Intro to Predictive Modeling

Jason Bryer, Ph.D.

2019-01-22

Agenda

- What is predictive modeling?
  - Training and validation matrices
  - Confusion matrices
  - ROC curves
- Classification and Regression Trees (CART)
- Ensemble methods
- Issues and limitations of predictive modeling

Materials are located here: https://github.com/jbryer/talks

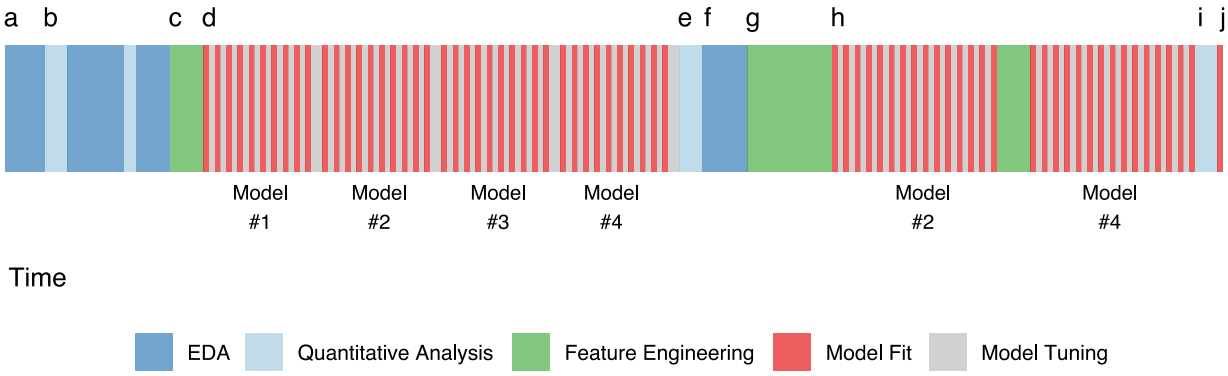
Overview

What is Predictive Modeling?

“Predictive modelling uses statistics to predict outcomes. Most often the event one wants to predict is in the future, but predictive modelling can be applied to any type of unknown event, regardless of when it occurred. For example, predictive models are often used to detect crimes and identify suspects, after the crime has taken place.” (Wikipedia)

Basic Steps of Predictive Modeling[1]

- Estimate model parameters (i.e. training models)
- Determine the values of tuning parameters that cannot be directly calculated from the data
- Model selection
  - Within model type - typically uses *n*-fold validation
  - Between model types - typically uses a validation dataset
- Calculate the performance of the final model that will generalize to new data



1. Adapted from Kuhn's rstudio::conf(2019) conference workshop: https://github.com/topepo/rstudio-conf-2019

Working with Data

Data is typically split into two data sets:

- **Training Set** - the data used to train predictions models (e.g. model estimation, tuning parameters, etc.)
- **Validation Set** - a separate data set used to assess the efficacy of the model. These data *should not* be used during the training phase.

Although there are not established guidelines as to the size of these data sets, randomly selected between 20% and 25% of the data set to "set aside" for validation is typical. The more data we spend on training, the better the model estimates will (often) be.

Stratified Sampling

It is generally advisable to stratify the selection of the training and validation data sets to ensure they are representative of the total sample.

- For **classification** models stratify on the outcome.
- For **regression** models stratify on the quintiles (or similar).

Splitting

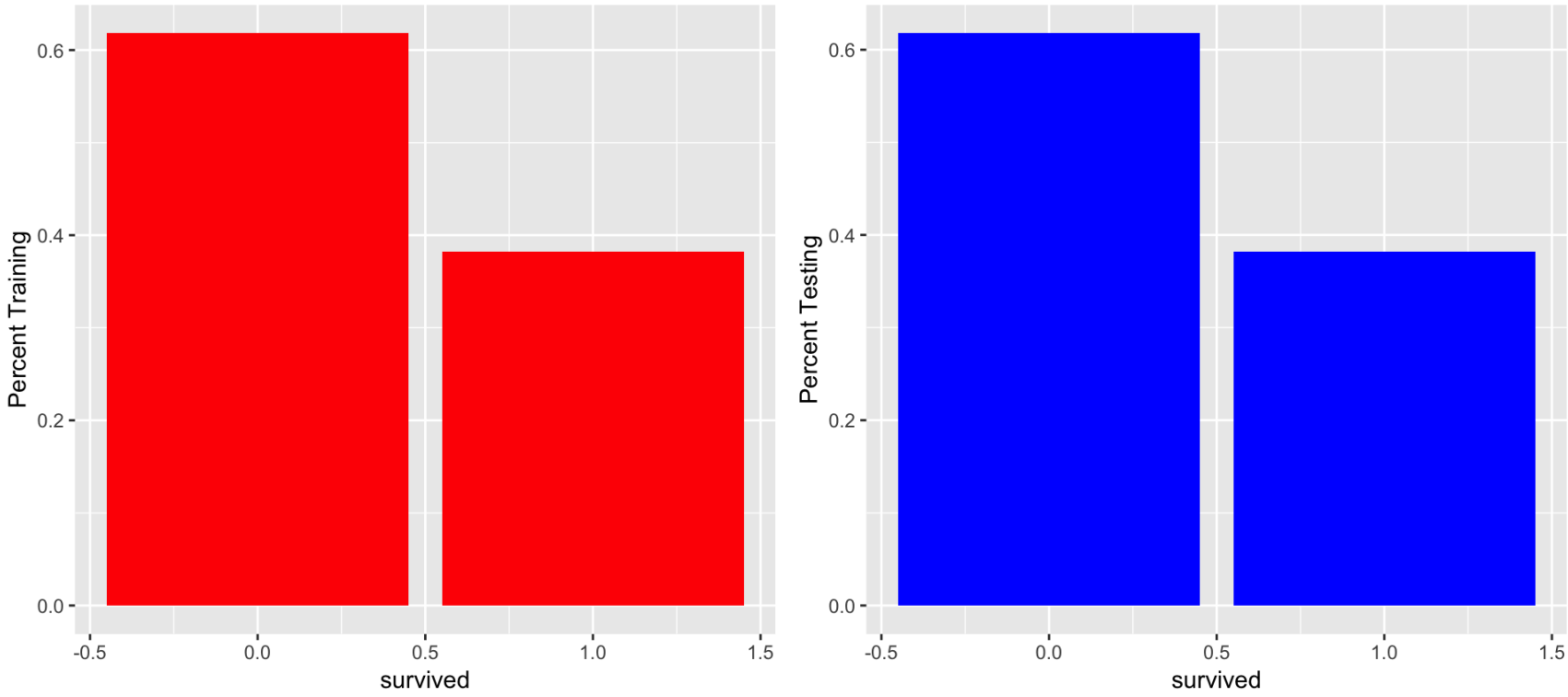
```
set.seed(2112)
titanic <- read.csv('data/titanic3.csv', stringsAsFactors = FALSE)


titanic.split <- rsample::initial_split(titanic, strata = 'survived')
titanic.train <- rsample::training(titanic.split)
titanic.valid <- rsample::testing(titanic.split)

calif <- read.table('data/cadata.dat', header=TRUE)
price.quintiles <- quantile(calif$MedianHouseValue, props = seq(0, 1, 0.2))
calif$cut.prices <- cut(calif$MedianHouseValue, price.quintiles, include.lowest=TRUE)
calif.split <- rsample::initial_split(calif, strata = 'cut.prices')
calif.train <- rsample::training(calif.split)
calif.valid <- rsample::testing(calif.split)
```

