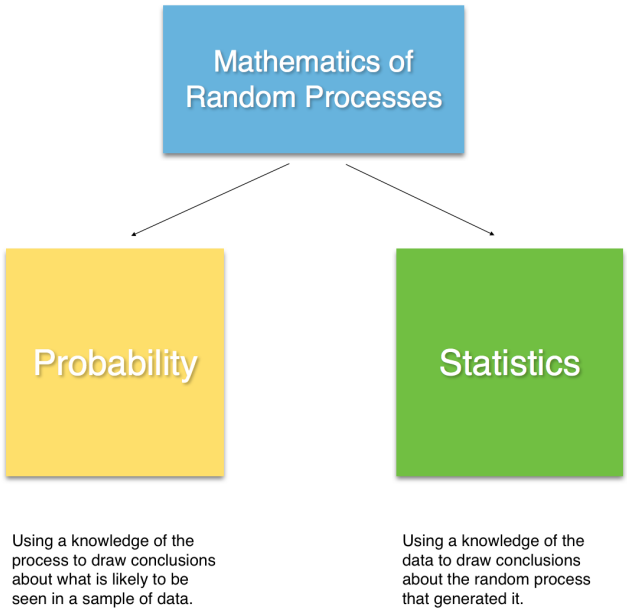
Lecture 6: A Grand Tour of Machine Learning

The Big Picture



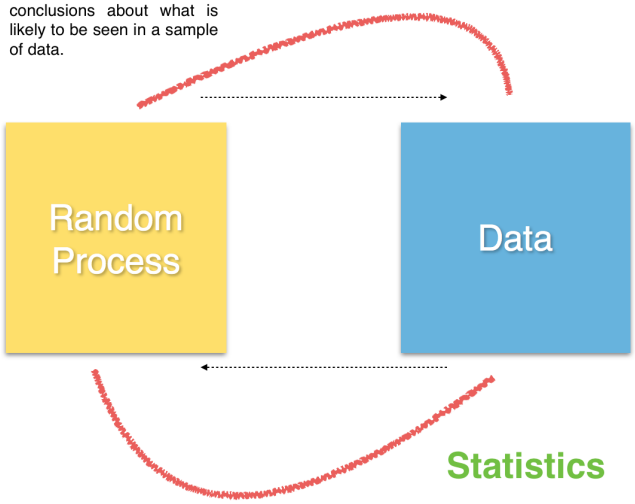
Using a knowledge of the process to draw conclusions about what is likely to be seen in a sample of data.

Using a knowledge of the data to draw conclusions about the random process that generated it.

Complementary Fields

Probability

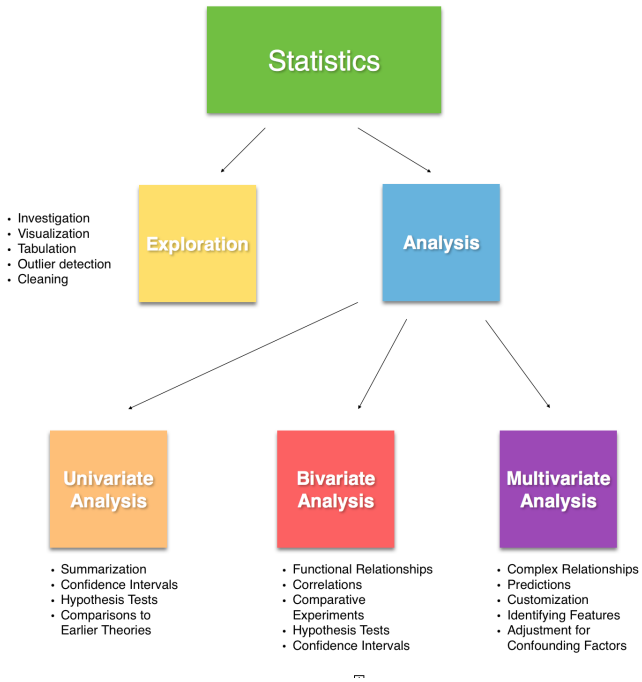
Utilizing a knowledge of the random process to draw conclusions about what is likely to be seen in a sample of data.



Statistics

Utilizing a knowledge of the data to draw conclusions about the random process that generated it.

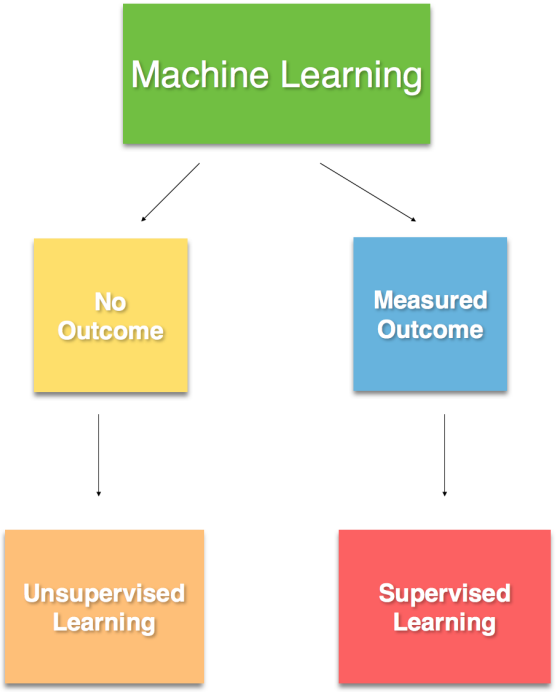
Statistics in All of Its Forms



Machine Learning is a Multivariate Statistical Analysis

- Any form of investigation that concerns the relationship of multiple variables based on measured data is potentially relevant.
- Machine Learning's techniques can serve a variety of purposes.
- Although the range is complex, the methods and aims can best be understood by exploring the branches of the tree we have been building.

Categories of Machine Learning



A Textbook

- James et al. **An Introduction to Statistical Learning**: https://www-bcf.usc.edu/~gareth/ISL/
- Provides an overview of many topics in Machine Learning
- Relative to many other works, this texts provides a good mix of the intuitions of many methods and some degree of technical detail. It is an approachable introduction to these topics for practitioners.
- As part of our discussion, I will be using some examples from the text.

Exploring Machine Learning with the AirBnB Data

```
library(data.table)
dat <- fread(input = "AirBnB analysisData.csv", sep = ",",
    fill = TRUE)
dim(dat)
```

```
[1] 29142    96
```

Some Data Cleaning Steps

```
num.bedrooms.name <- "bedrooms"
num.bathrooms.name <- "bathrooms"
lat.name <- "latitude"
long.name <- "longitude"
cozy.name <- "cozy"
description.name <- "description"
the.variables <- c(num.bedrooms.name, num.bathrooms.name, lat.name, long.name)

dat[, eval(cozy.name) := 0]
dat[grep(pattern = cozy.name, x = tolower(get(description.name))), eval(cozy.name) := 1]
dat[, c(eval(num.bedrooms.name), eval(num.bathrooms.name), eval(lat.name), eval(long.name)) := lapply(X = .SD, FUN =
  "as.numeric"), .SDcols = the.variables]
```

Notice how **multiple columns can be assigned** in a single step with this use of lapply and .SDcols.

Unsupervised Learning

- Aside from more standard explorations, the most common form of unsupervised learning is **clustering**.
- **Clustering** is an automated means of placing subjects into groups based upon similarities in their measured characteristics.
- But that raises the question: how do you decide how similar two subjects are when they have many characteristics?

Distance Functions

- A **distance function** is a non-negative value that shows how far apart two points are.
- We are already familiar with certain kinds of distance functions like the standard deviation. This is based upon **Euclidean distance**.
- In this setting, we'd have to handle categorical variables, e.g. with additional binary columns for each value relative to a reference category.

Different Methods of Clustering

- **K-Means**: Choose the best **K** points as the center of different clusters, grouping each point based on the closest center.
- **Hierarchical Clustering**: Iteratively build larger and larger groups of points by merging the two closest groups at each step.

K-Means Clustering

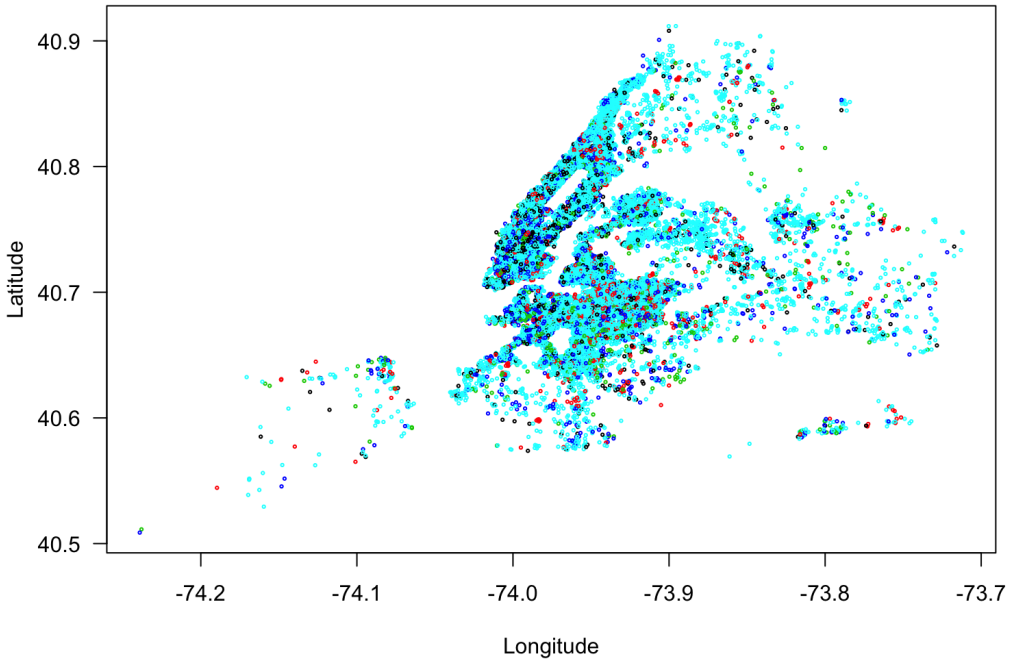
Algorithm 10.1 *K-Means Clustering*

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.
   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

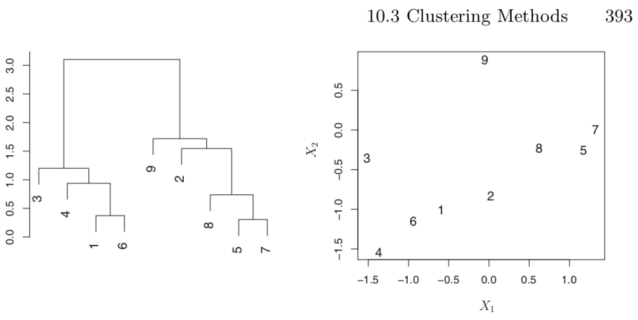
- The user selects how many clusters to divide the data into.
- Initial randomization of the clusters means that the algorithm can produce different results each time.

