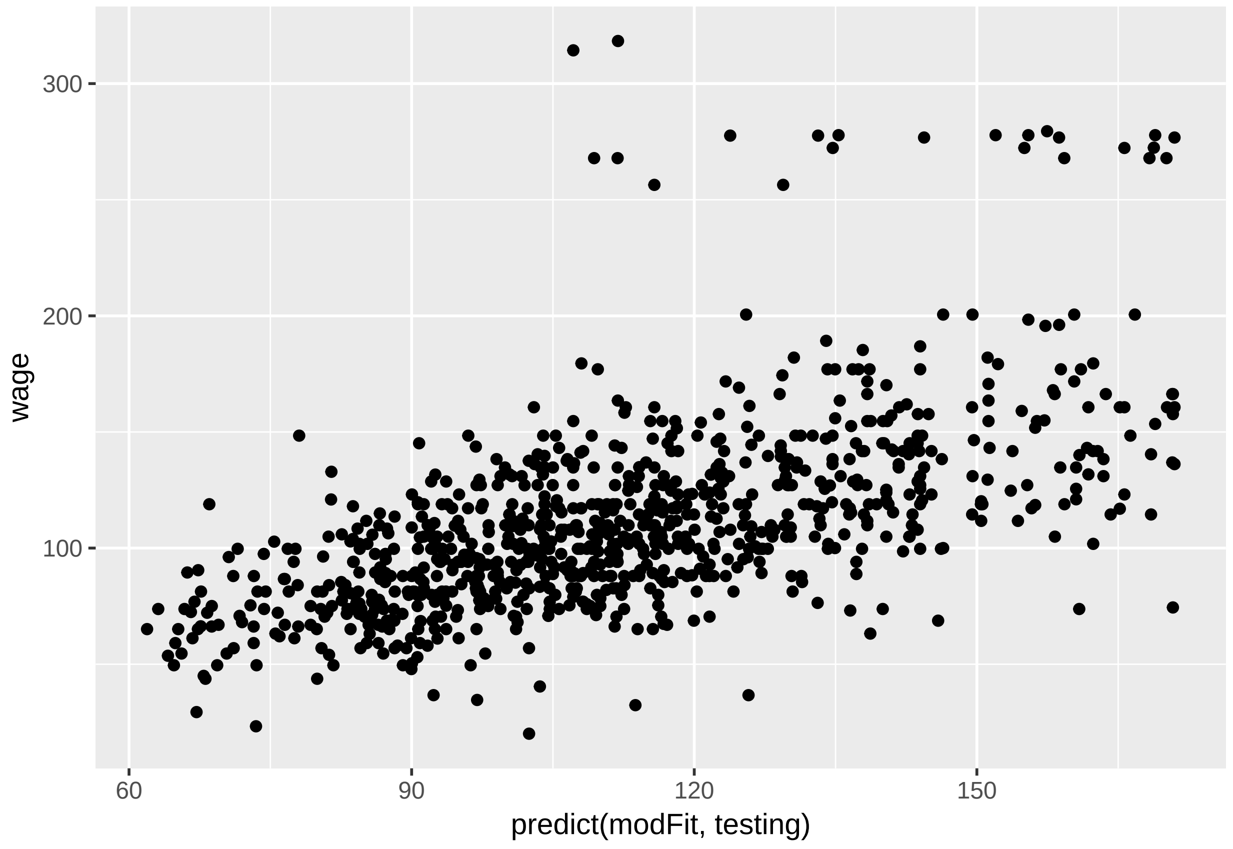
```
library(ISLR)
data(Wage)
library(ggplot2)
library(caret)

Wage<-subset(Wage,select=-c(logwage))
set.seed(1)
inTrain<-createDataPartition(y=Wage$wage,p=0.7,list=FALSE)
training<-Wage[inTrain,]
testing<-Wage[-inTrain,]
```

Fit the boosting model

`gbm` is boosting for tree models.

```r
modFit<-train(wage~.,data=training,method="gbm",verbose=FALSE)
qplot(predict(modFit,testing),wage,data=testing)
```



Model based prediction

What is model based prediction?