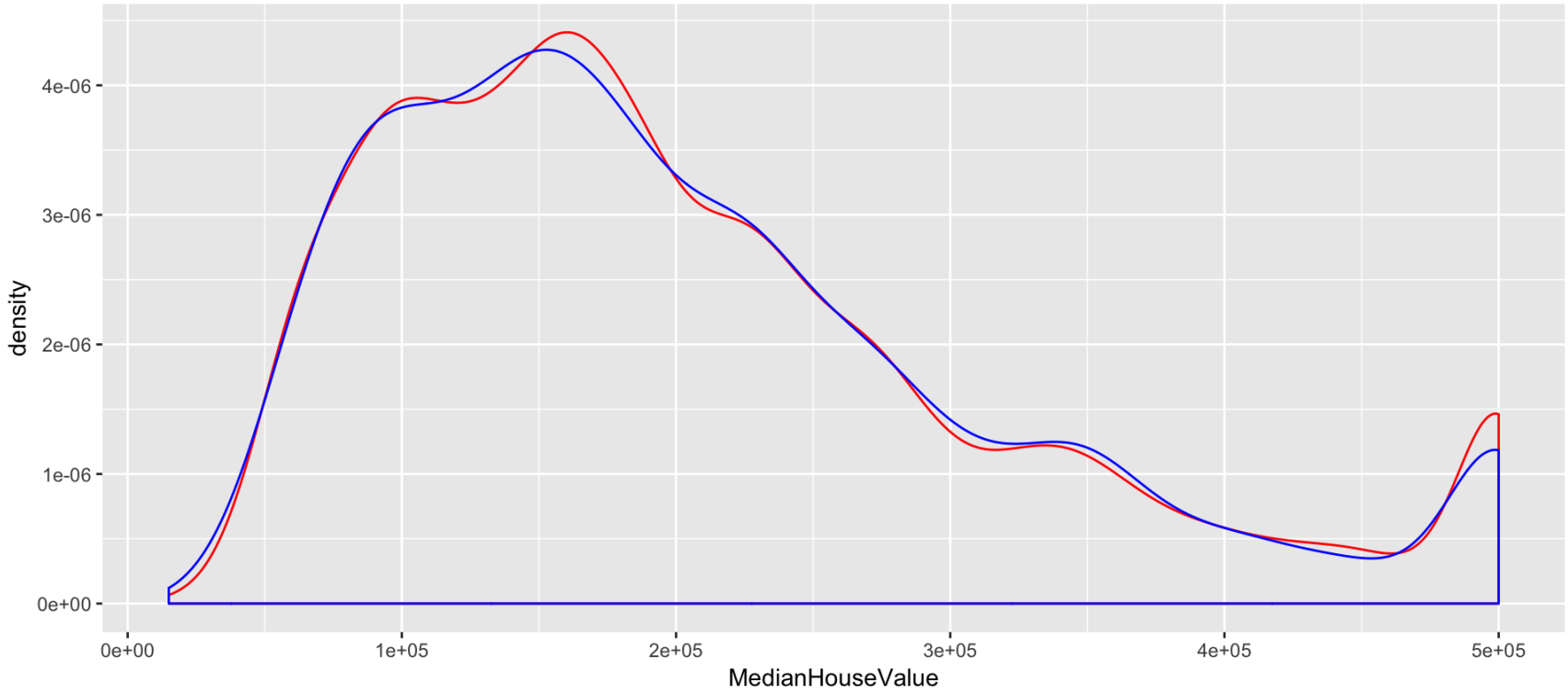
Outcome Distributions: Titanic

```
p1 <- ggplot(titanic.train, aes(x = survived, y = (..count..)/sum(..count..))) +
    geom_bar(fill = 'red') + ylab('Percent Training')
p2 <- ggplot(titanic.valid, aes(x = survived, y = (..count..)/sum(..count..))) +
    geom_bar(fill = 'blue') + ylab('Percent Testing')
p1 + p2 + plot_layout(ncol = 2)
```



Outcome Distributions: California Housing

```
ggplot(calif.train, aes(x = MedianHouseValue)) +
    geom_density(color = 'red') +
    geom_density(data = calif.valid, color = 'blue')
```



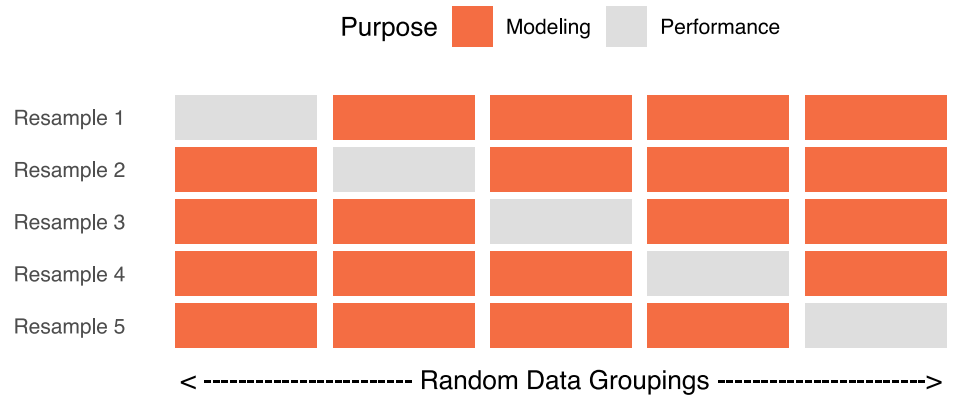
V-Fold Cross Validation

We split the data into V distinct blocks then:

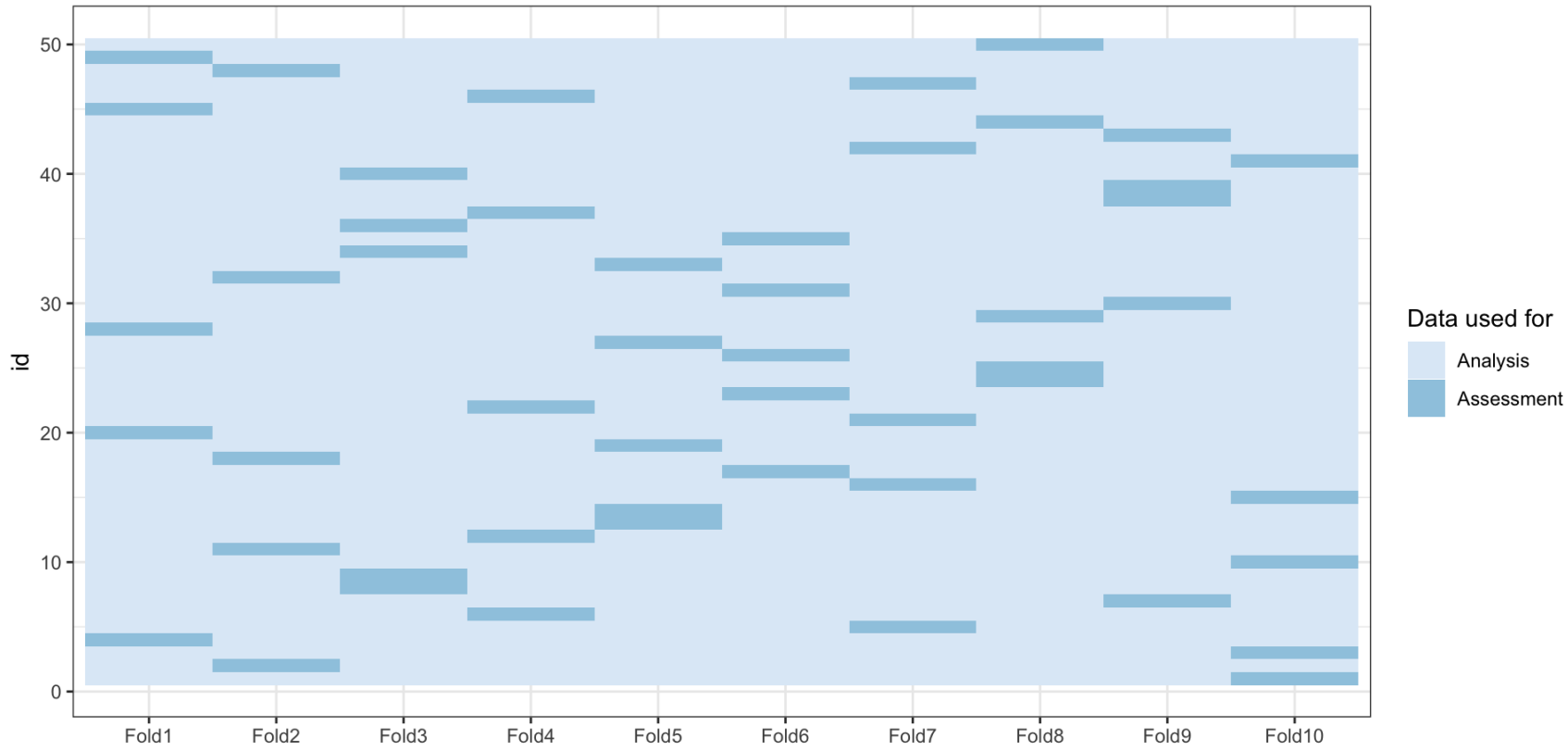
1. Leave the first block out and train with the remaining data.
2. The held-out block is used to evaluate the model.
3. Repeat steps 1 and 2 until all blocks have been used for validation.

We use the average across all models as the final performance for this model.

V is typically 5 or 10 depending on the data size.s



10-Fold Cross-Validation with $n = 50$



Logistic Regression: Titanic

```
titanic.lr <- glm(survived ~ pclass + sex + age + sibsp,
       data=titanic.train,
       family=binomial(logit))
```

Logistic Regression: Titanic

```
summary(titanic.lr)
```

```
##
## Call:
## glm(formula = survived ~ pclass + sex + age + sibsp, family = binomial(logit),
##     data = titanic.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5422  -0.6576  -0.4400   0.6645   2.5051
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.69643    0.44804  10.482  < 2e-16 ***
## pclass      -1.10258    0.11529  -9.564  < 2e-16 ***
## sexmale     -2.59592    0.17684 -14.680  < 2e-16 ***
## age         -0.03376    0.00692  -4.879 1.07e-06 ***
## sibsp       -0.33528    0.09789  -3.425 0.000615 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1306.01  on 981  degrees of freedom
## Residual deviance:  908.07  on 977  degrees of freedom
## AIC: 918.07
##
## Number of Fisher Scoring iterations: 5
```

Evaluating Model Accuracy

```
lr.valid <- predict(titanic.lr,
                    newdata = titanic.valid,
                    type = 'response')
tab <- table(lr.valid > 0.5, titanic.valid$survived) %>% prop.table * 100
tab
```

```
##
##                  0         1
##   FALSE 51.987768 12.844037
##   TRUE   9.785933 25.382263
```

Our total prediction accuracy is 77.4% (reminder that 38.2% of passengers survived).

CART Methods

Classification and Regression Trees

The goal of CART methods is to find best predictor in X of some outcome, y. CART methods do this recursively using the following procedures:

- Find the best predictor in X for y.
- Split the data into two based upon that predictor.
- Repeat 1 and 2 with the split data sets until a stopping criteria has been reached.

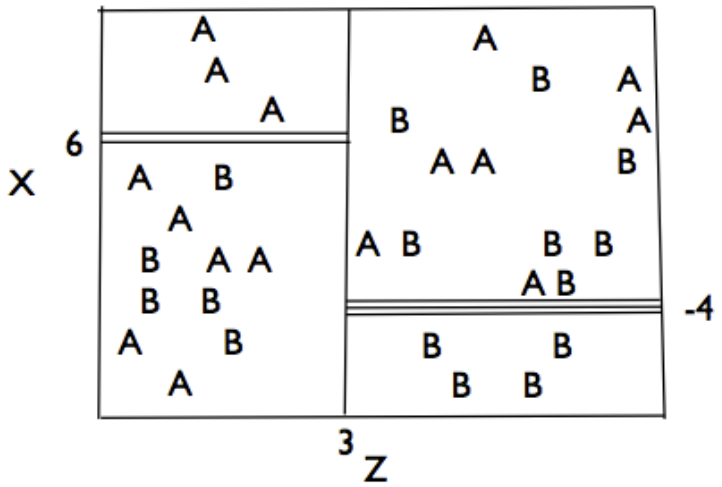
There are a number of possible stopping criteria including: Only one data point remains.

- All data points have the same outcome value.
- No predictor can be found that sufficiently splits the data.

Recursive Partitioning Logic of CART

Consider the scatter plot to the right with the following characteristics:

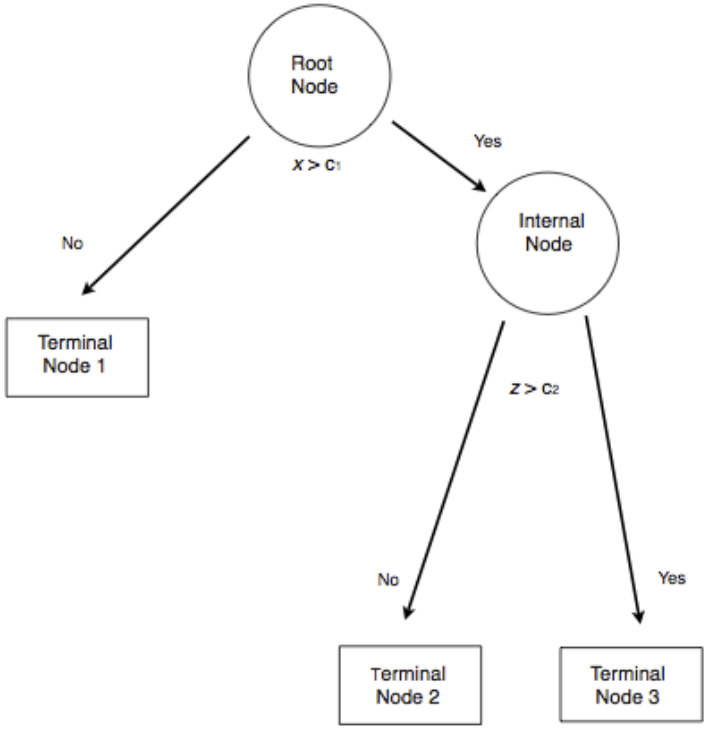
- Binary outcome, G, coded “A” or “B”.
- Two predictors, x and z
- The vertical line at z = 3 creates the first partition.
- The double horizontal line at x = -4 creates the second partition.
- The triple horizontal line at x = 6 creates the third partition.