Implementing K-Means

```
clust.dat <- dat[, .SD, .SDcols = the.variables]
clust.kmeans <- kmeans(x = clust.dat, centers = 5)
plot(x = clust.dat[, get(long.name)], y = clust.dat[, get(lat.name)],
    col = clust.kmeans$cluster, cex = 0.3, xlab = "Longitude",
    ylab = "Latitude", las = 1)
```



Hierarchical Clustering



- Each point starts out as its own group.
- At each step, the two closest groups are fused into a single group.
- There are different **linkage methods** (e.g. complete, average, median, etc.) for deciding how close each group is to the others. These selections can lead to different hierarchies.

Implementing Hierarchical Clustering

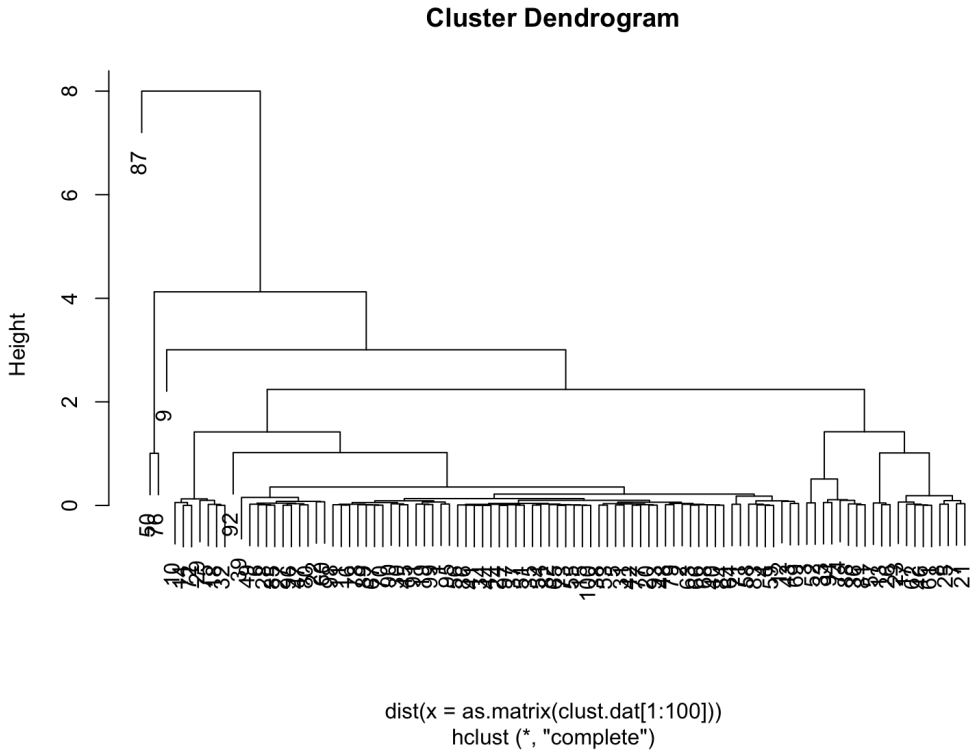
```
toc <- Sys.time()
clust.hier <- hclust(d = dist(x = as.matrix(clust.dat[1:100])))
tic <- Sys.time()
print(tic - toc)
```

```
Time difference of 0.003302813 secs
```

- The running time for hierarchical clustering methods is necessarily slow.
- As the number of features increase, the pairwise distance calculations between the points require more computation.
- As the sample size increases, the pairwise distance calculations and linkage calculations also increase. Each step in the hierarchy requires evaluating all of the pairs of groups!

Plotting the Hierarchy

```
plot(clust.hier)
```



dist(x = as.matrix(clust.dat[1:100]))
hclust (*, "complete")

Some Problems with Clustering

- **K-Means**: The choice of **K** is arbitrary, and it can be hard to find meaning in the selected groups.
- **Hierarchical**: The running time is slow for large data sets, and the view of the hierarchy requires judgments in how far to zoom in.
- **Overall**: It is hard to evaluate the impact of clustering methods because there is no outcome to predict. There can be benefits to clustering. However, there are limits to what Unsupervised Learning can do to further your understanding of the data.

Other Forms of Unsupervised Learning

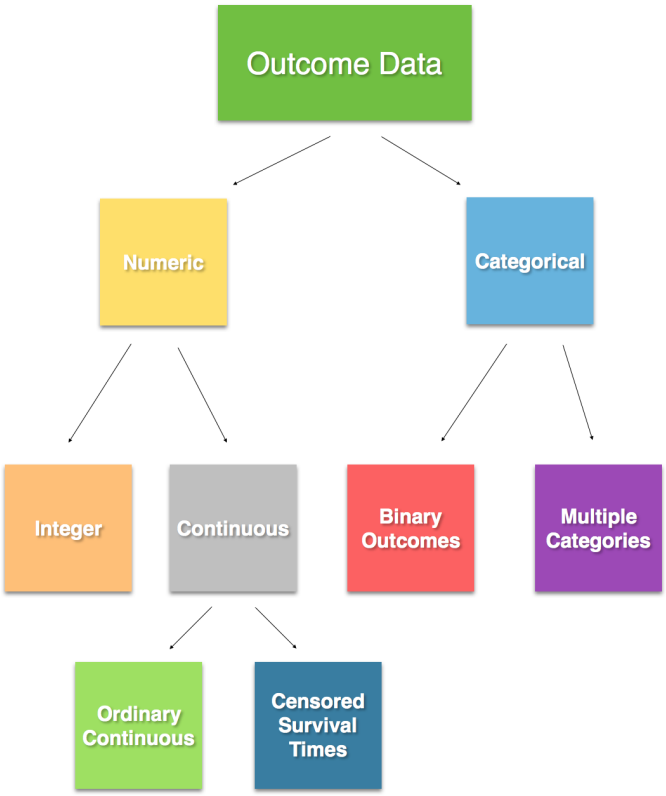
- **Principal Components Analysis**: A method used to explore relationships in high-dimensional data with the goal of reducing the complexity to a smaller number of features.

- **Qualitative Assessments**: Using your expertise of the domain to find relationships in the information. If you know that real estate is very different in urban versus suburban markets, or in condominiums versus single family homes, then any clustering techniques should consider these known factors.

Supervised Learning

- All of the investigations are driven by better understanding the outcome.
- Much of what is possible is determined by the outcome's type of data.
- Then, how to proceed depends on the goals of what you'd like to do to better understand the outcome and its relationship to the other variables.

Types of Outcomes



Describing the Types

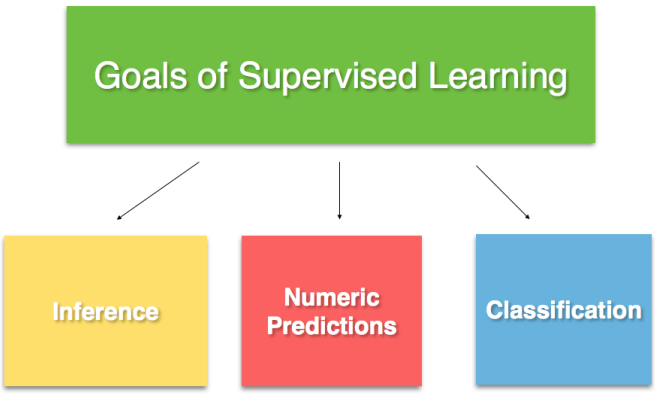
- **Integer**: Numbers that must be described in whole numbers (e.g. the volume of sales)
- **"Ordinary" Continuous**: A continuous measure (e.g. height or weight) that is not subject to **censoring**.
- **Censored Times to Events**: Data on survival time until an event (e.g. mortality, subscribing to an online service) occurs. Censoring (incomplete measurement) typically occurs due to an artificial end to the study or a loss of follow-up for some subjects.
- **Binary Categorical Variables**: Data that must result in one of two outcomes (e.g. TRUE or FALSE, Yes or No, etc.).
- **Multiple Categories**: Data with more than 2 categories (e.g. career paths, types of housing).

This list is not necessarily comprehensive but provides a good overview.

A Range of Methods For Each Setting

- Each type of outcome has some methods that apply and others that do not.
- Censored Survival Times typically require specialized methods that account for incomplete information.
- The **distribution** of the outcome variable can also impact the selection of methods. For instance, **income** variables can have a highly skewed distribution with a heavy right tail. Some methods (e.g. linear regression) may not be immediately suited for this setting even if they are generally built for continuous outcomes.

Differing Objectives



Understanding the Goals

- **Inference**: Drawing meaningful conclusions about the impact of the predicting variables in increasing or decreasing the outcomes.
- **Numeric Predictions**: Generating accurate numeric estimates of the outcomes based on the measured inputs, especially in new subjects for whom data will be prospectively collected.
- **Classification**: Generating categorical classifications as estimates of the outcomes based on the measured inputs, especially in new subjects for whom data will be prospectively collected.

Levels of Information

- **Any numeric prediction** can be converted into a classification.
- For instance, if an algorithm estimates that a user has a 5% chance of subscribing to a service, we could classify that user (and any under a threshold of 30%) as a poor candidate for a promotion.
- However, classifications cannot easily be converted into numeric predictions. If we only know that the user was classified as a poor candidate for a promotion, then we cannot retrieve the probability.

Telling a Story Versus Accuracy

- **Inference** is helpful for evaluating decisions. Which factors are likely to lead to changes in the outcomes, how much change can we expect under different scenarios?

- Just as importantly, inferences can also tell you which factors **are not so important** for creating changes in the outcomes.

- **Numeric Predictions** and **Classifications** are useful for building products. When success is driven by accuracy, then getting the best predictions or classifications will be more important than the story about why things change.

And What are We Estimating/Predicting?

- Every machine learning method aims to optimize some **parameter of the distribution of the outcome**.
- The most typical parameter is the **conditional expected value of the outcome** given the values of the inputs: $E[Y|X_1, X_2, \ldots, X_k]$.
- The **conditional** part is critical here. The outcome **Y** is a function of the inputs, and the idea is to use the value of the inputs to estimate/predict the value of the outcome.
- However, many other parameters, such as the conditional median or other percentiles, are also reasonable quantities to estimate.

Judging Accuracy

- For **numeric predictions**, especially when the parameter to estimate is an expected value, we typically judge the accuracy of the predictions using the root mean squared error (RMSE):

```
my.rmse <- function(predicted, actual, na.rm = TRUE) {
    return(sqrt(mean((predicted - actual)^2, na.rm = na.rm)))
}
```

- For **classifications**, we typically judge the accuracy based on the percentage that are correctly classified:

```
percentage.correctly.classified <- function(predicted, actual,
    na.rm = TRUE) {
    return(mean(predicted == actual, na.rm = na.rm))
}
```

- A variety of other criteria may also be used.