1. Assume the data follow a probabilistic model
2. Use Bayes' theorem to identify optimal classifiers

Pros

1. Take advantage of data structures
2. Computationally convenient
3. Reasonably accurate

Cons

1. Make additional assumptions about data

2. When model is incorrect, it may reduce accuracy

Naive Bayes assumes that all features are independent of each other - useful for binary or categorical data, e.g. text classification

Model based prediction with Iris data

```
data(iris)
library(ggplot2)
library(caret)

set.seed(2)
inTrain<-createDataPartition(y=iris$Species,p=0.7,list=FALSE)
training<-iris[inTrain,]
testing<-iris[-inTrain,]
```

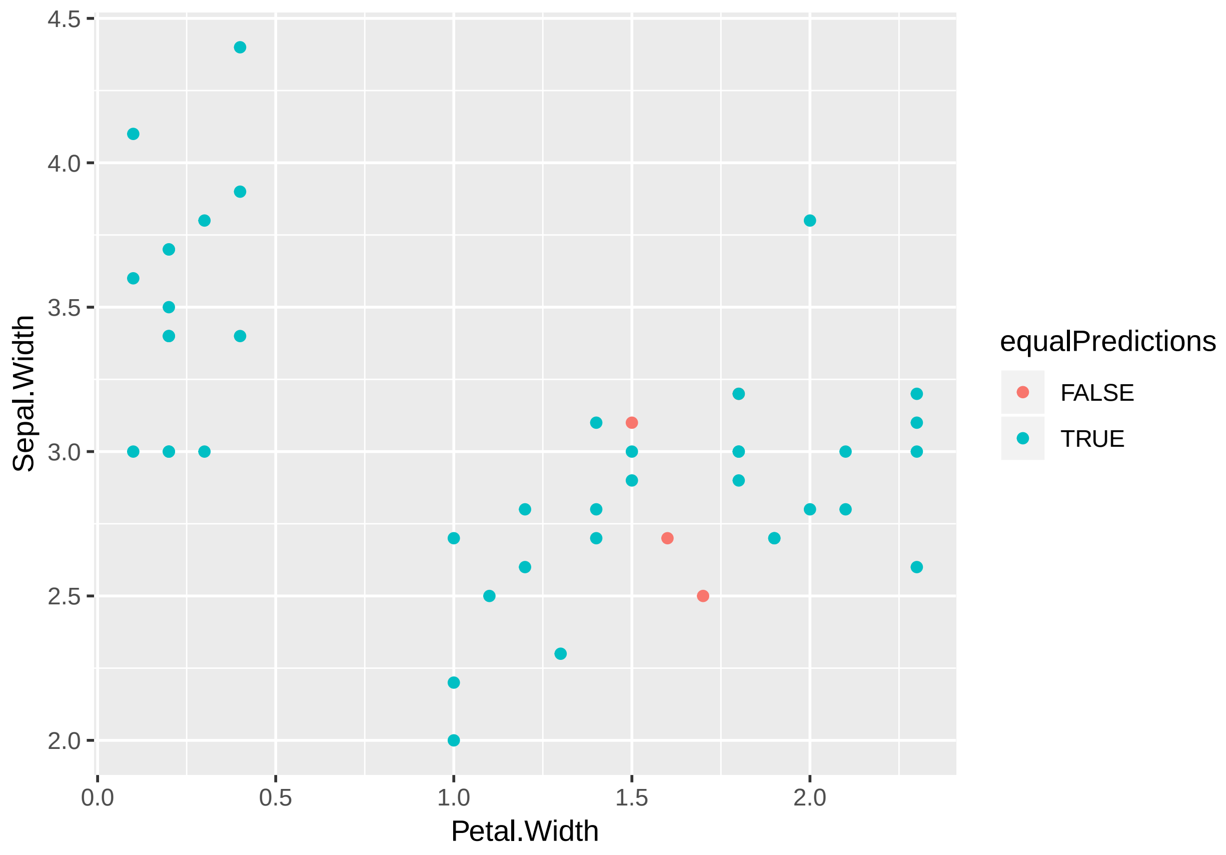
Build predictions

- `lda` = linear discriminant analysis
- `nb` = Naive Bayes

```
modlda<-train(Species~.,data=training,method="lda")
modnb<-train(Species~.,data=training,method="nb")
plda<-predict(modlda,testing)
pnb<-predict(modnb,testing)
table(plda,pnb)
```

```
##               pnb
## plda          setosa versicolor virginica
##   setosa          15          0         0
##   versicolor       0         12         2
##   virginica        0          2        14
```

```
equalPredictions =(plda==pnb)
qplot(Petal.Width,Sepal.Width,col=equalPredictions,data=testing)
```



Sepal.Width
4.5
4.0
3.5
3.0
2.5
2.0
0.0
0.5
1.0
1.5
2.0
Petal.Width
equalPredictions
FALSE
TRUE