Recursive Partitioning of a Binary Outcome
(where G = A or B and predictors are Z and X)

Tree Structure

- The root node contains the full data set.
- The data are split into two mutually exclusive pieces. Cases where x > ci go to the right, cases where x <= ci go to the left.
- Those that go to the left reach a terminal node.
- Those on the right are split into two mutually exclusive pieces. Cases where z > c2 go to the right and terminal node 3; cases where z <= c2 go to the left and terminal node 2.



Sum of Squared Errors

The sum of squared errors for a tree *T* is:

$$S = \sum_{c \in leaves(T)} \sum_{i \in c} (y - m_c)^2$$

Where, $m_c = \frac{1}{n} \sum_{i \in c} y_i$, the prediction for leaf .

Or, alternatively written as:

$$S = \sum_{c \in leaves(T)} n_c V_c$$

Where V_c is the within-leave variance of leaf .

Our goal then is to find splits that minimize S.

Advantages of CART Methods

- Making predictions is fast.
- It is easy to understand what variables are important in making predictions.
- Trees can be grown with data containing missingness. For rows where we cannot reach a leaf node, we can still make a prediction by averaging the leaves in the sub-tree we do reach.
- The resulting model will inherently include interaction effects. There are many reliable algorithms available.

Regression Trees

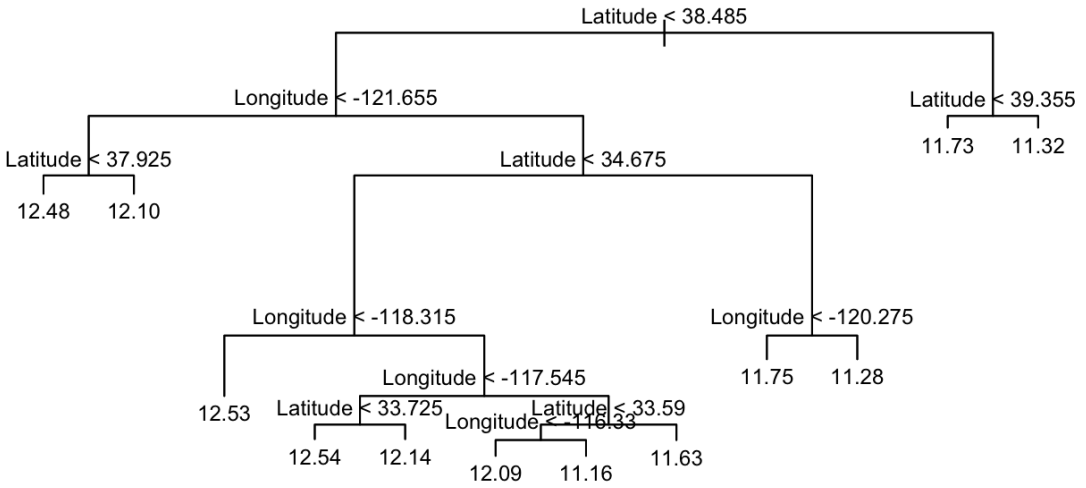
In this example we will predict the median California house price from the house's longitude and latitude.

```
str(calif)
```

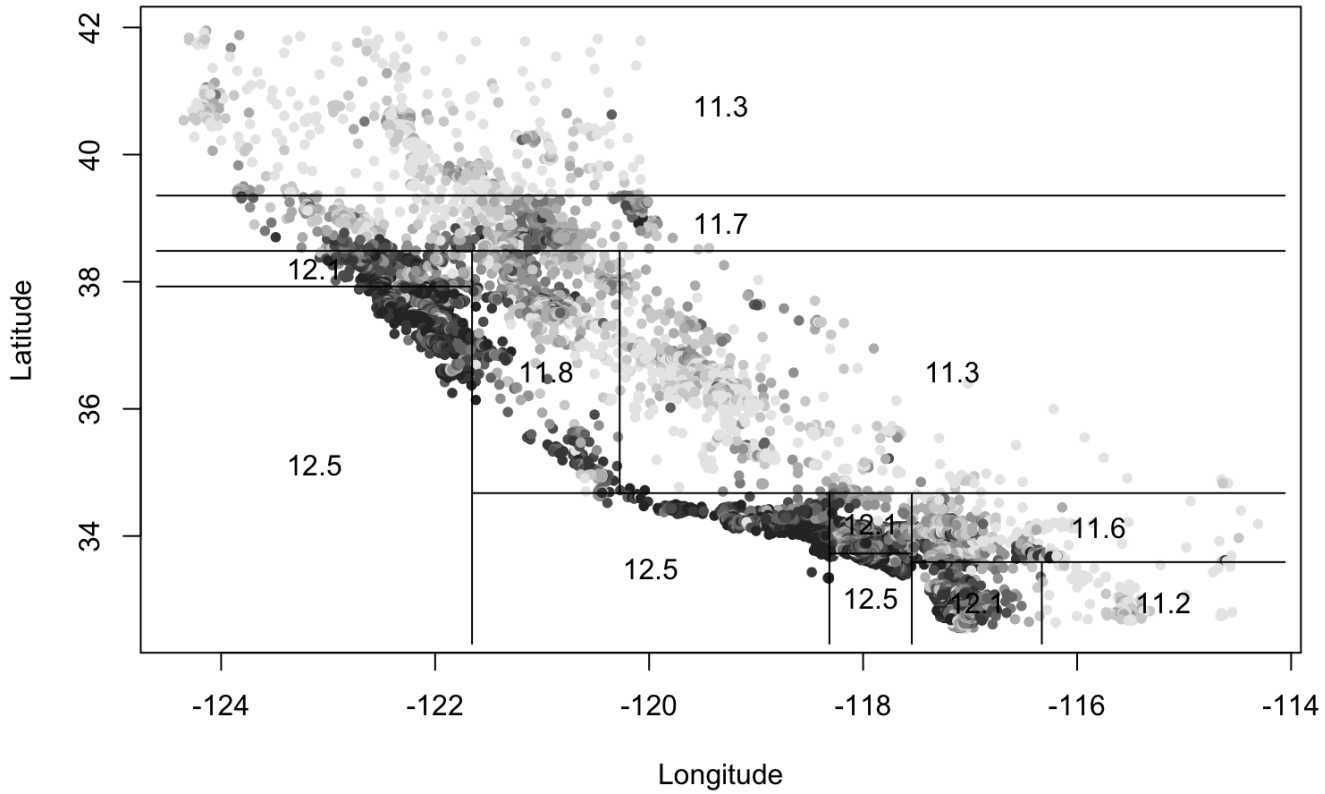
```
## 'data.frame':    20640 obs. of  10 variables:
##  $ MedianHouseValue: num  452600 358500 352100 341300 342200 ...
##  $ MedianIncome    : num  8.33 8.3 7.26 5.64 3.85 ...
##  $ MedianHouseAge  : num  41 21 52 52 52 52 52 52 42 52 ...
##  $ TotalRooms      : num  880 7099 1467 1274 1627 ...
##  $ TotalBedrooms   : num  129 1106 190 235 280 ...
##  $ Population      : num  322 2401 496 558 565 ...
##  $ Households      : num  126 1138 177 219 259 ...
##  $ Latitude        : num  37.9 37.9 37.9 37.9 37.9 ...
##  $ Longitude       : num  -122 -122 -122 -122 -122 ...
##  $ cut.prices      : Factor w/ 4 levels "[1.5e+04,1.2e+05]",..: 4 4 4 4 4 4 4 3 3 3 ...
```

Tree 1

```
treefit <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data=calif)
plot(treefit); text(treefit, cex=0.75)
```



Tree 1


Latitude
42
40
38
36
34
11.3
11.7
12.1
11.8
11.3
12.5
12.1
11.6
12.5
12.5
12.1
11.2
-124
-122
-120
-118
-116
-114
Longitude

Tree 1

```
summary(treefit)
```

```
##
## Regression tree:
## tree(formula = log(MedianHouseValue) ~ Longitude + Latitude,
##     data = calif)
## Number of terminal nodes:  12
## Residual mean deviance:  0.1662 = 3429 / 20630
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.75900 -0.26080 -0.01359  0.00000  0.26310  1.84100
```

Here “deviance” is the mean squared error, or root-mean-square error of $\sqrt{.166} = 0.41$.