Training and Testing Sets

- A model is estimated based on a **training set** of data.
- Once estimated, the model can be applied to any data set of a similar structure **to make predictions about the outcomes from the inputs**.
- New predictions are typically made on a **testing set** of data that was not used to estimate the model. Many data sets are randomly split with 80% of the data as a training set and 20% as a testing set:

```
training.row.name <- "training_row"
dat[, eval(training.row.name) := sample(x = c(TRUE, FALSE), size = .N, replace = TRUE, prob = c(0.8, 0.2))]
```

Overfitting

- Predictions on the training set may be **overfit** – relying too much on what was already seen – at the expense of accuracy on testing sets. This creates a **bias** that can degrade the quality of the model.
- Predictions on a testing set are a more fair way to judge the overall quality of a model. Selecting a model based on accuracy in the testing set is less prone to biases and overfitting.

Catalogging the Methods of Supervised Learning

- We will provide an overview of many useful methods.
- Each method will be discussed in terms of types of outcomes, range of uses, and overall quality.
- However, this presentation is only the beginning in terms of understanding the theory, capabilities, and limitations of these methods.

Categories of Methods

- Classical Models
- Regularized Regression
- Tree-Based Models
- Other Classes

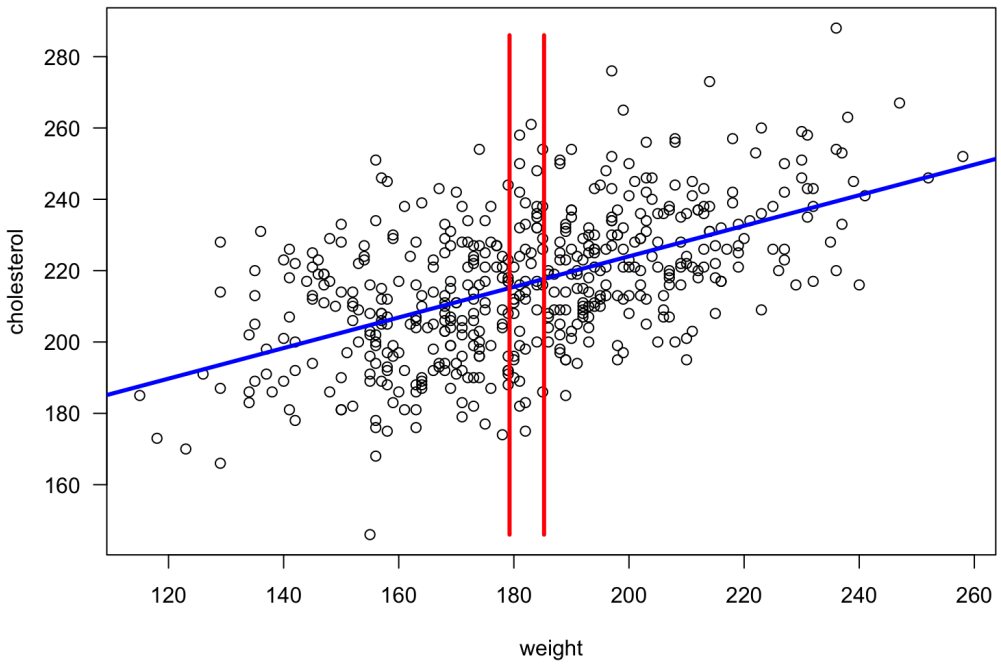
Classical Models

A number of techniques are considered the classical statistical models.

- **Linear Regression**: This is useful for modeling continuous outcomes based on a range of inputs.
- **Logistic Regression**: This is useful for modeling binary outcomes based on a range of inputs.
- **Cox Proportional Hazards Regression**: This is useful for modeling censored survival times based on a range of inputs.

We will discuss each of these classical methods in greater detail.

Linear Regression



Linear Regression: Setting

- **Typical Outcomes**: Continuous variables. Integer-valued measures can be used when rounding is not a concern.
- A **linear function** $E[Y|X] = \beta_0 + \sum_{k=1}^{m} \beta_k X_k$ must be specified.
- The coefficients β_k show the linear effect of increasing a variable X_k by 1 unit on the outcome Y.
- The coefficients are estimated by minimizing the sum of squares, with the famous matrix formula $\hat{\beta} = (X^T X)^{-1} X^T Y$.
- Then predictions can be made by applying the estimated coefficients to a data set.

Creating Formulas

From our previous lecture:

```
create.formula <- function(outcome.name, input.names, input.patterns = NA,
    all.data.names = NA, return.as = "character") {
    variable.names.from.patterns <- c()
    if (!is.na(input.patterns[1]) & !is.na(all.data.names[1])) {
        pattern <- paste(input.patterns, collapse = "|")
        variable.names.from.patterns <- all.data.names[grep(pattern = pattern,
            x = all.data.names)]
    }
    all.input.names <- unique(c(input.names, variable.names.from.patterns))
    all.input.names <- all.input.names[all.input.names !=
        outcome.name]
    if (!is.na(all.data.names[1])) {
        all.input.names <- all.input.names[all.input.names %in%
            all.data.names]
    }
    input.names.delineated <- sprintf("`%s`", all.input.names)
    the.formula <- sprintf("`%s` ~ %s", outcome.name, paste(input.names.delineated,
        collapse = " + "))
    if (return.as == "formula") {
        return(as.formula(the.formula))
    }
    if (return.as != "formula") {
        return(the.formula)
    }
}
```

Reduce Formula Function

```
reduce.formula <- function(dat, the.initial.formula, max.categories = NA) {
    require(data.table)
    dat <- setDT(dat)

    the.sides <- strsplit(x = the.initial.formula, split = "~")[[1]]
    lhs <- trimws(x = the.sides[1], which = "both")
    lhs.original <- gsub(pattern = "`", replacement = "",
        x = lhs)
    if (!(lhs.original %in% names(dat))) {
        return("Error:  Outcome variable is not in names(dat).")
    }

    the.pieces.untrimmed <- strsplit(x = the.sides[2], split = "+",
        fixed = TRUE)[[1]]
    the.pieces.untrimmed.2 <- gsub(pattern = "`", replacement = "",
        x = the.pieces.untrimmed, fixed = TRUE)
    the.pieces.in.names <- trimws(x = the.pieces.untrimmed.2,
        which = "both")

    the.pieces <- the.pieces.in.names[the.pieces.in.names %in%
        names(dat)]
    num.variables <- length(the.pieces)
    include.pieces <- logical(num.variables)

    for (i in 1:num.variables) {
        unique.values <- dat[, unique(get(the.pieces[i]))]
        num.unique.values <- length(unique.values)
        if (num.unique.values >= 2) {
            include.pieces[i] <- TRUE
        }
        if (!is.na(max.categories)) {
            if (dat[, is.character(get(the.pieces[i])) |
                is.factor(get(the.pieces[i]))] == TRUE) {
                if (num.unique.values > max.categories) {
                  include.pieces[i] <- FALSE
                }
            }
        }
    }
    pieces.rhs <- sprintf("`%s`", the.pieces[include.pieces ==
        TRUE])
    rhs <- paste(pieces.rhs, collapse = " + ")
    the.formula <- sprintf("%s ~ %s", lhs, rhs)
    return(the.formula)
}
```

Regression Wrapper Functions

We will build some useful tools for implementing and summarizing linear and logistic regressions:

```
linear.regression.summary <- function(lm.mod, digits = 3, alpha = 0.05) {
  lm.coefs <- as.data.table(summary(lm.mod)$coefficients,
  keep.rownames = TRUE)
  setnames(x = lm.coefs, old = "rn", new = "Variable")
  z <- qnorm(p = 1 - alpha/2, mean = 0, sd = 1)
  lm.coefs[, Coef.Lower.95 := Estimate - z * `Std. Error`]
  lm.coefs[, Coef.Upper.95 := Estimate + z * `Std. Error`]
  return(lm.coefs)
}
logistic.regression.summary <- function(glm.mod, digits = 3, alpha = 0.05) {
  glm.coefs <- as.data.table(summary(glm.mod)$coefficients,
  keep.rownames = TRUE)
  setnames(x = glm.coefs, old = "rn", new = "Variable")
  z <- qnorm(p = 1 - alpha/2, mean = 0, sd = 1)
  glm.coefs[, Odds.Ratio := exp(Estimate)]
  glm.coefs[, OR.Lower.95 := exp(Estimate - z * `Std. Error`)]
  glm.coefs[, OR.Upper.95 := exp(Estimate + z * `Std. Error`)]
  return(glm.coefs[])
}
```

Fitting Functions

```
round.numerics <- function(x, digits) {
    if (is.numeric(x)) {
        x <- round(x = x, digits = digits)
    }
    return(x)
}
fit.model <- function(dat, the.initial.formula, model.type,
    digits = 3) {
    the.formula <- reduce.formula(dat = dat, the.initial.formula = the.initial.formula)
    if (model.type == "logistic") {
        mod <- glm(formula = the.formula, family = "binomial",
            data = dat)
        mod.summary <- logistic.regression.summary(glm.mod = mod,
            digits = digits)
    }
    if (model.type == "linear") {
        mod <- lm(formula = the.formula, data = dat)
        mod.summary <- linear.regression.summary(lm.mod = mod,
            digits = digits)
    }
    mod.summary.rounded <- mod.summary[, lapply(X = .SD,
        FUN = "round.numerics", digits = digits)]
    return(list(summary = mod.summary.rounded, obj = mod))
}
```

Linear Price Model

```
price.name <- "price"
old.borough.name <- "neighbourhood_group_cleansed"
borough.name <- "Borough"
input.names <- c(num.bedrooms.name, num.bathrooms.name,
    cozy.name, borough.name)
setnames(x = dat, old = old.borough.name, new = borough.name)
formula.price <- create.formula(outcome.name = price.name,
    input.names = input.names)
mod.lm <- fit.model(dat = dat[get(training.row.name) ==
    TRUE, ], the.initial.formula = formula.price, model.type = "linear")
```

Price Model Results

```
datatable(data = mod.lm$summary, rownames = FALSE)
```

Show 10 entries Search:

Variable	Estimate	Std. Error	t value	Pr(>|t|)	Coef.Lower.95	Coef.Upper.95
(Intercept)	-39.136	4.388	-8.918	0	-47.737	-30.535
bedrooms	60.501	0.831	72.776	0	58.872	62.131
bathrooms	37.814	1.476	25.621	0	34.921	40.707
cozy	-10.723	1.428	-7.511	0	-13.52	-7.925
BoroughBrooklyn	35.462	4.154	8.536	0	27.319	43.604
BoroughManhattan	98.157	4.149	23.657	0	90.025	106.289
BoroughQueens	17.68	4.416	4.003	0	9.024	26.336
BoroughStaten Island	-2.936	8.559	-0.343	0.732	-19.711	13.839

Showing 1 to 8 of 8 entries Previous 1 Next

The prices of the rentals are positively associated with increases in the number of bedrooms, bathrooms, and in the boroughs of Brooklyn, Manhattan, and Queens (relative to the Bronx). A **cozy** description was associated with lower prices. All of these effects are statistically significant. Staten Island has lower prices, but the effect is not statistically significant and may in fact be the same as in the Bronx.

The Accuracy of Linear Regression's Predictions

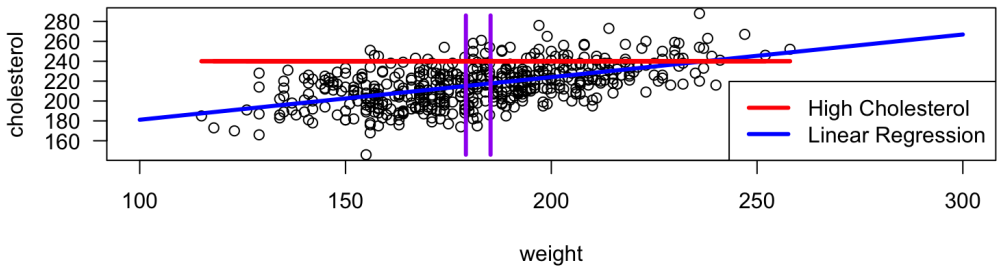
```
pred.lm <- predict.lm(object = mod.lm$obj, newdata = dat[get(training.row.name) ==
    FALSE])
rmse.lm <- my.rmse(predicted = pred.lm, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
print(round(x = rmse.lm, digits = 1))
```

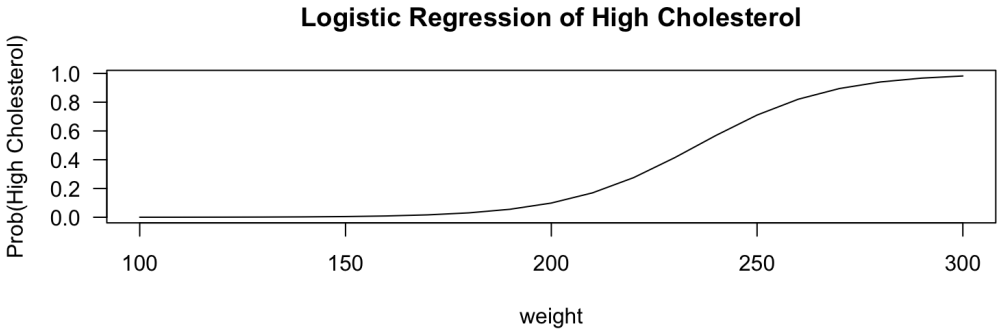
```
[1] 87
```

This is an improvement on the standard deviation of the prices: 106.

In other words, measuring the information in the predicting variables leads to better predictions!

Logistic Regression





Logistic Regression: Setting

- **Typical Outcomes**: Binary Variables: TRUE or FALSE, Yes or No, etc.
- A **linear function** $\log \frac{p}{1-p} \approx \beta_0 + \sum_{i=1}^{m} \beta_i X_i$ must be specified.
- The coefficients β_k show the log-linear effect of increasing a variable X_k by 1 unit on the **odds** of the outcome Y being **successful**.
- It is important to define a successful outcome means (e.g. with the value TRUE or 1). Otherwise, you might accidentally flip the results.
- The **Odds Ratio** $\exp(\beta_k)$ shows the **approximate proportional increase** in the risk of the outcome for each 1-unit increase in variable X_k.

Logistic Regression: High Rental Prices

- Let's say that an AirBnB rental has a **high price** if it costs at least $200:

```
high.price.name <- "High Price"
threshold.high.price <- 200
dat[, eval(high.price.name) := (get(price.name) >= threshold.high.price)]
```

- Let's build a logistic regression model for high prices:

```
formula.high.price <- create.formula(outcome.name = high.price.name,
    input.names = input.names)
mod.logistic <- fit.model(dat = dat[get(training.row.name) ==
    TRUE, ], the.initial.formula = formula.high.price, model.type = "logistic")
```

Model Results

```
datatable(data = mod.logistic$summary, rownames = FALSE)
```

Show 10 entries Search:

Variable	Estimate	Std. Error	z value	Pr(>|z|)	Odds.Ratio	OR.Lower.95	OR.Upper.95
(Intercept)	-7.337	0.475	-15.459	0	0.001	0	0.002
bedrooms	1.367	0.03	44.924	0	3.923	3.696	4.164
bathrooms	0.627	0.049	12.787	0	1.873	1.701	2.062
cozy	-0.385	0.058	-6.68	0	0.68	0.608	0.762
BoroughBrooklyn	2.458	0.468	5.254	0	11.68	4.669	29.216
BoroughManhattan	4.029	0.468	8.612	0	56.219	22.472	140.642
BoroughQueens	1.657	0.477	3.472	0.001	5.245	2.058	13.368
BoroughStaten Island	0.502	0.68	0.738	0.46	1.652	0.435	6.27

Showing 1 to 8 of 8 entries Previous 1 Next

Additional bedrooms, bathrooms, and a location in Manhattan, Brooklyn, or Queens (relative to the Bronx) are all greatly associated with increases in the odds of having a high-priced rental. A cozy description was associated with a decrease in odds of a high priced rental. All of these effects are statistically significant, while Staten Island has no significant difference with the Bronx in terms of these odds.

Logistic Regression's Predictions

```
pred.logistic <- predict.glm(object = mod.logistic$obj,
    newdata = dat[get(training.row.name) == FALSE], type = "response")
rmse.logistic <- my.rmse(predicted = pred.logistic, actual = dat[get(training.row.name) ==
    FALSE, get(high.price.name)])
print(round(x = rmse.logistic, digits = 2))
```

```
[1] 0.32
```

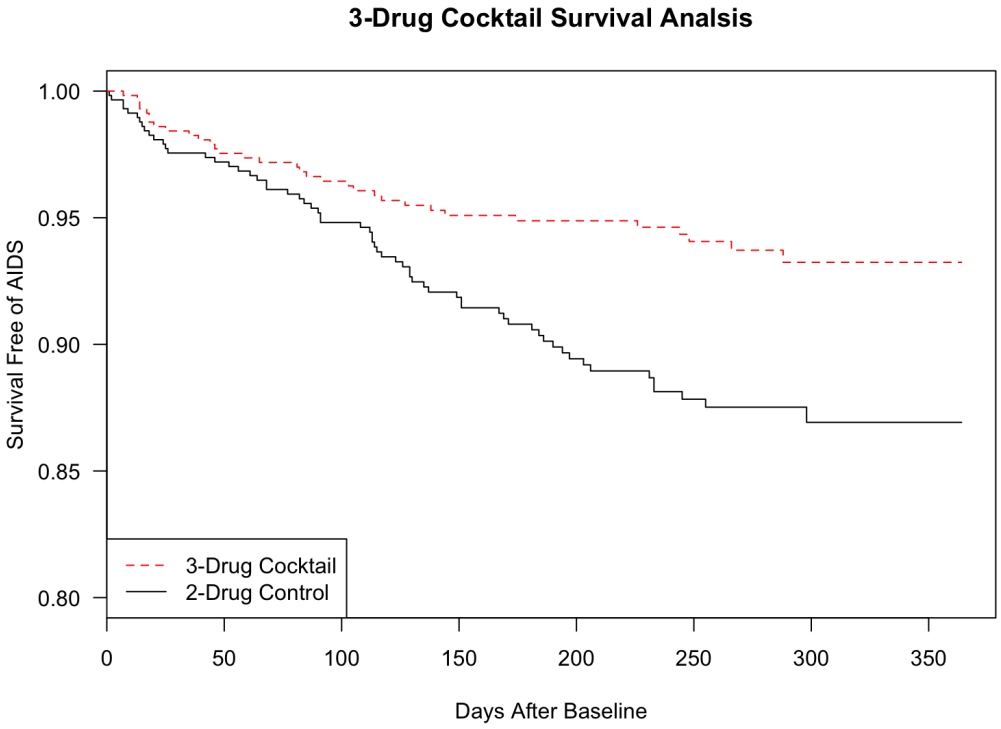
Logistic Regression's Classifications

```
classifications.logistic <- 1 * (pred.logistic >= 0.5)
correct.pct.logistic <- percentage.correctly.classified(predicted = classifications.logistic,
    actual = dat[get(training.row.name) == FALSE, get(high.price.name)])
print(round(x = correct.pct.logistic, digits = 3))
```

```
[1] 0.876
```

The logistic regression correctly classified the testing set's outcomes a reasonably high percentage of the time.

Cox Proportional Hazards Regression



The Setting of Cox Regression

- **Typical Outcome**: Survival times are measured, possibly with **censored data**.
- Treatment groups can be compared in a bivariate way using **survival curves**.
- Multivariable adjustment is possible with a Regression approach. The equation is:

Conditional Hazard of the outcome at time t given inputs X = Baseline Hazard of the outcome **times** $exp(\sum_{k=1}^{m} \beta_k X_k)$

- Just as in Logistic Regression, it is important to define what constitutes an adverse outcome (e.g. mortality or an AIDs diagnosis) versus its opposite (survival).

The Hazard Ratio

- The **Hazard Ratio** $\exp(\beta_k)$ shows the approximate proportional increase in the hazard of an adverse event for each 1-unit increase in variable X_k.
- Hazard Ratios **less than 1** show reductions in the risk of an adverse event.
- Hazard Ratios **close to 1** show no change in the risk of an adverse event.
- Hazard Ratios **greater than 1** show increases in the risk of an adverse event.