Tree 2, Reduce Minimum Deviance

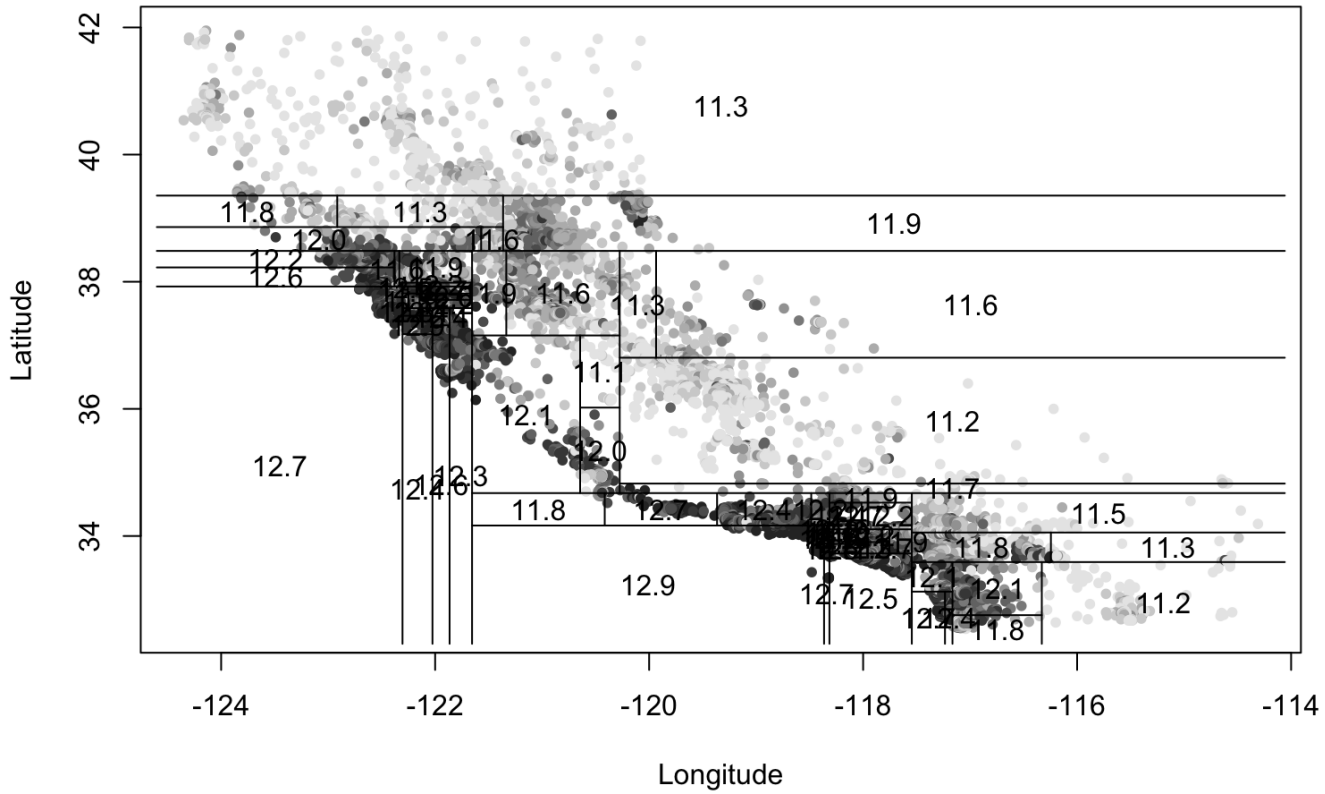
We can increase the fit but changing the stopping criteria with the mindev parameter.

```
treefit2 <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data=calif, mindev=.001)
summary(treefit2)
```

```
##
## Regression tree:
## tree(formula = log(MedianHouseValue) ~ Longitude + Latitude,
##     data = calif, mindev = 0.001)
## Number of terminal nodes:  68
## Residual mean deviance:  0.1052 = 2164 / 20570
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.94700 -0.19790 -0.01872  0.00000  0.19970  1.60600
```

With the larger tree we now have a root-mean-square error of 0.32.

Tree 2, Reduce Minimum Deviance



Tree 3, Include All Variables

However, we can get a better fitting model by including the other variables.

```
treefit3 <- tree(log(MedianHouseValue) ~ ., data=calif)
summary(treefit3)
```

```
##
## Regression tree:
## tree(formula = log(MedianHouseValue) ~ ., data = calif)
## Variables actually used in tree construction:
## [1] "cut.prices"
## Number of terminal nodes:  4
## Residual mean deviance:  0.03608 = 744.5 / 20640
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.718000 -0.127300  0.009245  0.000000  0.130000  0.358600
```

With all the available variables, the root-mean-square error is 0.11.

Classification Trees

- `pclass`: Passenger class (1 = 1st; 2 = 2nd; 3 = 3rd)
- `survival`: A Boolean indicating whether the passenger survived or not (0 = No; 1 = Yes); this is our target
- `name`: A field rich in information as it contains title and family names
- `sex`: male/female
- `age`: Age, a significant portion of values are missing
- `sibsp`: Number of siblings/spouses aboard
- `parch`: Number of parents/children aboard
- `ticket`: Ticket number.
- `fare`: Passenger fare (British Pound).
- `cabin`: Does the location of the cabin influence chances of survival?
- `embarked`: Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
- `boat`: Lifeboat, many missing values
- `body`: Body Identification Number
- `home.dest`: Home/destination

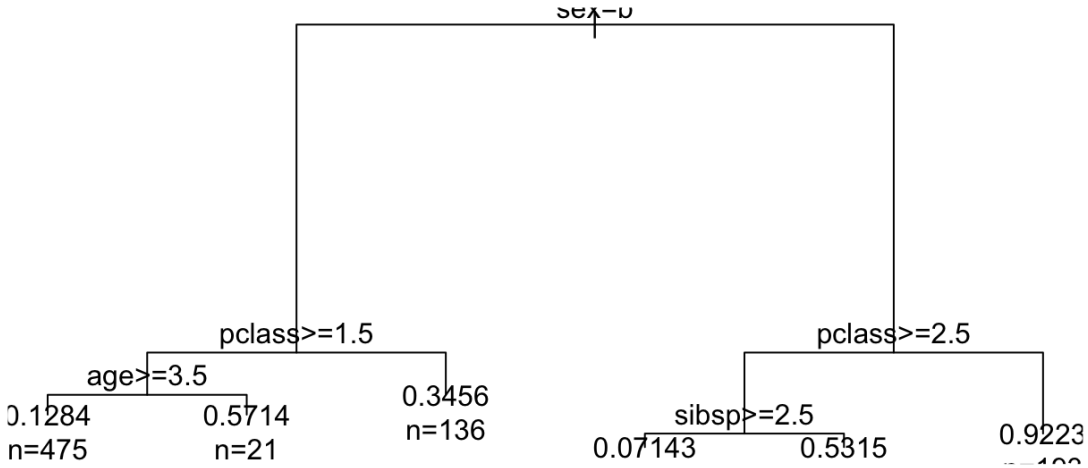
Classification using `rpart`

```
(titanic.rpart <- rpart(survived ~ pclass + sex + age + sibsp,
   data=titanic.train))
```

```
## n= 982
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 982 231.7974000 0.38187370
##    2) sex=male 632  97.2151900 0.18987340
##      4) pclass>=1.5 496  62.2560500 0.14717740
##        8) age>=3.5 475  53.1663200 0.12842110 *
##        9) age< 3.5 21   5.1428570 0.57142860 *
##      5) pclass< 1.5 136  30.7573500 0.34558820 *
##    3) sex=female 350  69.2142900 0.72857140
##      6) pclass>=2.5 157  39.2356700 0.49044590
##       12) sibsp>=2.5 14   0.9285714 0.07142857 *
##       13) sibsp< 2.5 143  35.6083900 0.53146850 *
##      7) pclass< 2.5 193  13.8342000 0.92227980 *
```