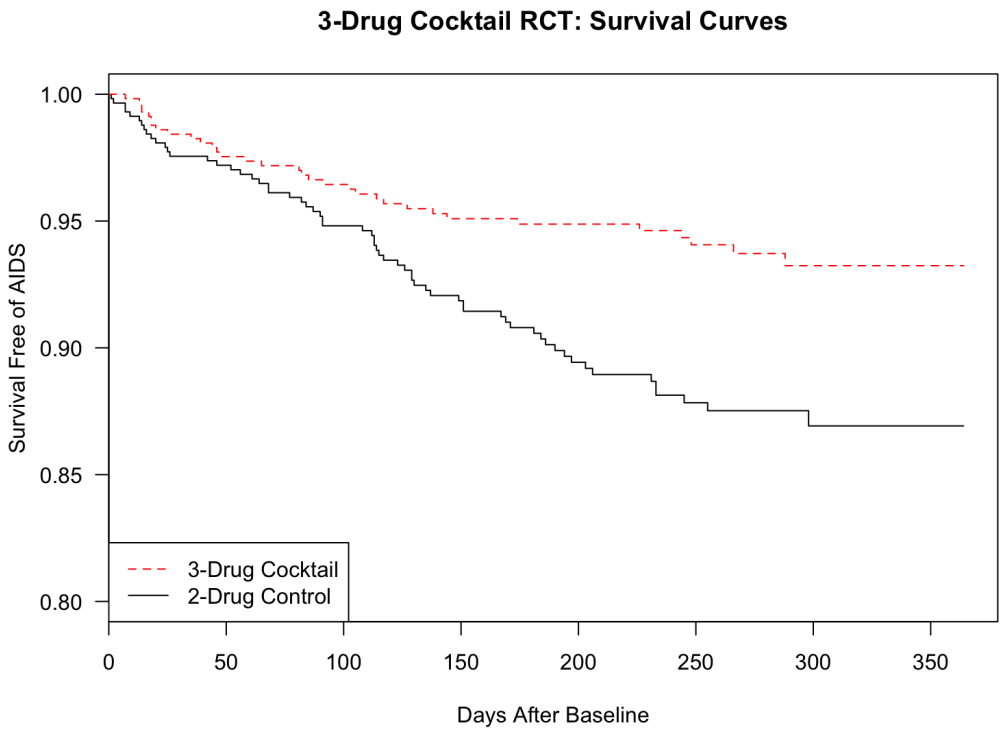
A Detour: The HIV Data from Lecture 3

```
time.name <- "time"
outcome.name <- "censor"
tx.name <- "tx"
sex.name <- "sex"
age.name <- "age"
ivdrug.name <- "ivdrug"
cd4.name <- "cd4"
predictor.names <- c(tx.name, sex.name, age.name, ivdrug.name,
    cd4.name)
hiv.dat <- fread(input = "AIDS data.csv")
```

Plotting Survival Curves

We'll display the comparative survival curves free of AIDs (where 1 is 100%) for the two treatment groups:

```
library(survival)
surv.obj <- Surv(time = hiv.dat[, get(time.name)], event = hiv.dat[,
    get(outcome.name)])
plot(survfit(formula = surv.obj ~ hiv.dat[, get(tx.name)]),
    las = 1, lty = c(1, 2), col = c(1, 2), ylim = c(0.8,
        1), xlab = "Days After Baseline", ylab = "Survival Free of AIDS",
    main = "3-Drug Cocktail RCT: Survival Curves")
legend(x = "bottomleft", legend = c("3-Drug Cocktail", "2-Drug Control"),
    lty = c(2, 1), col = c(2, 1))
```



Cox Regression of AIDs-Free Survival

```
the.formula <- as.formula(sprintf("surv.obj ~ %s", paste(sprintf("hiv.dat$%s",
    predictor.names), collapse = "+")))
coxmod <- coxph(formula = the.formula)
the.coefs <- as.data.table(summary(coxmod)$coefficients,
    keep.rownames = TRUE)
setnames(x = the.coefs, old = names(the.coefs), new = c("Variable",
    "Coefficient", "Hazard Ratio", "Std. Error", "Z", "P"))
alpha <- 0.05
zval <- qnorm(p = 1 - alpha/2, mean = 0, sd = 1)
the.coefs[, `:=`(HR.Lower.95, exp(Coefficient - zval * `Std. Error`))]
the.coefs[, `:=`(HR.Upper.95, exp(Coefficient + zval * `Std. Error`))]
the.coefs[, `:=`(Variable, gsub(pattern = "hiv.dat$", replacement = "",
    x = Variable, fixed = TRUE))]
```

Interpreting the Coefficients

```
datatable(data = the.coefs[, lapply(X = .SD, FUN = "round.numerics",
    digits = 3)], rownames = FALSE)
```

Show 10 entries Search:

Variable	Coefficient	Hazard Ratio	Std. Error	Z	P	HR.Lower.95	HR.Upper.95
tx	-0.663	0.515	0.215	-3.082	0.002	0.338	0.786
sex	0.112	1.119	0.284	0.394	0.693	0.641	1.952
age	0.029	1.03	0.011	2.643	0.008	1.008	1.053
ivdrug	-0.245	0.783	0.164	-1.488	0.137	0.567	1.081
cd4	-0.017	0.983	0.003	-6.537	0	0.979	0.988

Showing 1 to 5 of 5 entries Previous 1 Next

The 3-Drug Cocktail significantly reduces the hazard of AIDS or Death while adjusting for a variety of additional risk factors.

Regularized Regression: Motivation

- What can we do about a classical model's tendency to overfit the training data when the sample size is small or moderate?
- Is there a way to **automatically select** the most important variables that should be in a model?
- Can we overcome some of the limitations of the classical models?

Types of Regularization and Variable Selection Algorithms

- **Ridge Regression**: shrinks the value of all coefficients to create a more moderate estimate, which is less likely to overfit the training data.
- **Lasso Regression**: performs variable selection to choose a small number of variables for inclusion in the model.
- **Other Variable Selection Procedures**: Early methods included stepwise selection methods (forward and backward) that select new variables to improve the relative fit. However, these methods often have poor performance.

Ridge Regression's Setting

obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance. The two best-known techniques for shrinking the regression coefficients towards zero are *ridge regression* and the *lasso*.

6.2.1 Ridge Regression

Recall from Chapter 3 that the least squares fitting procedure estimates $\beta_0, \beta_1, \ldots, \beta_p$ using the values that minimize

$$\text{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

Ridge regression is very similar to least squares, except that the coefficients are estimated by minimizing a slightly different quantity. In particular, the ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

ridge regression

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \text{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2, \qquad (6.5)$$

where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately. Equa-

tuning

Lasso Regression's Setting

6.2.2 The Lasso

Ridge regression does have one obvious disadvantage. Unlike best subset, forward stepwise, and backward stepwise selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model. The penalty $\lambda \sum \beta_j^2$ in (6.5) will shrink all of the coefficients towards zero, but it will not set any of them exactly to zero (unless $\lambda = \infty$). This may not be a problem for prediction accuracy, but it can create a challenge in model interpretation in settings in which the number of variables p is quite large. For example, in the `Credit` data set, it appears that the most important variables are `income`, `limit`, `rating`, and `student`. So we might wish to build a model including just these predictors. However, ridge regression will always generate a model involving all ten predictors. Increasing the value of λ will tend to reduce the magnitudes of the coefficients, but will not result in exclusion of any of the variables.

The *lasso* is a relatively recent alternative to ridge regression that overcomes this disadvantage. The lasso coefficients, $\hat{\beta}_\lambda^L$, minimize the quantity [1]

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| \;= \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|. \qquad (6.7)$$

Multiple Outcomes

Both Ridge and Lasso regression can be used in a variety of settings:

- Numeric Estimates or Classifications
- Continuous Outcomes
- Binary Outcomes
- Categorical Outcomes
- Censored Survival Times

Understanding the Penalty

- **Small values of** λ (near zero) will shift the focus back to minimizing on the **Mean Squared Error** – essentially bringing us back to a classical regression model.

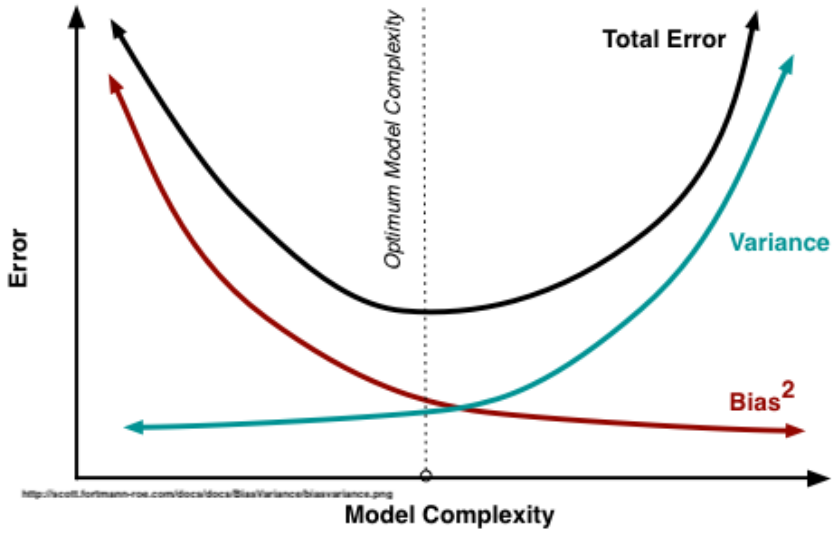
- **Large values of** λ will lead to greater penalties for non-zero coefficients. Larger values of λ effectively lead to **shrinkage** in the estimates of the coefficients toward zero.

- The **Lasso's** L_1 penalty is larger than Ridge's L_2 penalty when $0 < \beta < 1$. Therefore, larger values of λ for the Lasso will more quickly force more of the coefficients to be **exactly zero**. This is very helpful for **selecting a small number of variables** to be included in the model while zeroing out the rest.

Selecting λ

- Because each data set has a different set of variables and a different relationship between the outcome and the inputs, it is **impossible to say ahead of time** what the best value of λ should be.
- **Selecting the right value of** λ can lead to improved predictions on testing sets, while other choices may lead to a poor performance.
- Ultimately, it would help to think of this selection within a broader **framework of optimization**.

A Tradeoff Between Bias and Variance



Understanding the Tradeoffs

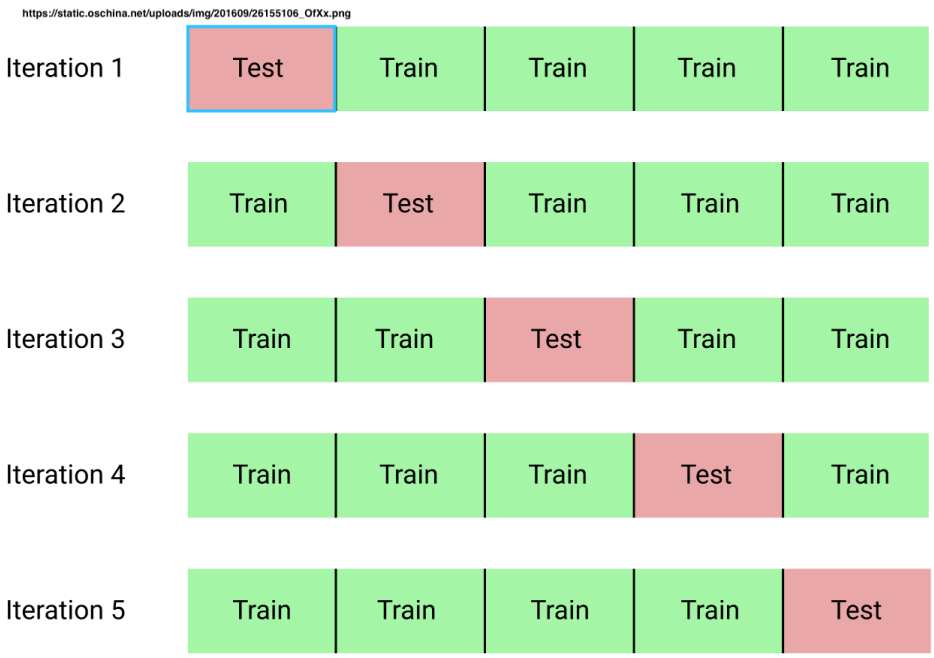
- More complex models (more variables, greater dimensions, etc.) measure more things and can therefore be less biased.
- However, greater complexity simultaneously leads to greater variability in the model.
- The Expected Testing MSE, described by the equation below, is controlled by the **variance of the model**, the **squared bias of the model**, and the natural variance of the error terms.

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \mathrm{Var}(\hat{f}(x_0)) + [\mathrm{Bias}(\hat{f}(x_0))]^2 + \mathrm{Var}(\epsilon). \qquad (2.7)$$

- Finding the proper balance between the bias and variance of a model is the great challenge of predictive modeling. Doing this well can lead to more accurate predictions.

K-Fold Cross Validation

- Splits the training set into K smaller random subsets (mutually exclusive, collectively exhaustive). K = 5 is a typical choice.
- Each subgroup serves as the testing subset once.



Selecting λ with K-Fold Cross Validation

- Start by making a list of possible values of λ (e.g. numbers from 0 to 5 in increments of 0.1).
- For each k from 1 to K, use the **k**th training and testing sets to fit each of the models (as defined by the different possible values of λ), make predictions, and evaluate the accuracy.
- Then, across the K testing sets, **average the accuracy** of each model (as defined by a specific value of λ).
- Finally, select the model with the best accuracy.

Cross Validated RMSE

Model	k = 1	k = 2	k = 3	k = 4	k = 5	Average Accuracy
Model 1	0.85	0.93	0.72	0.97	0.84	**0.862**
Model 2	0.91	1.15	0.66	0.84	0.65	**0.842**
Model 3	1.6	0.91	0.81	1.14	0.72	**1.036**
Model 4	1.31	0.68	1.05	0.94	0.61	**0.918**
Model 5	1.42	0.82	0.64	0.78	0.86	**0.904**
Model 6	0.77	1.22	1.57	1.25	1.12	**1.186**
Model 7	1.42	0.74	0.97	0.69	1.3	**1.024**
Model 8	0.66	1.58	0.61	1.52	0.64	**1.002**
Model 9	0.79	1.06	1.25	0.96	1.41	**1.094**
Model 10	0.7	1.54	1.32	1.26	1.57	**1.278**

Regularized Regression with glmnet

```
create.x.and.y <- function(the.formula, data) {
    require(data.table)
    setDT(data)
    x <- model.matrix(object = as.formula(the.formula),
        data = data)
    y.name <- trimws(x = gsub(pattern = "`", replacement = "",
        x = strsplit(x = the.formula, split = "~")[[1]][1],
        fixed = TRUE))
    y <- data[as.numeric(rownames(x)), get(y.name)]
    return(list(x = x, y = y))
}
my.regularized.model <- function(the.formula, training.data,
    testing.data, alpha = 0, family = "gaussian") {
    library(glmnet)
    x.y.train <- create.x.and.y(the.formula = formula.price,
        data = training.data)
    mod <- glmnet(x = x.y.train$x, y = x.y.train$y, family = family,
        alpha = alpha)
    the.coefs <- as.data.table(mod$beta[, ncol(mod$beta)],
        keep.rownames = TRUE)
    setnames(x = the.coefs, old = names(the.coefs), new = c("Variable",
        "Beta"))

    x.y.test <- create.x.and.y(the.formula = the.formula,
        data = testing.data)
    pred <- predict(object = mod, newx = x.y.test$x, type = "response")
    the.rmse <- my.rmse(predicted = pred[, ncol(pred)],
        actual = x.y.test$y)
    return(list(coefs = the.coefs, pred = pred, rmse = the.rmse,
        lambda = mod$lambda[length(mod$lambda)]))
}
```

Ridge Regression on AirBnB

```
mod.ridge <- my.regularized.model(the.formula = formula.price,
    training.data = dat[get(training.row.name) == TRUE,
        ], testing.data = dat[get(training.row.name) ==
        FALSE, ], alpha = 0, family = "gaussian")
datatable(data = mod.ridge$coefs[, lapply(X = .SD, FUN = "round.numerics",
    digits = 3)], rownames = FALSE)
```

Show 10 entries Search:

Variable	Beta
(Intercept)	0
bedrooms	57.868
bathrooms	37.61
cozy	-10.573
BoroughBrooklyn	0.891
BoroughManhattan	61.749
BoroughQueens	-16.328
BoroughStaten Island	-35.284

Showing 1 to 8 of 8 entries Previous 1 Next

Lasso Regression on AirBnB

```
mod.lasso <- my.regularized.model(the.formula = formula.price,
    training.data = dat[get(training.row.name) == TRUE,
        ], testing.data = dat[get(training.row.name) ==
        FALSE, ], alpha = 1, family = "gaussian")
datatable(data = mod.lasso$coefs[, lapply(X = .SD, FUN = "round.numerics",
    digits = 3)], rownames = FALSE)
```

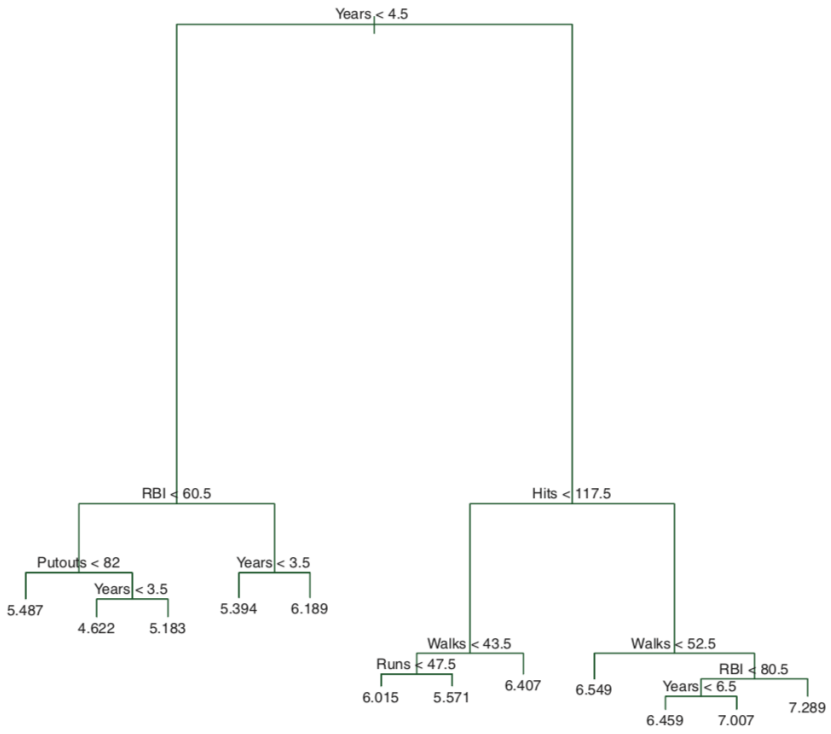
Show 10 entries Search:

Variable	Beta
(Intercept)	0
bedrooms	60.471
bathrooms	37.733
cozy	-10.635
BoroughBrooklyn	32.038
BoroughManhattan	94.754
BoroughQueens	14.217
BoroughStaten Island	-5.782

Showing 1 to 8 of 8 entries Previous 1 Next

Tree-Based Models

- Decision trees naturally align with decisions.
- Many people understand categorical splits better than, say, the coefficients in the classical regression models.
- For these reasons, tree models are considered very easy to interpret.



Multiple Outcomes

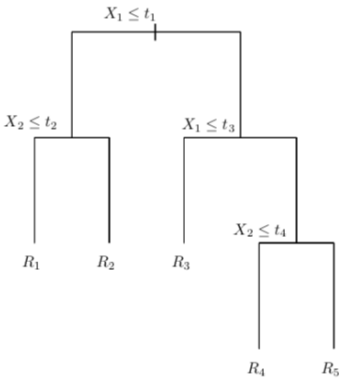
- Numeric Estimates or Classifications
- Continuous Outcomes
- Binary Outcomes
- Categorical Outcomes
- Inferences and Predictions are possible

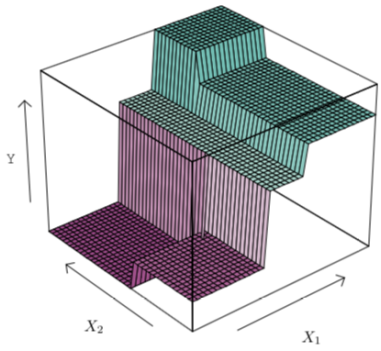
Decision Trees: The Simple Version

- From the top (the **root**) of the tree, the best variable is selected.
- The best categorical split of that variable (into 2 or more branches) is selected.
- All of the selections are made to minimize a criterion such as the **Gini index** (total variation across the categorical splits) or the **entropy**.
- Branching continues until a stopping criterion (e.g. little or no reduction in the Gini index or entropy with further branching) is reached.

Making Predictions or Classifications with a Tree

- Branching effectively **partitions** the space of the variables into a number of different regions.
- Then, within each partitioned region, an overall estimate is produced.
- For **predictions**, we can average the outcomes within a partition. (Likewise, we could also estimate the median, a percentile, or any other parameter.)
- For **classifications**, a method like **plurality voting** can be used to select one categorical outcome to represent the partition.





A Decision Tree Model of Price

```
library(rpart)
mod.rpart <- rpart(formula = formula.price, data = dat[get(training.row.name) ==
    TRUE, ])
pred.rpart <- predict(object = mod.rpart, newdata = dat[get(training.row.name) ==
    FALSE, ], type = "vector")
rmse.rpart <- my.rmse(predicted = pred.rpart, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
print(rmse.rpart)
```

```
[1] 80.15542
```

Disadvantages of Decision Trees

- Small changes in the data can dramatically change the structure of the tree.
- Inferences are not necessarily reliable when the structure is not robust.
- Furthermore, simple decision trees often have **poor predictive performance** relative to other models.

A Big Advantage of Decision Trees

- Non-linear splits are a natural feature of decision tree models.
- For this reason, decision tree models can better handle more granular features.
- For the AirBnB data, this would enable us to better incorporate **latitude and longitude** rather than boroughs or neighborhoods.