Classification using **rpart**

```
plot(titanic.rpart); text(titanic.rpart, use.n=TRUE, cex=1)
```



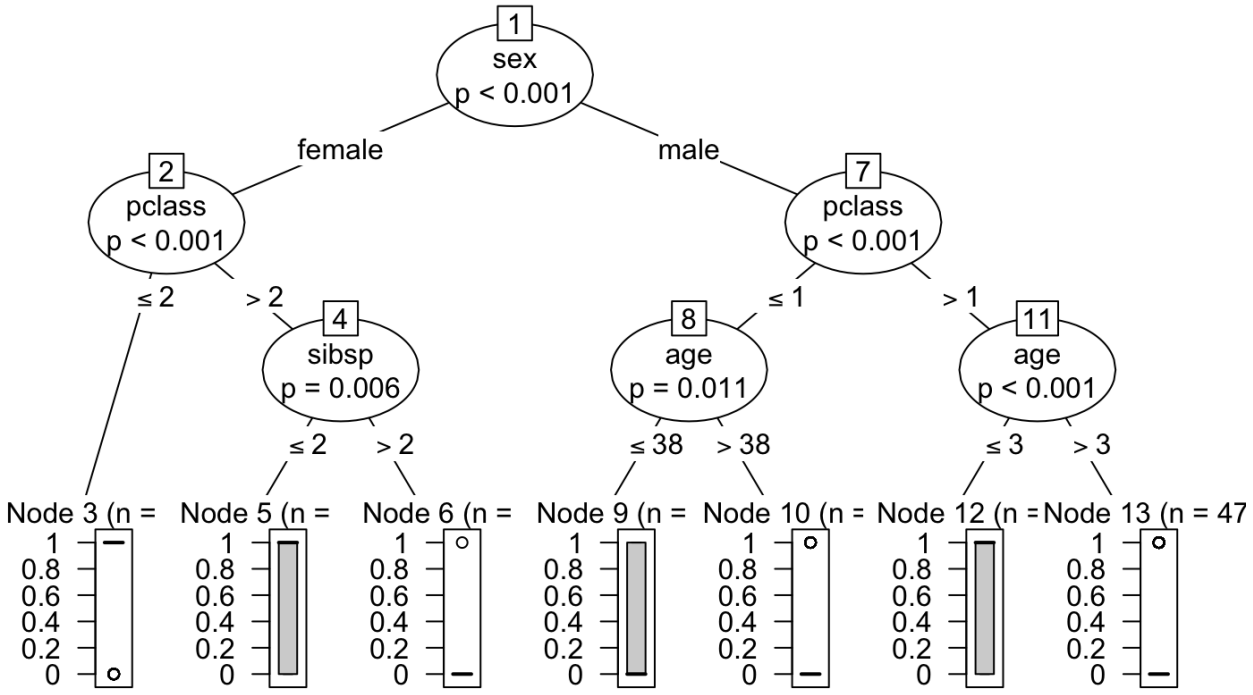
Classification using `ctree`

```
(titanic.ctree <- ctree(survived ~ pclass + sex + age + sibsp, data=titanic.train))
```

```
##
##   Conditional inference tree with 7 terminal nodes
##
## Response:  survived
## Inputs:  pclass, sex, age, sibsp
## Number of observations:  982
##
## 1) sex == {female}; criterion = 1, statistic = 276.646
##   2) pclass <= 2; criterion = 1, statistic = 74.935
##     3)*  weights = 193
##   2) pclass > 2
##     4) sibsp <= 2; criterion = 0.994, statistic = 10.228
##       5)*  weights = 143
##     4) sibsp > 2
##       6)*  weights = 14
## 1) sex == {male}
##   7) pclass <= 1; criterion = 1, statistic = 19.547
##     8) age <= 38; criterion = 0.989, statistic = 8.919
##       9)*  weights = 60
##     8) age > 38
##       10)*  weights = 76
##   7) pclass > 1
##     11) age <= 3; criterion = 1, statistic = 15.735
##       12)*  weights = 21
##     11) age > 3
##       13)*  weights = 475
```