Decision Tree Model with Coordinates

```
rating.name <- "review_scores_rating"
formula.price.lat.long <- create.formula(outcome.name = price.name,
    input.names = c(num.bedrooms.name, num.bathrooms.name,
        cozy.name, lat.name, long.name, rating.name))
mod.rpart.lat.long <- rpart(formula = formula.price.lat.long,
    data = dat[get(training.row.name) == TRUE, ])
pred.rpart.lat.long <- predict(object = mod.rpart.lat.long,
    newdata = dat[get(training.row.name) == FALSE, ], type = "vector")
rmse.rpart.lat.long <- my.rmse(predicted = pred.rpart.lat.long,
    actual = dat[get(training.row.name) == FALSE, get(price.name)])

res <- data.table(`Tree with Borough RMSE` = rmse.rpart,
    `Tree with Coordinates RMSE` = rmse.rpart.lat.long)
datatable(data = res[, lapply(X = .SD, FUN = "round.numerics",
    digits = 3)], rownames = FALSE)
```

Show 10 entries Search:

Tree with Borough RMSE	Tree with Coordinates RMSE
80.155	78.469

Showing 1 to 1 of 1 entries Previous 1 Next

Improving on a Simple Decision Tree

- How can we improve upon the performance of a decision tree?
- **Pruning**: This is a method of removing some of the branches from a larger tree with an eye on improving the performance.
- **More Data**: A tree's accuracy will certainly improve if you're able to collect more data. However, that's not especially helpful for the data that you do have. **Or is it?**

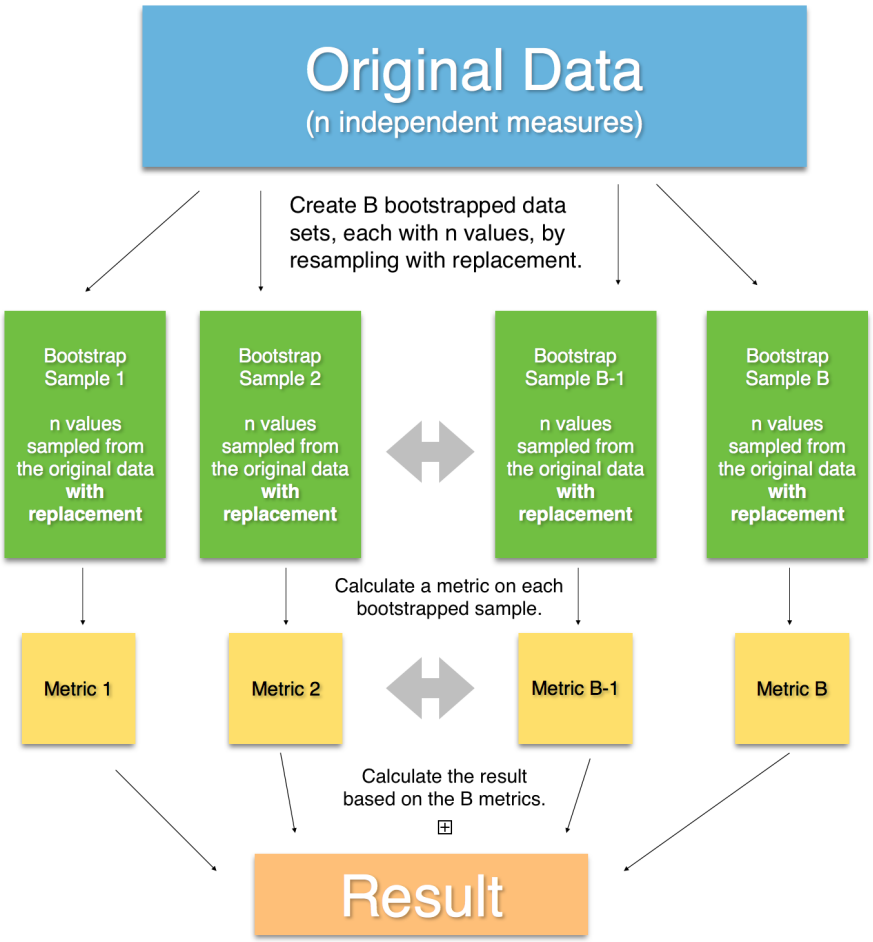
Many Trees

- Our usual scientific standard is that many independent experiments are better than a single one. Statistics shows us that averaging the results of multiple experiments will be more precise (less variation) than any single result would be.
- If we could repeat the experiment to independently collect **n** data points many times (a large number **B**), we could compute **B** trees on samples of size **n**.
- The average of these trees (or plurality vote for classification) would likely improve upon the results of any one tree. This gives us a path toward overcoming the limitations of a single tree. If only we had more data.

The Bootstrap

- The Bootstrap is a general purpose method for approximating the effect of repeating an experiment many times **based only on the data that you've collected**.
- Given a sample of **n** independent, identically distributed observations, we can create variations of these data by **resampling** from the observed data.
- Each Bootstrap sample consists of **n** records that are randomly sampled independently from the original data, drawn **with replacement** and with equal probability.
- You can create as many Bootstrap samples from your data as you'd like. For many applications, it is common to create hundreds or thousands of Bootstrap samples, each of size **n**.

Bootstrapped Metrics



- The metric you're computing can be applied to each Bootstrap sample.
- Then the **B** independent calculations of the metric can be used in multiple ways. The values could be **averaged**. The standard deviation of the **B** values would approximate the **standard error** of the metric. The **2.5th and 97.5th** quantiles of these metrics could be used to approximate a 95% confidence interval.

Bagging

- Bootstrapped Aggregation (Bagging) is a general purpose technique that applies an algorithm to bootstrapped samples and averages their results.
- For decision trees, you could **average the predictions of the B trees** on each record of the testing set. Likewise, classifications could be derived from a **plurality vote** among the **B** trees on each record of the testing set.
- Averaging the predictions (or voting on the classifications) should lead to greater precision than any one tree would likely have. That's an excellent way to improve upon the limitations of one tree.

A Problem with Bagging

- Bagging on decision trees has one **fundamental flaw**: the trees all select the root variables according to the same criteria (Gini index or entropy).
- This means that many trees will have the same few variables at the root. The results will be **highly correlated**.
- Taking the average of correlated variables will be **less precise** than the average of independent (uncorrelated) variables. Therefore, the improvement associated with Bagging on decision trees may be limited.

Random Forests

- How can we overcome Bagging's limitation? A **Random Forest** uses Bagging to build **B** trees from Bootstrap samples… all with one small difference.
- When each Bootstrap sample is fit to a tree, only a **randomly selected subset of all of the predictor variables** is used. No other variables may be applied.
- Because the strongest variables will not always be included, the **B** trees produced by the Random Forest will be far less correlated.
- In fact, Random Forests are often a very competitive method for generating the most accurate predictions and classifications!
- However, this comes at a price of **greater computational complexity** due to the bootstrapping that's involved.

Fitting a Random Forest to AirBnB's Prices

```
library(randomForest)
mod.rf <- randomForest(formula = as.formula(formula.price.lat.long),
    data = dat[get(training.row.name) == TRUE, ])
pred.rf <- predict(object = mod.rf, newdata = dat[get(training.row.name) ==
    FALSE, ])
rmse.rf <- my.rmse(predicted = pred.rf, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
print(rmse.rf)
```

```
[1] 71.27092
```

This is a substantial improvement over the linear regression and simple tree-based models.

Gradient Boosting: Algorithm

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Gradient Boosting: Concepts

- **Bootstrapped samples** are used to fit new tree models, going one at a time.
- The overall model with **b** trees is a sum of the predictions of the trees, each weighted by a **shrinkage parameter** λ.
- Meanwhile, the **(b+1)**st tree is fit to estimate **the errors** of the prior model using only **b** trees.
- In this way, the additional trees work directly on improving the accuracy of the fit to the training set.
- The **shrinkage parameter** λ's value therefore controls **how quickly** the model learns. It is often said that Boosting algorithms perform well because they are **slow learners** that allow a good number of trees (e.g. at least 20-50) to play a role in fitting the model.

Fitting a Gradient Boosting Model to AirBnB's Prices

```
library(xgboost)
dmat.train <- xgb.DMatrix(data = data.matrix(dat[get(training.row.name) ==
    TRUE, .SD, .SDcols = c(num.bedrooms.name, num.bathrooms.name,
    cozy.name, lat.name, long.name, rating.name)]), label = dat[get(training.row.name) ==
    TRUE, get(price.name)])
dmat.test <- xgb.DMatrix(data = data.matrix(dat[get(training.row.name) ==
    FALSE, .SD, .SDcols = c(num.bedrooms.name, num.bathrooms.name,
    cozy.name, lat.name, long.name, rating.name)]), label = dat[get(training.row.name) ==
    FALSE, get(price.name)])

mod.xgb <- xgboost(data = dmat.train, eta = c(0.05, 0.1),
    gamma = c(1, 2), nrounds = 40, verbose = 0)
pred.xgb <- predict(object = mod.xgb, newdata = dmat.test)
rmse.xgb <- my.rmse(predicted = pred.xgb, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
print(rmse.xgb)
```

```
[1] 75.0919
```

Other Models

- There earlier models fell neatly into categories of **classical, regularized, and tree-based** methods.
- However, there are still more methods, from the simple to the complex, that each form their own categories.
- We will explore a few of these models here.

K Nearest Neighbors

- Perhaps the most simple Machine Learning method.
- **The idea**: A user sets a neighborhood parameter **K**. Predictions at each data point are based on the **average** of the outcomes of the **K** nearest neighbors. (For classifications, a plurality vote is used.)
- Like clustering methods in unsupervised learning, K Nearest Neighbors relies on some kind of **distance metric** such as Euclidean distance to determine what constitutes the **K** nearest neighbors.
- Just to be clear, predictions for each point in the testing set are based on the **K** closest points from the training set.
- The value of **K** can also be tuned (e.g. with cross validation) to improve the predictive performance.

K Nearest Neighbors on the AirBnB Prices

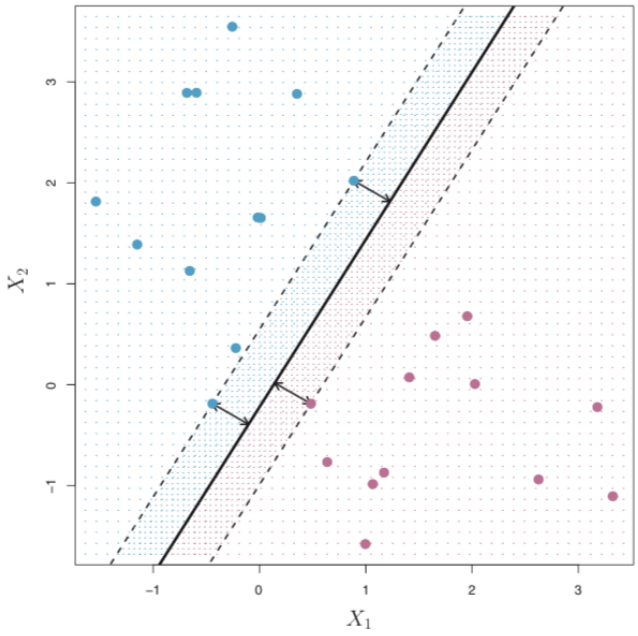
```
library(FNN)
mod.knn <- knn.reg(train = dat[get(training.row.name) ==
    TRUE, .SD, .SDcols = c(num.bedrooms.name, num.bathrooms.name,
    cozy.name, lat.name, long.name, rating.name)], test = dat[get(training.row.name) ==
    FALSE, .SD, .SDcols = c(num.bedrooms.name, num.bathrooms.name,
    cozy.name, lat.name, long.name, rating.name)], y = dat[get(training.row.name) ==
    TRUE, get(price.name)], k = 20)
rmse.knn <- my.rmse(predicted = mod.knn$pred, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
print(rmse.knn)
```

```
[1] 76.89153
```

Because we are including latitude and longitude, the nearest neighbors are likely to be similar units as close as possible to those we are making predictions on.