Classification using `ctree`

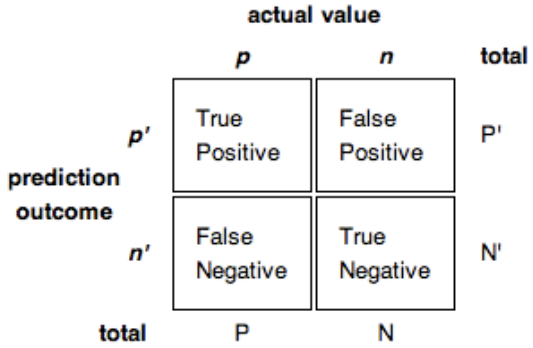
```
plot(titanic.ctree)
```

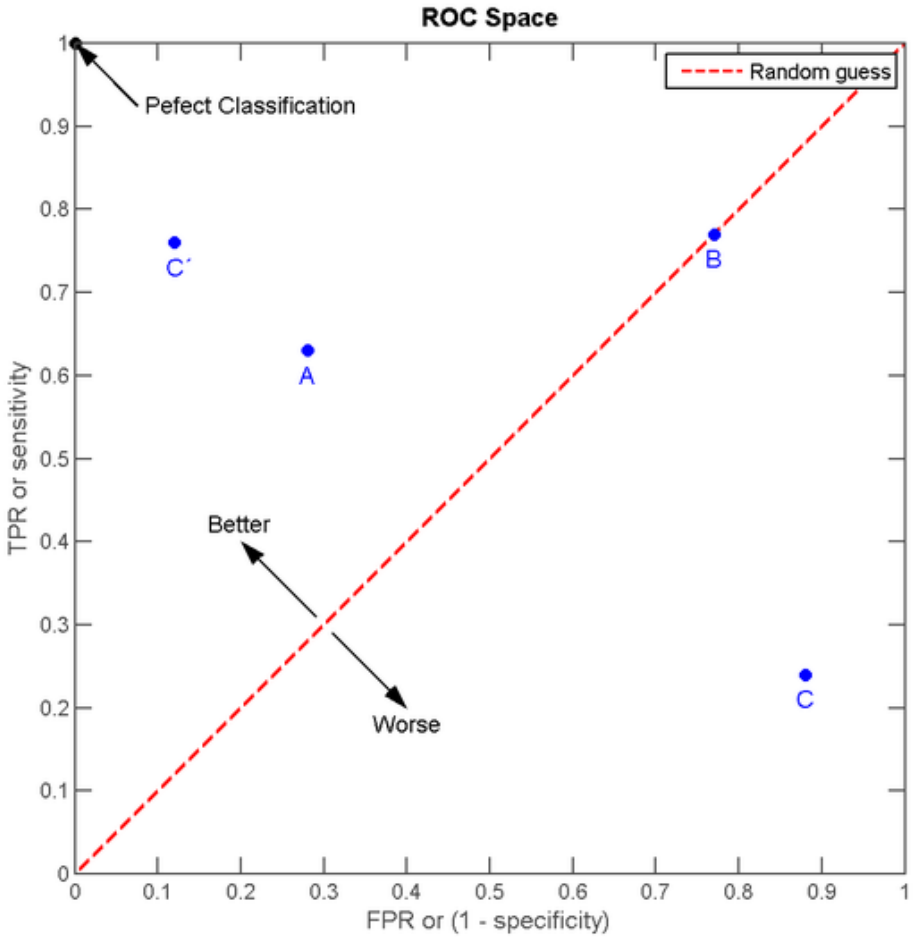


Receiver Operating Characteristic (ROC) Graphs

In a classification model, outcomes are either as positive (*p*) or negative (*n*). There are then four possible outcomes:

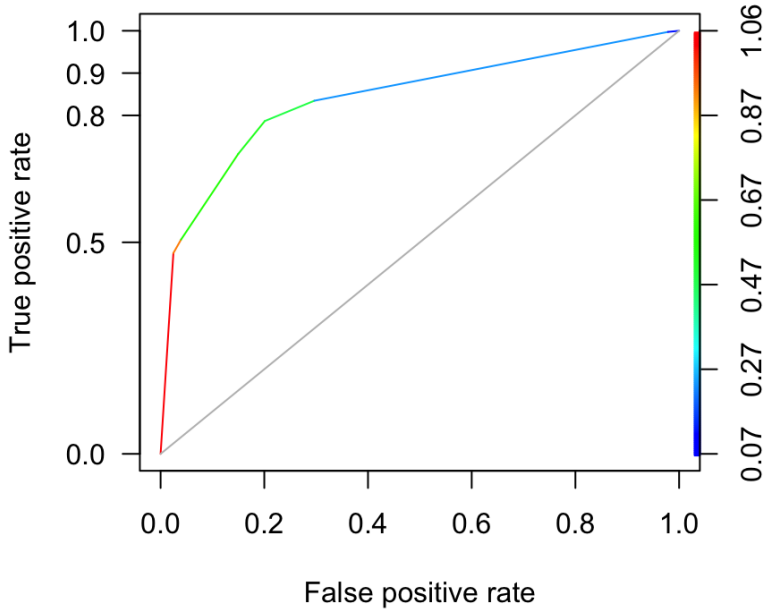
- **true positive** (TP) The outcome from a prediction is *p* and the actual value is also *p*.
- **false positive** (FP) The actual value is *n*.
- **true negative** (TN) Both the prediction outcome and the actual value are *n*.
- **false negative** (FN) The prediction outcome is *n* while the actual value is *p*.




ROC Space
Random guess
Pefect Classification
C´
B
A
Better
Worse
C
TPR or sensitivity
FPR or (1 - specificity)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

ROCR Package

```
titanic.pred <- predict(titanic.ctree)
pred <- prediction(titanic.pred,
            as.integer(titanic.train$survived))
perf <- performance(pred,
            measure="tpr",
            x.measure="fpr")
plot(perf, colorize=TRUE,
            yaxis.at=c(0,0.5,0.8,0.9,1),
            yaxis.las=1)
lines(c(0,1), c(0,1), col="grey")
```



Ensemble Methods

Ensemble Methods

Ensemble methods use multiple models that are combined by weighting, or averaging, each individual model to provide an overall estimate. Each model is a random sample of the sample. Common ensemble methods include:

- *Boosting* - Each successive trees give extra weight to points incorrectly predicted by earlier trees. After all trees have been estimated, the prediction is determined by a weighted "vote" of all predictions (i.e. results of each individual tree model).
- *Bagging* - Each tree is estimated independent of other trees. A simple "majority vote" is take for the prediction.
- *Random Forests* - In addition to randomly sampling the data for each model, each split is selected from a random subset of all predictors.
- *Super Learner* - An ensemble of ensembles. See https://cran.r-project.org/web/packages/SuperLearner/vignettes/Guide-to-SuperLearner.html

Random Forests

The random forest algorithm works as follows:

1 Draw n_{tree} bootstrap samples from the original data.

2 For each bootstrap sample, grow an unpruned tree. At each node, randomly sample m_{try} predictors and choose the best split among those predictors selected[2].

3 Predict new data by aggregating the predictions of the ntree trees (majority votes for classification, average for regression).

Error rates are obtained as follows:

1 At each bootstrap iteration predict data not in the bootstrap sample (what Breiman calls “out-of-bag”, or OOB, data) using the tree grown with the bootstrap sample.

2 Aggregate the OOB predictions. On average, each data point would be out-of-bag 36% of the times, so aggregate these predictions. The calculated error rate is called the OOB estimate of the error rate.

2. Bagging is a special case of random forests where $m_{try} = p$ where p is the number of predictors

Random Forests: Titanic

```
titanic.rf <- randomForest(factor(survived) ~ pclass + sex + age + sibsp,
                           data = titanic.train,
                           ntree = 5000,
                           importance = TRUE)
```

```
print(titanic.rf)
```

```
##
## Call:
##  randomForest(formula = factor(survived) ~ pclass + sex + age +      sibsp, data = titanic.train, ntree = 5000, i
##                Type of random forest: classification
##                      Number of trees: 5000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 21.08%
## Confusion matrix:
##     0   1 class.error
## 0 538  69   0.1136738
## 1 138 237   0.3680000
```

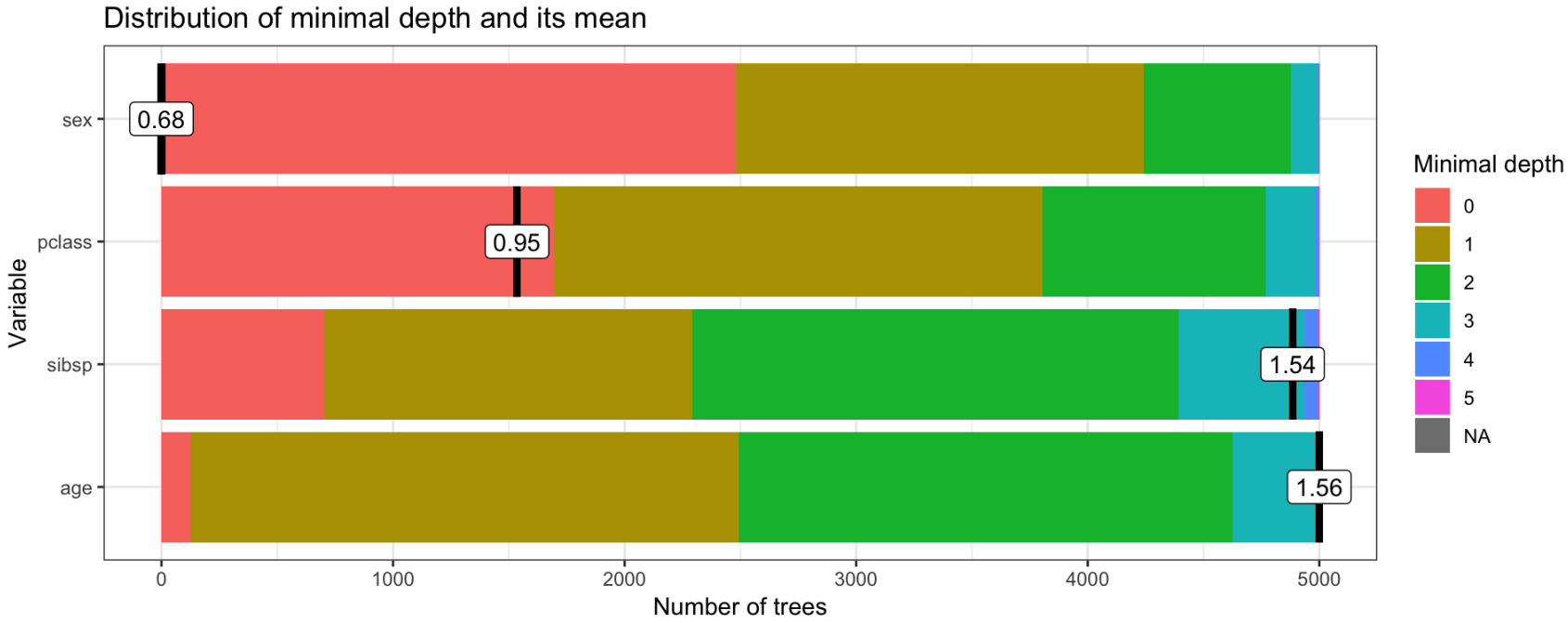
```
importance(titanic.rf)
```

```
##                0         1 MeanDecreaseAccuracy MeanDecreaseGini
## pclass  78.12523 103.87515            116.38536         45.48943
## sex    259.56409 307.90063            322.85158        125.63139
## age     61.39519  42.38810             80.93138         56.56754
## sibsp   49.74559  11.65141             47.21834         16.72281
```

Random Forests: Titanic

```
min_depth_frame <- min_depth_distribution(titanic.rf)
```

```
plot_min_depth_distribution(min_depth_frame)
```



caret Package

The `caret` Package

```
The caret package (short for _C_lassification _A_nd _RE_gression _T_raining)
is a set of functions that attempt to streamline the process for creating
predictive models.
```

The `caret` package creates a unified interface for using many modeling packages.

Function	Package	Code
lda	MASS	predict(obj)
glm	stats	predict(obj, type = “response”)
gbm	gbm	predict(obj, type = “response”, n.trees)
mda	mda	predict(obj, type = “posterior”)
rpart	rpart	predict(obj, type = “prob”)
Weka	RWeka	predict(obj, type = “probability”)
logitboost	LogitBoost	predict(obj, type = “raw”, nIter)

Tutoring Example: Data Preparation

```
data("tutoring")
tutoring$treat2 <- as.factor(tutoring$treat != 'Control')

inTrain <- createDataPartition(
    y = tutoring$treat2,
    ## the outcome data are needed
    p = .75,
    ## The percentage of data in the
    ## training set
    list = FALSE
)

tutoring.train <- tutoring[inTrain,]
tutoring.valid <- tutoring[-inTrain,]
```

Tutoring Example: Training with Random Forests

```
rfFit <- train(
    treat2 ~ Gender + Ethnicity + Military + ESL + EdMother + EdFather +
        Age + Employment + Income + Transfer + GPA,
    data = tutoring.train,
    method = "parRF"
)
rfFit
```

```
## Parallel Random Forest
##
## 857 samples
##  11 predictor
##   2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 857, 857, 857, 857, 857, 857, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8058582  0.1224764
##    7    0.8002241  0.2133111
##   12    0.7925550  0.2035628
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Tutoring Example: Validation of Random Forests

```
rf.valid.pred <- predict(rfFit, newdata = tutoring.valid)
confusionMatrix(rf.valid.pred, tutoring.valid$treat2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   225   51
##      TRUE      4    5
##
##                Accuracy : 0.807
##                  95% CI : (0.7563, 0.8512)
##     No Information Rate : 0.8035
##     P-Value [Acc > NIR] : 0.4763
##
##                   Kappa : 0.1052
##  Mcnemar's Test P-Value : 5.552e-10
##
##             Sensitivity : 0.98253
##             Specificity : 0.08929
##          Pos Pred Value : 0.81522
##          Neg Pred Value : 0.55556
##              Prevalence : 0.80351
##          Detection Rate : 0.78947
##    Detection Prevalence : 0.96842
##       Balanced Accuracy : 0.53591
##
##        'Positive' Class : FALSE
##
```

Tutoring Example: Training with Naive Bayes

```
bayesFit <- train(
    treat2 ~ Gender + Ethnicity + Military + ESL + EdMother + EdFather +
        Age + Employment + Income + Transfer + GPA,
    data = tutoring.train,
    method = "naive_bayes"
)
```

Tutoring Example: Validation of Naive Bayes

```
bayes.valid.pred <- predict(bayesFit, newdata = tutoring.valid)
confusionMatrix(bayes.valid.pred, tutoring.valid$treat2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   229   56
##      TRUE      0    0
##
##                Accuracy : 0.8035
##                  95% CI : (0.7526, 0.848)
##     No Information Rate : 0.8035
##     P-Value [Acc > NIR] : 0.5357
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 1.987e-13
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.8035
##          Neg Pred Value :    NaN
##              Prevalence : 0.8035
##          Detection Rate : 0.8035
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : FALSE
##
```

Discussion

Considerations for Predictive Modeling

Feature Engineering

- What variables should be transformed (e.g. recoded, log transformed, elimination of predictors, centered, scaled, etc.)?
- What to do with missing data (e.g. impute)?
- Would conducting a principal component analysis help?
- Are there interaction effects between some variables?

Are your predictions actually any good?

Loh, Soo, and Xing (2016) were able to predict one's sexual orientation with approximately 90% accuracy. However, according to Gallop, only 4.1% of American identify as LGTBQ. If I were to guess every American was heterosexual I would be correct 95.9% of the time, doing better than the Loh, Soo, and Xing's predictive models.

Are your predictions reinforcing social biases?

- When it Comes to Gorillas, Google Photos Remains Blind (Wired)
- How Amazon Accidentally Invented a Sexist Hiring Algorithm (Inc.)

Additional Resources

Kuhn and Johnson's *Applied Predictive Modeling*

- Website: http://appliedpredictivemodeling.com/
- Workshop Materials: https://github.com/topepo/rstudio-conf-2019
- `caret` package: http://topepo.github.io/caret

`randomForestExplainer`:
https://cran.rstudio.com/web/packages/randomForestExplainer/vignettes/randomForestExplainer.html

`SuperLearner` package: https://github.com/ecpolley/SuperLearner

Ferndández-Delgado, Cernadas, Barro, & Amorin (2014). Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research, 15*, 3133-3181

Thank You!

Jason Bryer, Ph.D.

Email: jason@bryer.org
Twitter: [@jbryer](https://twitter.com/@jbryer)
Website: https://www.bryer.org
Github: https://github.com/jbryer

Albany R Users' Group: https://www.meetup.com/Albany-R-Users-Group/