Support Vector Machines

342 9. Support Vector Machines



- Classifications between groups are built by drawing the best dividing line or curve (maximizing the margin of the border) while allowing for some degree of misclassification.
- Non-linear boundaries like radial functions can be specified with alternative **kernels**.
- Numeric predictions are adapted from the classification method's boundaries.

Support Vector Machines on AirBnB's Prices

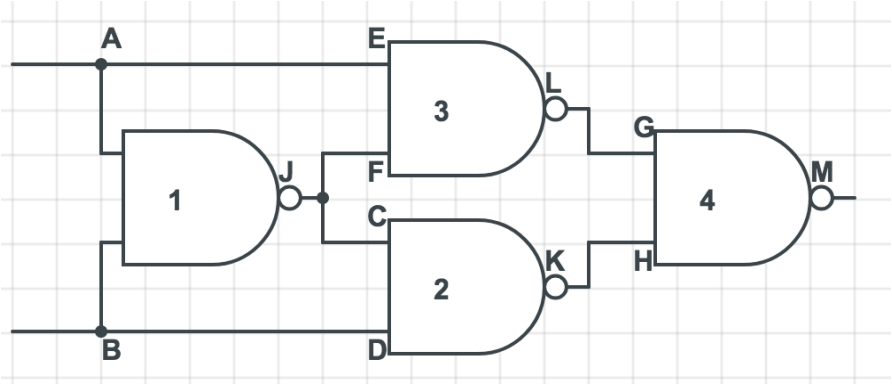
```
library(e1071)
mod.svm <- svm(formula = as.formula(formula.price.lat.long),
    data = dat[get(training.row.name) == TRUE, ])
pred.svm <- predict(object = mod.svm, newdata = dat[get(training.row.name) ==
    FALSE, .SD, .SDcols = c(num.bedrooms.name, num.bathrooms.name,
    cozy.name, lat.name, long.name, rating.name)])
rmse.svm <- my.rmse(predicted = pred.svm, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
print(rmse.svm)
```

```
[1] 74.82019
```

Neural Networks

- A **simple idea**: The connection between the outcome and the inputs can be constructed, much in the way that electrical circuits are built.
- This was initially modeled on (simplistic) notions about how the brains of animals function.
- The **complex reality**: Building circuits across numerous inputs to estimate noisy outcomes can require many layers of interconnections. Describing how that comes together or what it looks like is very complex.
- In modern times, neural networks have been renamed as **deep learning** techniques that utilize intense computations to construct neural networks with many layers.
- Deep learning is currently one of the most powerful techniques for creating accurate predictions.

A Motivating Example: The XOR function



- The **XOR** function of two binary inputs **x** and **y** returns 1 if **x + y = 1** and 0 otherwise. It is an **exclusive or** because both inputs can't simultaneously be true to generate a true response.

Linear Regression Struggles with XOR

```
dt <- data.table(x = c(0, 0, 1, 1), y = c(0, 1, 0, 1), xor = c(0,
    1, 1, 0))
fit.model(dat = dt, the.initial.formula = "xor ~ x + y",
    model.type = "linear")$summary
```

```
      Variable Estimate Std. Error t value Pr(>|t|) Coef.Lower.95
1: (Intercept)      0.5      0.866   0.577    0.667        -1.197
2:           x      0.0      1.000   0.000    1.000        -1.960
3:           y      0.0      1.000   0.000    1.000        -1.960
   Coef.Upper.95
1:         2.197
2:         1.960
3:         1.960
```

- This issue would be resolved using an interaction term between x and y, which would lead to a perfect prediction of XOR.
- However, in typical settings, it's not obvious when adding an interaction would be helpful.
- Neural Networks can be considered a way to automatically build better connections (e.g. interactions, combinations of functions, etc.) in complex data.

Fitting a Neural Network to AirBnB's Prices

```
library(nnet)
mod.nnet <- nnet(formula = as.formula(formula.price.lat.long),
    data = dat[get(training.row.name) == TRUE, ], size = 5)
```

```
# weights:  41
initial  value 654983004.544586
final  value 652297576.000000
converged
```

```
pred.nnet <- predict(object = mod.nnet, newdata = dat[get(training.row.name) ==
    FALSE, ])
rmse.nnet <- my.rmse(predicted = pred.nnet, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
print(rmse.nnet)
```

```
[1] 170.1958
```

Interestingly enough, the neural network does not perform well in this case. That was traditionally the issue. It is only through sharply increased complexity that deep learning applications can generate successful results.

Ensembled Methods

- Now you know a variety of interesting and competitive methods for supervised learning.
- Which one is best will depend on the circumstances – there is and cannot be any single algorithm that outperforms all others in all settings (the No Free Lunch theorem).
- However, there is nothing stopping you from **combining your predictions** to generate more of them.
- An **ensembled model** uses the predictions of other models as inputs to its own.
- This can be as simple as **averaging** the predictions of the other models.
- You can also use the predictions of the other models as inputs to a weighting model, e.g. as predictors in a linear regression.

An Ensembled Model of AirBnB's Prices

```
pred.ensemble <- rowMeans(cbind(pred.rf, pred.xgb, pred.svm))
rmse.ensemble <- my.rmse(predicted = pred.ensemble, actual = dat[get(training.row.name) ==
    FALSE, get(price.name)])
rmse.ensemble
```

```
[1] 72.66922
```

Getting Better Models

- Work on identifying **better features**.
- Implement and evaluate a **variety of methods.**
- Then, for promising models, **tune the parameters** using cross validation.
- Finally, when a small number of competitive models are in place, consider **ensembled methods** to combine their strengths and mitigate their weaknesses.
- The ensembled models may themselves involve complex weightings, modeling techniques (e.g. Gradient boosting applied with the inputs being the predictions of a neural network, a random forest, and a gradient boosted model), and tuning of its own parameters.

A Summary of the Methods

This is just my opinion!

Show 10 entries Search:

Method	Class	Outcome Data	Method of Inference	Numeric Predictions	Classifications	Special Features	Limitations
Linear Regression	Classical Model	Continuous Measures	Coefficients	Yes	Only by applying thresholds	Inferences, confidence intervals for the coefficients, interpretability	Linearity assumption, requires more rows than columns, categorical inputs create additional columns
Logistic Regression	Classical Model	Binary Data	Coefficients, Odds Ratios	Yes	Yes	Inferences, confidence intervals for the coefficients, interpretability	Logistic assumption, requires more rows than columns, categorical inputs create additional columns, Effective sample size based on smallest category

Method	Class	Outcome Data	Method of Inference	Numeric Predictions	Classifications	Special Features	Limitations
Cox Proportional Hazards Regression	Classical Model	Censored Survival Data	Coefficients, Hazard Ratios	Yes	Only by applying thresholds	Inferences, confidence intervals for the coefficients, interpretability	Linearity assumption, requires more rows than columns, categorical inputs create additional columns, Effective sample size based on smallest category
K Nearest Neighbors	Simple Machine Learning	Numeric or Categorical	None	Yes	Yes	Very simple to implement, adapts to any distribution, K parameter can be tuned	High variability, poor predictive performance relative to other methods
Ridge Regression	Regularized Regression	Continuous, Binary, or Censored Survival Data	Coefficients / Odds Ratios / Hazard Ratios	Yes	Only by applying thresholds	Good for inference, Regularization avoids overfitting by shrinking the coefficient estimates, Shrinkage parameter can be tuned	Poor predictive performance relative to other methods

Method	Class	Outcome Data	Method of Inference	Numeric Predictions	Classifications	Special Features	Limitations
Lasso Regression	Regularized Regression	Continuous, Binary, or Censored Survival Data	Coefficients / Odds Ratios / Hazard Ratios	Yes	Only by applying thresholds	Good for inference, Regularization avoids overfitting by shrinking the coefficient estimates, Shrinkage parameter can be tuned, Lasso can perform variable selection, can work with a large number of predictors relative to the sample size	Poor predictive performance relative to other methods
Decision Trees	Tree-based Models	Numeric or Categorical	Tree Diagrams	Yes	Yes	Creates interpretable models for decisions	Results can change drastically with small changes in the data, poor predictive performance
Random Forest	Tree-based Models	Numeric or Categorical	Limited: Variable Importance Measures	Yes	Yes	Excellent predictive performance, highly adaptable, Parameters can be tuned	Slow running time due to bootstrapping. Considered a black box algorithm

Method	Class	Outcome Data	Method of Inference	Numeric Predictions	Classifications	Special Features	Limitations
Gradient Boosting	Tree-based Models	Numeric or Categorical	Limited	Yes	Yes	Excellent predictive performance, highly adaptable, Parameters can be tuned	The results depend a lot on finding the right combination of the tuning parameters. Considered a black box algorithm
Support Vector Machines	Support Vector Machines	Numeric or Categorical	None	Yes	Yes	Good predictive performance, kernals allow for numerous kinds of classifying boundaries (e.g. radial)	Selection of kernals is improved by understanding the trends, but the data are highly complex. Considered a black box algorithm

Practical Considerations

When you pursue machine learning as part of your work with an organization, it always requires a balance among:

- **Accuracy**: More is better, but the improvements need to be worth the investments you'll make in time, computing power, labor, etc.
- **Sample Size**: Collecting more data is also better, but you may have to work with less than you'd like to have.
- **Computational Complexity and Running Times**: Running the computer (e.g. with more Bootstrapped samples) will likely lead to better results, but it has to be balanced against the time and costs.

Machine Learning Competitions

- Most competitions in Machine Learning aim exclusively for predictive accuracy.
- There is no penalty for exhaustively running the computer or pursuing every angle to achieve marginal improvements.
- In fact, some competitors work so hard that they **overfit the specific testing set** used in the competition. This can lead to hidden biases that degrade the selected algorithm's performance on other, equally valid testing sets. I have seen candidates who appeared to be winning a competition ultimately fare poorly when the results were evaluated on a different testing set.

For this reason, many competitions make a point of hiding the testing set that is used for evaluation.

A More Practical Machine Learning Competition

- Rather than exclusively focusing on accuracy, it is more fair to consider its balance with the other considerations.
- What is the best mix of accuracy, sample size, and computational time? That depends on what is valued.
- However, a competition that optimizes for the best balance may be more reflective of the real world.

Your Midterm Project

- Your teams will build Machine Learning approaches to classifying images.
- The goal will be to find the method that best optimizes for a combination of accuracy, sample size, and computational running time.
- This is an opportunity for you to evaluate the merits of different approaches, all while considering the practical aspect of working with these tools.